BoxView

Debugger for DSP56xxx

BoxView for DSP56xxx User's Guide,
September, 2003

Domain Technologies, Inc.
811 East Plano Pkwy, Suite 115
Plano, Texas 7507**4**
Tel.: (972) 578-1121
Fax: (972) 578-1086
E-mail: support@domaintec.com
Web page: http://www.domaintec.com

TABLE OF CONTENTS

CONTENTS

CONTENTS

CONTENTS

CHAPTER 1 - Introduction

The DSP56xxx development system consists of the following components:

- Target hardware - (DSP56xxxEVM, SB56K, LINK-56K, or other))
- BoxView - Debugger Graphical User Interface
- BoxServer (optional) - server used for remote and multiple user access

This manual describes the second component of the above configuration. A description of the other components can be found in separate documents.

Software components need to be installed first. The system software requires a 32-bit operating system, either Windows 95/98 or Windows NT. The system software cannot be run on earlier versions of the Windows PC operating system (Win 3.1, 3.11), even if the Win 32s option is installed.

1.1 - Quick Installation

The software is supplied on the installation disk. The installation program, setup.exe, will automatically create all needed directories, copy the required files to those directories, and create a BoxView program group .

The installed software can be removed from the host computer using standard Windows' procedures (Add/Remove Programs from the Control Panel).

BoxView can be invoked directly for different target configurations without any additional changes.

1

CHAPTER 2 - Tutorial

Installation will create following shortcuts in the BoxView program group:

Fig.1 - Shortcuts created by setup

- BoxView shortcut configured for the debugger execution.
- BoxView Help — this will invoke Help system

The default configuration file is called BoxView.ini. Multiple configurations can be created within the program folder. A debugger configuration saved in the configuration file can be specified with the command line option -s within the shortcut properties. Custom configuration will have different command lines.

After the BoxView program is invoked, it will initialize itself and create the following set of windows:

Fig.2 - Initial windows layout

Since the BoxView software can support various DSPs, the window contents cannot be initialized before selecting the target device.

If the target device has been specified correctly in the .ini file, then a simple CONNECT command can be to initialize the connection with the target device. The initialization process will take a few seconds. For customization or connection to a different target, use the Open Connection dialog (available from the Options pull down menu).

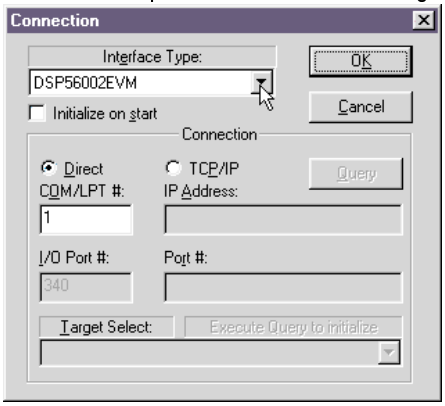

Fig.3 - Open Connection dialog

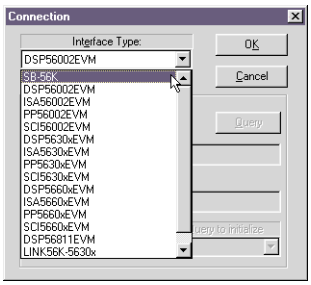

Fig.4 - Available Interface Types

In order to initialize the target, a proper Target type needs to be selected and an LPT# or base I/O address specified. The above dialog also allows the user to select remote target DSPs. The remote target initialization procedure involves selecting BoxServer's IP address (default: boxserver.softbox.com), performing the "QUERY" command to find out the number, and type of devices on the remote end. After selecting the desired device from the list, the system will connect and initialize its operation. Used devices will show IP addresses of the users connected through the BoxServer.

2.1 - Command line parameters

To allow target configuration at program startup, command line parameters can be specified. All target reference options can be also set within the Open Connection dialog (Options pull-down menu).

For multiple configurations, it is recommended to use custom file names for the system “INI” file.For multiple configurations, it is recommended to use custom file name for the system “INI” file.

-Sname	File name to save windows settings on exit

2.2 - Function keys

Defined standard function keys:

F1	Help
F5	Go
Shift-F5	Halt
F7	Go to the cursor (Code window only)
F8, F11	Step
F9	Toggle breakpoint at the cursor (Code window only)
F10	Next (jump over subroutines)

2.3 - Mouse Operation

A left-click sets the cursor within the window, a right mouse click invokes a local window pop-up menu, which is used to set the local window’s options.

To facilitate running the target code, special mouse processing has been added to the Code window:

Double left-click toggles a breakpoint (same as F9 key).

2.4 - Sample session

Select the target type that the PC is connected to. For the EVM target, which is referred to as the DSP56xxxEVM, a COM number also needs to be selected. For the ISA Host Interface adapter, an I/O base address needs to be entered.

If the target is installed on a remote workstation, a connection needs to be specified through the TCP/IP interface. After entering the IP address or alias name and the port number used for the socket connection, the Query button will initialize communication with the server. The Target Select box will show the available targets at the remote site.

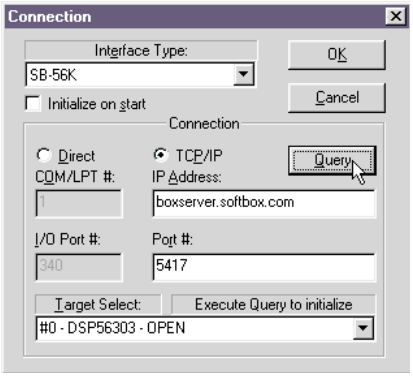

Fig.5 - Remote Connection to Server

When the OK button is pressed in the above dialog, the system will start the initialization of the target processor - local or remote.

If a problem is detected during interface initialization, an error message will be displayed, and the Open Connection procedure will need to be repeated.

After a connection to the target is initialized, the debugger will display data from the DSP.

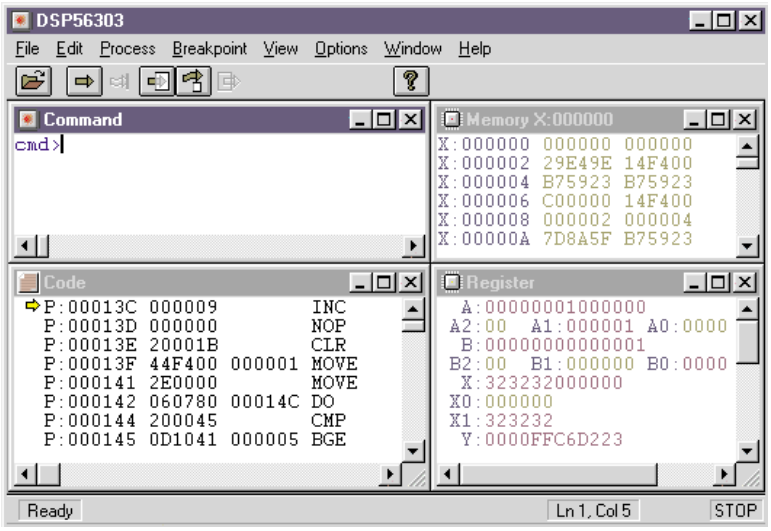

Fig.6 - After System Initialization

Next, a sample program can be loaded into the DSP. To simply load an object file, the "load" command can be used, but to specify additional load options, the Load File dialog should be used (pull-down menu LoadFile):

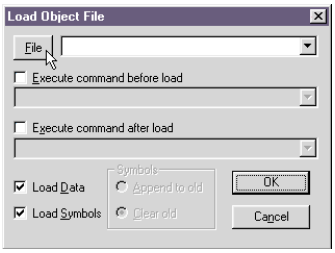

Fig.7 - Load File Dialog

The details of the above dialog are discussed in the following chapter. For the purpose of the tutorial, we will need only to select the file name with the "File" button. This will open a standard Windows file selection dialog:

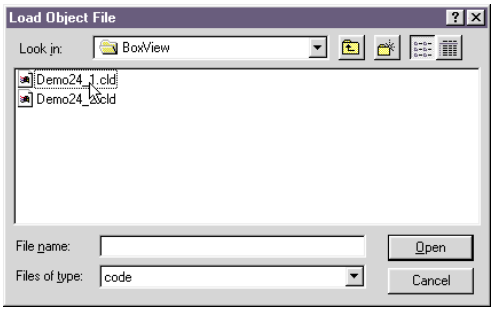

Fig.8 - File selection dialog

We will select a sample ASM program — demo24_1.cld

2

- Set a breakpoint at “mainloop” with the “BREAK LOOP” command within the command window.
- Start code execution with GO command
- Display data memory using symbolic entry: DISPLAY VALUE_A2

After those steps, the debugger windows should look as follows:

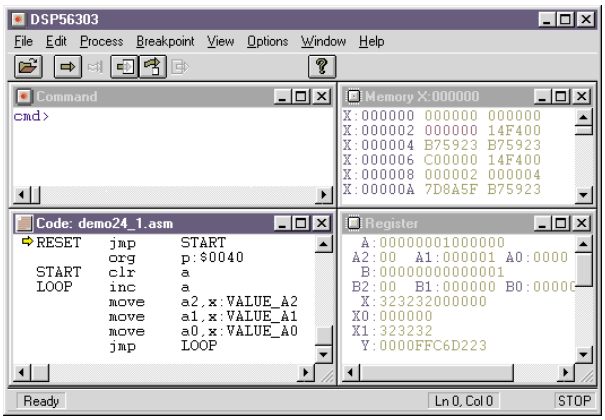

Fig.9 - After Loading

Some of the above commands could also be executed using the “Symbols” dialog. This dialog can stay open for the whole debugging session. To open this dialog, use the View—Symbol pull-down menu.

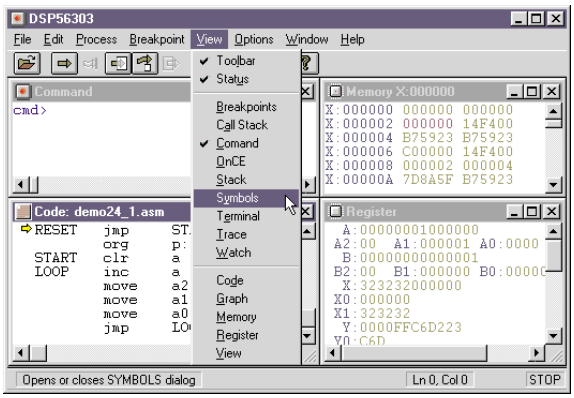

Fig.10 - Selecting Symbols Window

The symbols dialog allows the user to set breakpoints, add items to the Watch window, set display address for the Memory and Code windows, and start program execution from a temporary breakpoint.

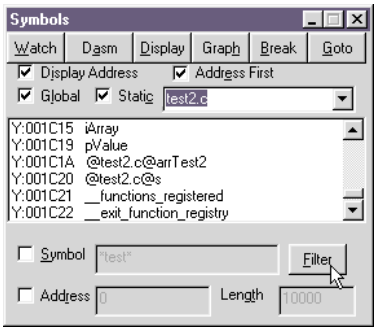

Fig.11 - Symbols Dialog

Additional features of the above dialog are described in the following chapter.

To execute the program with “automatic” screen refresh, set a breakpoint at address P:42 with the “show” attribute. This can be done with a double-click or shift-F9. Also the “show” attribute can be set with the Software Breakpoint dialog. The address p:41 is inside the program loop. Every time program execution stops at this address, the register and memory contents are updated.

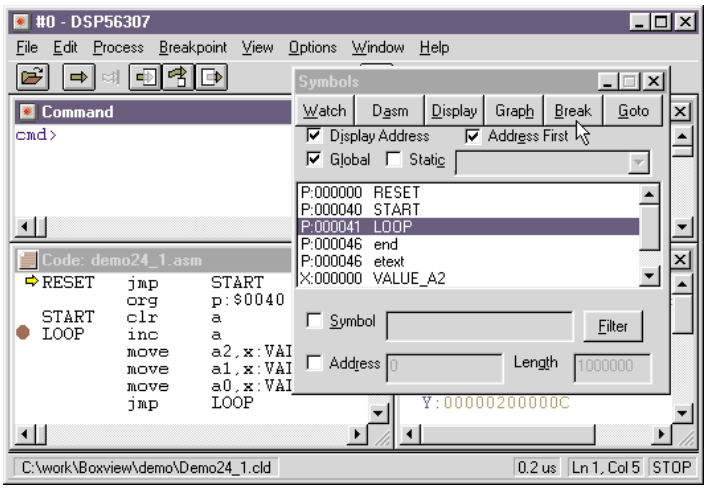

Fig.12 - Setting the Breakpoint

Variables can be added to the Watch window using the Watch command, or by pressing the Watch button within the Symbols window.

Complex variable (arrays, data structures, pointers) can be expanded or collapsed by double-clicking on the line.

To remove an item from the Watch window, press the Del key when cursor is on the line to be deleted.

Run code in the continuous step mode (Shift-F8 or Process—ContStep). The last selected Code window will follow the Program Counter (PC), opening the correct source file for the display. Other Code windows will show the PC (right yellow arrow) only if it is within the lines displayed.

The following figure shows a few other windows opened. All windows will be described in the following chapter.

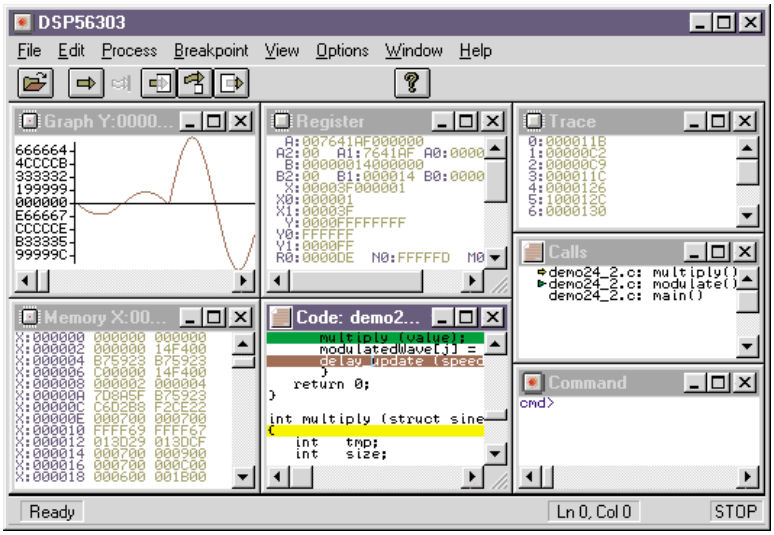

Fig.13 - Multiple windows opened

2

CHAPTER 3 - BoxView Reference

3.1 - Calls Window

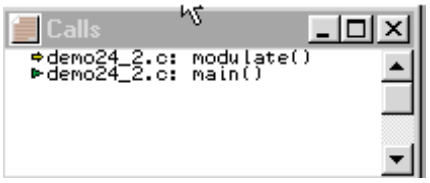

Fig.14 - Call stack window

The Calls window shows the call stack. It also allows the user to change the program frame pointer used for variables display. If the frame is different from the real one, a green triangle is displayed. This will affect local variables displayed within the Watch window

3.2 - Code Window

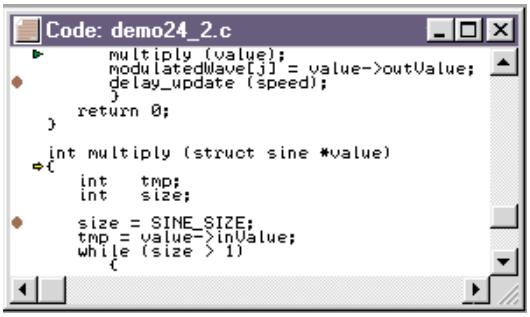

Fig.15 - Code window in source mode

The Code window displays the debugged program in one of three modes: source, assembly or mixed. Display modes can be selected either from the window local menu (by right-clicking within the code window), or by entering SRC, ASM or MIX commands within command window.

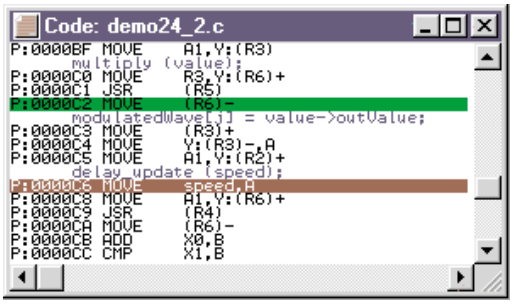

Fig.16 - Code window in mixed mode

Below is the layout of the local menu for the Code window:

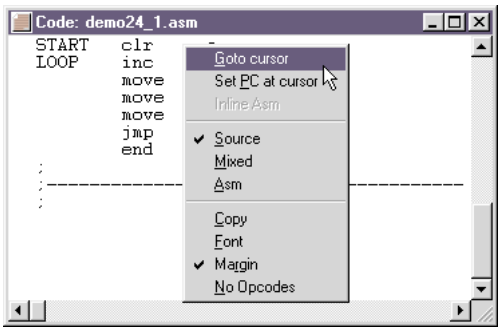

Fig.17-Code window's local menu

Local menus can be activated with the right-mouse button. The following functions can be accessed from this menu:

- execute code to the cursor (possible only for the code executed out of RAM). After reaching the desired location, a temporary breakpoint is automatically removed. The same effect can be accomplished with the F7 function key.
- invoke the inline assembler to modify the currently selected line of code
- set the mode of the Code window: Asm, Mix or Src. This can be also done with commands from the command window (ASM, MIX, SRC). If the command is executed from the command window, it will affect only the last selected code window (in the case where multiple code windows are open).

- change font size for this window. The global font selection is available from the main pull-down menu: Options—Font.
- disable/enable display of the opcode values. If the option “No Opcodes” is checked, no opcodes will be displayed, so the code can be displayed in the smaller window.
- enable/disable selection margin. With the selection margin enabled, small icons indicate the attributes of the particular line within the code window. If the margin is disabled, attributes are indicated by the background/foreground color of the displayed code line.

Following attributes are displayed with icons and/or colors:

- current PC value: yellow arrow
- selected frame: green triangle
- SW breakpoint: dark red circle
- SW breakpoint with “show” option: light red circle
- disabled SW breakpoint: red empty circle

3.3 - Command Window

Fig.18 - Command window

The Command window allows the user to enter text commands. It also saves a history of executed commands. Old commands can be edited and/or re-executed. Pressing the Enter key when the command line is empty will repeat the last command.

3.4 - Memory Window

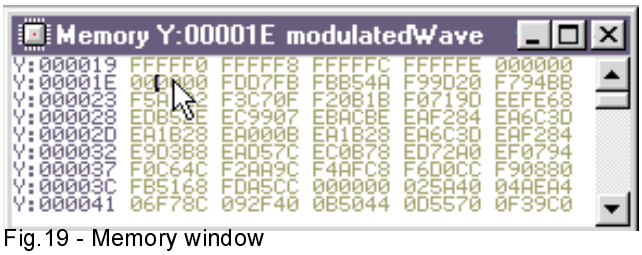

Fig.19 - Memory window

The Memory window allows the user to display the contents of the DSP memories. The starting address to display can be set from the Command window, using Display command, or from the Symbol window by selecting the symbol and pressing the Display button.

The starting address can be modified either with the command DISPLAY, or by directly editing the address field. The memory contents can be also directly edited within the window. The highlighted portions of the memory window can be copied to the Windows clipboard, and pasted to any other application.

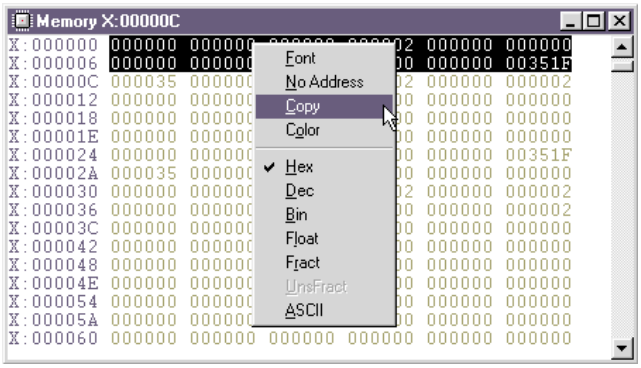

Fig.20 - Memory contents window

The local menu of the memory window (right-mouse button) allows the user to disable the address display, so more data can be shown within the window. The current address, where the cursor is located, is displayed within the window tile. If the address corresponds to a valid symbol, the symbol name will also be displayed at the window title.

3.5 - Register Window

Fig.21 - Registers window

The Register window displays the contents of the DSPs core registers. The number and the format of the registers displayed depends on the DSP type. The value of any core register can be modified with the CHANGE or SR commands, or by direct editing of the register value.

BoxView allows for multiple Register windows to be displayed.

3.6 - Graph Window

The Graph window allows the user to display the contents of memory in a graphical format. A single pixel on the horizontal axis represents one memory location. The vertical axis represents the memory value. The vertical axis can be scaled with the up/down arrows The scale can range from a single increment per pixel, up to a maximum memory value per graph height. There is also a logarithmic scale available.

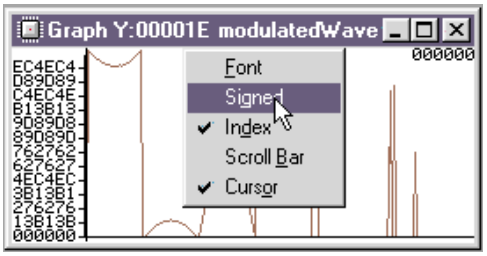

Fig.22 - Graph window

The local menu available with the right-mouse button, allows for following controls:

- Change font size - this affects index and value at the cursor display
- Sign toggle - allows for formatting the graph in signed or unsigned mode
- Index toggle - enables/disables display of the memory value scale on the left side of the window
- Scroll bar toggle - displays/hides the horizontal scroll-bar
- Cursor toggle - enables display of the cursor. The value at the cursor position is displayed in the upper right corner of the window. The position of the cursor can be set with mouse or right/left cursor keys.

3.7 - Symbols Window

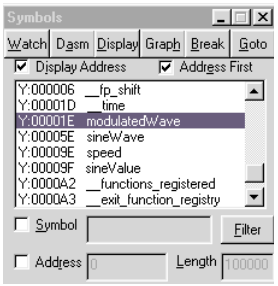

Fig.23 - Symbols window

The Symbols window displays the global symbols available in the debugged application.

The format of the display can be changed with two options:

The Display Address check box enables the display of the numeric address value of the symbol.

The Address Firstcheck box controls if the address of the symbol is displayed before or after the symbol name. This controls the sorting of the list entries.

Here are five operations which can be performed on the selected symbol:

Watch	adds the symbol to the Watch window
Dasm	sets the starting address of the Code window
Display	sets the starting address of the Memory window
Graph	sets the starting address of the graph window
Break	inserts software breakpoint at the symbol address
Goto	inserts temporary breakpoint at symbols address and starts execution
	Displayed symbols can be also filtered with two optional filters:
Symbol	will select all the names matching the control string. The * character is a wild card matching any number of any characters.
Address	will display the symbols within the specified address range. The starting address can be in either of two formats: with and without memory space prefix. No space prefix will allow all memory spaces to match the address range.

3.8 - Trace Window

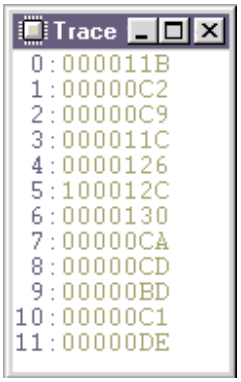

Fig.24 -Trace window

The Trace window shows the DSP execution history. For the DSP563xx devices it will show discontinuity stack, i.e. Pairs of addresses, when the program flow is interrupted. The first address in the pair indicates the last address of the interrupted flow, and the second one shows the address to which the Program Counter was changed with the JUMP, BRANCH, JSR, etc. instruction. Aborted jumps are marked with 1000000.

For the DSP560xx and DSP561xx devices, the Trace window will show the last five executed instructions.

3.9 - Watch window

The Watch window allows the user to display selected symbolic items or expressions. Items can be added to the watch window with the WATCH command, or by pressing the Watch button within the Symbols window. To delete items from the watch window, either press the Del key on the keyboard, when the cursor is over the line to be deleted, or use the WATCH command.

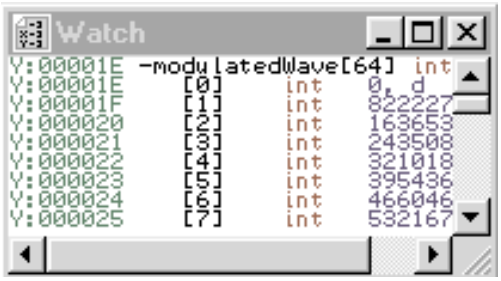

Fig.25 - Watch window with sample data

3

If the watch contains an element which can be expanded (array, structure, union, etc.) a + will be displayed in front of the symbol name. Expanded symbols will have a - character, indicating, that the item can be collapsed. The watch line properties can be edited through the local dialog available with the right-mouse button.

3.10 - Open connection dialog

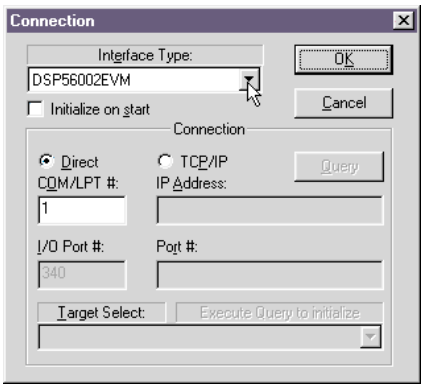

Fig.26 - Open connection dialog

The Open Connection dialog allows the user to select the target for the BoxView operation. This dialog is accessible from Options pull-down menu (Open Connection menu item). The primary item to be selected is the Interface Type.
The connection type can be either Direct or Remote over the TCP/IP network (optional BoxServer application is required).
For a direct connection, depending on the interface type, the proper port needs to be selected. It can be one of the COM ports, one of the printer ports, or the base address for the ISA interface card.
For a TCP/IP connection the only required parameters are the address of the server, and port # of the IP server to be connected to. To get a list of available targets from the server, use the Query button. After the query is completed, a list of the available target devices will be displayed in the Target Select list.
The Initialize on start check box, if checked, will cause auto target initialization on next BoxView invocation.

3.11 - Startup Options

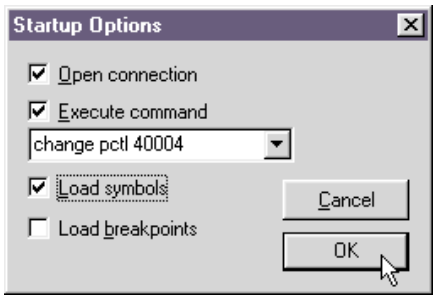

Fig. 27 - Startup Options

This dialog is available from the main pull down menu: Options — Startup.The options checked within this dialog control the program's actions during initialization:

- Open connection - will initialize the connection to the target, as it was last specified by the Open Connection dialog.
- Load symbols - will reload the last used symbol table. This option is disabled if the open connection is not active.
- Load breakpoints - will reload the software breakpoint table. The breakpoint table is saved during program termination, just before clearing all software breakpoints on program exit.

3.12 - Load File Dialog

Load File dialog provides options necessary for uploading the code to the target DSP. Selection of the file is available with the File button. This will invoke a standard Windows file browser dialog. The main check-boxes are: Load Data and Load Symbols. Disabling of data loading can be useful for debugging applications which are loaded to the DSP through another interface.

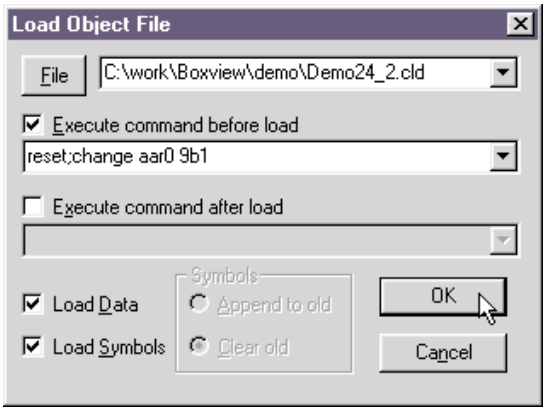

Fig.28 - Load File dialog

The Execute command before load check-box enables an editable field, where the user can specify the commands required to enable access to the DSP memory, or set any initial conditions.

The Execute command after load check-box enables an editable field, where the user can specify the commands to initialize certain variables, set breakpoints and start execution.

3.13 - Hardware Breakpoints

Hardware breakpoints are controlled through this dialog. The dialog is available through the main pull-down menu: Breakpoints — Hardware break.

This dialog follows the features available on the emulation circuitry for the specific DSP processor.

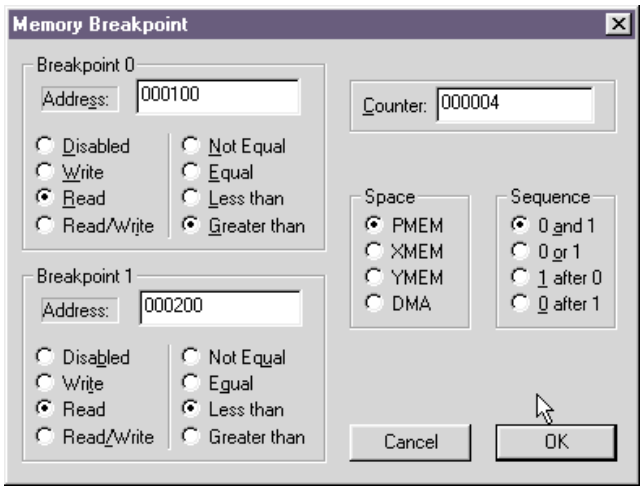

Fig.29 - Hardware breakpoint for DSP563xx

3.14 - Software Breakpoints

Software breakpoints can be set in the following ways:

- BREAK command
- By selecting the symbol and Break option within the symbols dialog
- By double-clicking on the code line, or with the F9 function key.

The action of the mouse-click or function key can be modified by pressing the Shift or Ctrl keys:

- Ctrl key - toggles the “show” attribute. The breakpoint with the “show & go” attribute will cause the code execution to continue, after updating the DSP resource windows.
- Shift key - will toggle the disable attribute. A disabled breakpoint is removed from memory, but it is still present in the symbol table for future use.

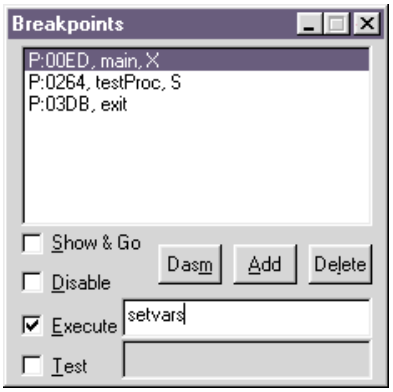

Fig. 30 - Software Breakpoints

A list of the breakpoints and attributes associated with every breakpoint, can be controlled from the Breakpoint Dialog (main pull-down menu: Breakpoint—Display).

The three buttons allow for following action actions:

- Dasm - will set the address of the code window to display the code at the breakpoint address.
- Add - allows to specify a new breakpoint address
- Delete - will clear the breakpoint and remove it from the list

The check boxes allow for direct control of eech breakpoint’s attributes:

- Show & Go - causes redisplay of memory, graph and register windows, and continuation of the DSP execution
- Disable - removes the breakpoint from memory
- Execute - will execute the specified command or macro
- Test - will invoke the expression evaluator. If the expression evaluates to non zero (TRUE), execution of the DSP code will be terminated, otherwise it will continue.

3.15 - User Buttons

BoxView allows for customization of the toolbar with user buttons. Buttons can be added with the dialog available at the main pull-down menu (Options-User Buttons).

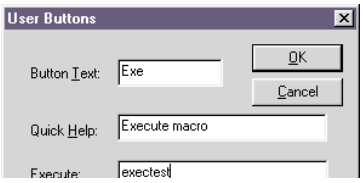

Fig. 31 - User Buttons Dialog

The dialogs allows for filling three fields:

- Button Text: this text will be stretched to the button size, and displayed on top of it. The text can be between 1 and 4 characters long.
- Quick Help: represents the text which will be displayed when mouse pointer is held over the button.
- Execute: represents the actual command or macro command name to be executed when the button is pressed.

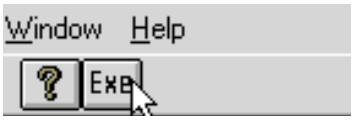

Fig. 32 - User Button with help

When the mouse pointer is held over the button, the quick help text will be displayed. The contents of the help text is user configurable.

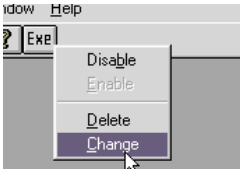

Fig. 33 - User Button local dialog

Every button can be changed or removed from the local menu available with the right-mouse button.

CHAPTER 4 - BoxView Commands

4.1 The command interpreter

The command interpreter is a recursive function which parses and executes commands. These commands are entered at the keyboard, or come from command files, macros or trap / breakpoint commands. The command interpreter runs as an infinite loop that executes the following steps:

1. Reads a line from the input source.
2. Breaks the string read into words, using a sequence of one or more of blanks, tabs, or commas as separators. Characters enclosed between double quotes are treated as a single word (this is only used in the PRINT command).
3. Replaces each occurrence of the symbol “%n” with the actual value of that argument, if the input source is a macro.
4. Evaluates the expression and substitutes the result (printed as one to four hexadecimal digit characters) in the expression string. Each time a new input source is set up, the command interpreter is recursively called to read the new source up to the end. If the specified command does not match any of the debugger commands the new command file is opened automatically. When the debugger reads input from a command file, the file is read until the end of the file is reached.

Macros, command files, and trap commands can be nested up to 20 levels.

The IF/ELSE/ENDIF commands select a group of commands to execute without changing the input source and without affecting the level count. The command interpreter ends when the ELSE or the ENDIF command is found.
Ending the top level (standard input reading) command interpreter, terminates the emulator program (after a confirmation request).]

Lines beginning with a colon character are ignored and can be used for comments.
The following key sequences are defined in keyboard input mode:

ESC	clears the input line
Ins	switches between insert and overtype
Del	erases the current character

Commands and arguments are not case sensitive, except in some particular cases which will be noted where applicable.

4.2 Command line expressions

Command line expressions contain one or more terms. Each term can be preceded by an unary operator, and is separated by binary operators. Expressions must be single words. Therefore, they cannot contain spaces. The syntax and semantics of expressions are very close to that of C language.

When an expression must be interpreted as a number, either to be used as a memory address, or a register, or value, the expression is sent to the expression evaluator. The expression evaluator returns the numeric value of the expression. Intermediate results are stored as 32 bit unsigned integers. The final result can be truncated to the lowest 8 or 16 bits as required.

4.2.1 Expression terms

In the following definitions:

- Lowercase words are generic names
- Upper case words are keywords, which can be in uppercase or lowercase inside the expression

4.2.2 Unary operators

The unary operators, which group right to left, are defined as follows:

Definition of unary operators:

Operator	Description
!	!*term* is evaluated as 1 if *term* has value 0, 0 otherwise.
~	~*term* is the bitwise negation of term, *i*.e. ~0x55 is equal to FFFFFFAA.

4.2.3 Binary operators

All binary operators group left to right, except the assignment operator, which groups right to left. In decreasing order of precedence, the binary operators are:

Definition of binary operators

Operator	Description
* / %	Multiplication, division and modulo (remainder).

+ -	Addition and subtraction.
>> <<	Logical right and left shift, i.e. 0xC>>2 is equal to 3.
== !=	Logical equality and inequality, evaluated as 1 if true and 0 if false, i.e. 5==3+2 is equal to 1 and 5!=3+2 is equal to 0.
>= <=	Logical comparison for greater, greater or equal, less, less or equal, evaluated to 1 if true and 0 if false.
&	Bitwise AND.
^	Bitwise EXCLUSIVE OR.
|	Bitwise OR.
=	Assignment operator. The left operand can be only an M, MM or #term. The right operand can be an M, MM, #term or the result of an expression evaluation.

4.3 Programming the emulator with the command line interface

4.3.1 Defining macro commands

Macro commands can be defined to carry out a sequence of individual emulator commands. For example, the watch data or action in breakpoint commands can be used to run a program starting from address 100, and see the value of the memory at address 50 when the program reaches the address 110. Alternatively, a macro command can be written to accomplish the same result.

1. To define a macro, use the de -m command (ref.“DEFINE”):

   ```
   de -m run_and_see
   change PC 100
   break P:110
   go
   Print x:50      ; display memory at address x:50
   endm
   ```

   The first line defines the macro name “run_and_see”; the last line indicates ends the macro command.
2. To run a previously defined macro, use the do command (ref.“DO”):

   ```
   do run_and_see
   ```

3. To define the same macro with the possibility of defining the address at each execution, specify the parameter “%1” in the macro definition:

   ```
   de -m -f run_and_see
   change PC 100
   Break  P:110
   go
   Print   X:%1
   endm
   ```

4. To execute this macro, type:

   ```
   run_and_see 40
   ```

   The macro will be executed as before, replacing parameter %1 by its specified value: 40.

5. To list the contents of the macro commands, use command LISTSYMBOL :

   ```
   li -m run_and_see
   ```

6. To store your macros in a file on disk, use the command ar (ref.“ARCHIVE”):

   ```
   ar -p -m file.mac
   ```

   This command stores the macro in the file named “file.mac”. This is an ASCII file which can be modified with any text editor.

7. To retrieve the macro from file, type:

   ```
   ar -g -m file.mac
   ```

4.3.2 Programming

Variables and programming statements can be included in macro-commands. For full details on programming inside macros, please refer to description of the commands: FOR, IF, and JUMP.

Up to 30 variables can be user defined from “v0" to ”v1d" hexadecimal address. These variables can be set, tested, or used in programming statements.

Expressions must not contain spaces. For example, the line:

```
v0  =  0
```

is incorrect. The correct format is: v0=0

Using the same macro as above:

1. To run the program 10 times and see the following byte each time, use the define command (ref.“DEFINE”):

   ```
   de -m -f run_and_see
   change PC 100
   break P:110
   for v0 1 A 1
   go
   print x:(%1+v0)
   endfor
   endm
   ```

When typing:

```
do run_and_see 40
```

the commands between FOR and ENDFOR will be executed 10 times (i.e. A times in hexadecimal, note that parameters are in current base). Command “print X:(%1+v0)” will be interpreted as “print x:41", then ”print x:42", and so on.

2. To execute out a test program:

   ```
   de -m -f run_and see
   Reset
   Change pc 0
   Break  p:110
   v0=1
   .loop
   go
   print x:(%1+v0)
   v0=v0+1
   if (v0<=A) jump loop
   endm
   ```

3. To define a label use the de command (ref. “DEFINE”).
4. To jump to a label use the jump command (ref. “JUMP”).

Each macro command can have local variables. The local variables names start with the “vl”, so valid names are vl0..vl1a.

4.3.3 Memory and register access

To display the memory location in hexadecimal format and the memory value in decimal format, define a macro as follows:

```
de -m -f run_and_see
change PC 100
break P:110
for v0 1 A 1
go
print -n -f "%x = " X:(%1+v0)
print -f "%d" X:(%1+v0)
endfor
endm
```

The PRINT command gives a formatted display.

1. To obtain the data in P memory at address 10, type or use in a macro:
   v0=P:10
2. To obtain the data in X memory at address 10, type or use in a macro:
   v0=X:10

Memory and register access syntax is fully explained in the PRINT command description.

4.4 File management

Channels are used to store the results of programs. In the BoxView debugger, the channel feature is similar to file management in C or other high-level language: channels can opened, read from, written to and closed.

Channels may be opened as either input or output, but not both; they can contain ASCII or binary values. All values must have the same length. Channel syntax is fully explained in the OPEN and CLOSE command description. (Ref. “OPEN” and “CLOSE”).

See the following example:

1. Edit a file called “address”, containing the addresses to check.
   Since addresses are 16-bits long, use a file that contains 16-bits long values. In ASCII mode, each value must be separated by spaces, tab or end of line. Suppose the “address” file is:
   0000 0100 0105 0107 0168 00a4 0078 0123 0150 0120
2. To open the file “address” for reading, using channel number 1, reading 2 bytes each time, type:
   open 1 address input ascii 2
   Output values are stored in a binary file “result”.
3. To open file “result” for writing, using channel number 2, writing each time 2 bytes, type:
   open 2 result output binary 2
4. To read and write from/to a channel, use the “#” character followed by the channel number. For example, “#1" means one value, read from or written to the file associatedwith channel 1.
   The macro becomes:
   de -m test_auto
   open 1 address input ascii 2
   open 2 result output binary 2
   for v0 1 A 1
   go
   eval v1=#1
   eval #2=x:v1
   endfor
   close 1
   close 2
   endm
   Expression “eval v1=#1" writes the 16-bit value read in file ”address" (channel #1) to variable v1. Expression “eval #2=x:v1" writes the contents of the memory at location ”v1" in the memory space X to the “result” file (channel #2).
   By running this macro and checking the contents of the file “result” versus a reference version each time the program is changed, you have a non-regression procedure to test program evolution.
5. To test for the end of file, open the file as input and use the @EOF command.

4.5 General notes

The following notes describe the syntax and general concepts used in the description of commands.

1.

The debugger commands can be typed in either lower or upper case letters. Command arguments must be separated by at least one space or comma.

2. The names of commands can be written in full or in abbreviated form: in the command descriptions, the shortest allowed abbreviation is shown in bold case.

3 The following syntax has been used:

The character "|" means "or".

An argument between square brackets is optional.

An argument followed by dots as in "value ... " means that the argument can be specified more than once.

Braces "{" and "}" are used to group arguments that are treated as a single construct.

4. Numeric values are interpreted and printed in the selected numeric base (DECIMAL, OCTAL, or HEXADECIMAL). Some commands require and/or print value in decimal base only; these will be explicitly indicated in the command description.

5. Iteration or long commands can be aborted by typing <ESC> key

6. The last command executed may be repeated by pressing the Enter key

7. All previous commands can be recovered with the up and down arrow keys, and modified using the left and right arrow keys, as well as , , ome, ...

8. Error messages produced by the debugger are self-explanatory and indicate the cause of the error.

4.6 Commands details

On the following pages all debugger commands are described in indetail.

? — Display evaluator result

Syntax ? expression
Menu None
Description This command displays the result from the expression evaluator. This is identical to the EVAL command, but it also displays the result in the command window. The expressions supported are any C type expression string.

Example
```
? r4+n4
```
See also EVAL

ADDR — Sets disassembly/source address

Syntax ADDR *address*
Menu View → Symbols
Description The ADDR command sets starting address for the last active CODE window.
address starting address value of the program to be displayed.

ALIAS — Set/display command alias

Syntax ALIAS [name[,"string"]]
Menu ---
Description The ALIAS command entered without any parameters will display all defined aliases. If only name is specified, the system will display the current definition of the alias associated with this name.
name alias name for the reference as the system command
string command string referenced by the name.
Multiple commands in the alias definition must be separated with a semicolon.Command parameters can be specified with the percent sign anda number (%1, %2, ...). Previously defined alias names can be used in other command strings.

ARCHIVE — Archive symbols and macros in files

Syntax ARCHIVE {-g|-p} {-a|-m} [-f] *file*

Menu None

Description The ARCHIVE command loads a symbol table of the specified type into memory from a file, or saves the symbol table in a file:

-a memory address type symbols

-m macros

-g (shorthand for "get") Symbols and definitions found in file are loaded into the symbol table. Unless the -f option is selected, symbols already defined in the symbol table are not redefined and an error message is issued.

-p (shorthand for "put") Symbols of the given type and their definitions are stored in the specified file. If the file already exists and option -f is not selected, a confirmation is requested before overwriting the file.

Symbols of memory addresses are saved in files in alphabetical order, one per line with their value in hexadecimal format.

Macros are saved in files as they are defined (that is, literally)

See also DEFINE, LISTSYMBOL, UNDEFINE

ASM — Change code window display to the assembly mode, on-line assembler

Syntax ASM [*address mnemonic]*

Menu Local menu of the code window

Description The ASM command without any parameters will change the display mode of the last selected Code window to assembly mode.

The ASM command with an address and mnemonic string will invoke the inline assembler at the specified address.

This command allows the user to write instruction opcodes with symbolic references to the target processor. The inline assembled opcodes will be placed in the program memory at the specified address, overwriting the data at this location. Interactive editing of the existing code is invoked from the code window's local menu (right mouse click).

Example

```
cmd>asm P:100 jump begin
```

See also DISASM

BASE — Base change base of numbers

Syntax BASE [x|o|d]

Menu None

Description The BASE command may be used to select the base used to display numbers.

"x" selected hexadecimal numbers
"o" selected octal numbers
"d" decimal numbers
The base currently selected is indicated by the prompt symbol of the debugger, according to the parameter of the BASE command.

Example

```
Base x
```

BAUD — Set/Get RS-232 baudrate

Syntax BAUD [value]

Menu Options → Baudrate

Description The BAUD command will change the speed of the RS-232 interface. If a value is not specified, the system will display the current baudrate. Value is a decimal number representing the requested speed. The system will accept any value, and than round down to the nearest valid value.

BEEP — Play sound notification

Syntax BEEP

Menu None

Description This command activates system's default sound (configured through "Sounds" in the Window's Control Panel)

BENCHMARK — Measure DSP execution time.

Syntax BENCHMARK [ON|OFF]

Menu None

Description Benchmark command allows to measure execution time of the target DSP. This feature is available only for the SB-56K emulator. Execution time is measured by the emulator's high-speed counter. Functionality of the counter requires -DE signal from the target DSP to be available. Counter is started when the target device exits debug mode, and it is stopped with the -DE signal asserted.

If the Benchmark mode is enabled, first few significant digits of the execution time will be displayed in the BoxView status area. The exact count, and exact execution time is returned with the Benchmark command without ON/OFF option.

ON Enables benchmark mode for the DSP controlled by the BoxView debugger

OFF Disables benchmark mode

Example:

```
BENCHMARK ON                ;enable benchmark mode

RUN 0x10                    ;execute 16 instructions

BENCHMARK                   ;display counter result

:00:00.000.000.5 (0x4)      ;value returned

R0UN 0x100                  ;execute 256 instructions

BENCHMARK                   ;display counter result

0:00:00.000.006.7 (0x32)    ;value returned

RUN 0x1000                  ;execute 4096 instructions

BENCHMARK                   ;display counter result

0:00:00.000.106.7 (0x313)   ;value returned
```

BREAK — Set/Clear/Display SW breakpoints

Syntax BREAK [ENA]|[DIS]|[OFF][addr[Xcommand][Texpr][s]]

Menu Breakpoint → Display

Description The BREAK command sets a software breakpoint at the specific address. If no parameters are specified, the system will display all breakpoints currently set. The OFF parameter will remove the breakpoint at the specified address, or all breakpoints if the address is omitted.

address	absolute address or symbol
ENA	enables specified breakpoint or all breakpoints
DIS	disables specified breakpoint or all breakpoints
Xcommand	specifies command name to be executed, when the breakpoint is reached.
Texpr	specifies logical expression to be tested. If the result is 0, target processor will be restarted.
S	refreshes all windows and restarts the processor.

CALLS — Display call stack

Syntax CALLS

Menu None

Description This command displays the current call stack in the command window. This represents the information displayed in the Calls window, but it allows the user to add information to the log file. The call stack is built using the information created by C programs. This information is not available for assembly programs.

Example

```
calls
```

See also LOG

CASCADE — Cascade windows

Syntax CASCADE

Menu Window → Cascade

Description Arrange all opened windows in the cascade format.

CD — Change directory

Syntax CD [directory name]

Menu None

Description The CD command changes the current working directory from within the debugger. You can use relative pathnames as part of the *directory name*. If you don't use a *pathname*, the CD command displays the name of the current directory. When it is implemented with the USE command, CD can affect any other command whose parameter is a filename, such as FILE, LOAD, and TAKE commands. You can also use the CD command to change the current drive.

Example:

```
cd c:
cd d:\csource
```

CHANGE — Change memory/register contents

Syntax CHANGE address | address range | register name, value

Menu Direct resource editing

Description The CHANGE command allows the user to change the contents of the memory range or the register.address memory address to be changed in the format: space:offset, where space is one of the allowed memory spaces for the processor (P, X, Y or L), offset absolute value of the address offset. Address can be also specified in the form of the application symbol.address range, which specifies the memory area to be changed to the value.
Range can be in either of two formats:start_address..end_address or start_address#count.Register name refers to one of the valid DSP registers (A0, SP, PC, etc.)

CLOSE — Close I/O channel

Syntax CLOSE *channel*

Menu None

Description This command allows the user to close the channel specified.channel must be a decimal number from 1 to 10.

Closing a channel that has not yet been opened does not cause an error. Once closed, a channel cannot be used for reading or writing data.

Example

```
close 1
```

See also OPEN

CLS — Clear screen

Syntax CLS

Menu None

Description The CLS command clears the contents of the Terminal window , and positions the cursor at the first line, first column.

Example

```
CLS
```

See also @CURX, @CURY, LOCATE

CM — Compare memory

Syntax CM *addr-1 addr-2* [count]

Menu None

Description The CM command allows the user to compare the contents of count data of memory at address addr-1, with the contents of count data of memory at address addr-2.

If count is not specified, it is assumed equal to 1.

For each comparison resulting in a difference, the addresses and the contents of the corresponding memory cells are printed.

Example

```
CM P:10 Y:100 20
CM x:10 x:100 20
```

COLOR — Set window colors

Syntax COLOR *window id value*

Menu Options → Colors

Description This command allows the user to change default colors in the specific window.

window represents the window name, as it appears in its title.

Id the number for the color type to change. There are different numbers of colors for different windows. This information is displayed in the colors setup dialog (Options-->Colors).

value hexadecimal value representing the background (high nibble) and foreground (low nibble). The color association with color values 0..15 can be seen in the Color Setup dialog. The default for 0xf (15) is white, default for 0 is black, but any color can be changed with above dialog

Example

```
Color memory 1 f2
```

CONNECT — Initialize hardware interface

Syntax CONNECT [interface type[,device[,[IpAddr][,port]]]]

Menu Options → Open Connection

Description The CONNECT command initializes the connection to the target processor. Parameters specify details of the connection. interface type can be one of the following:SB56K, LINK56K, DSP56002EVM, ... Device indicates device # in the scan chain to connect to. This is required for the multi processor systems. IpAddr defines TCP/IP address for the remote connection to the server.For the local connection this parameter needs to be empty. port indicates physical address of the target. In the case of a remote connection to the IpAddr this is the server's port number (by default 5417 is used).For the local RS-232 connection this is the COM port number (1, 2, ...). For the local I/O port interface, this is the I/O port base address (240, 340, ...).

COPY — Copy memory

Syntax COPY address_range mem_address

Menu None

Description The COPY command allows the user to execute a fast copy of the memory block specified by the address range to another location specified by mem_address.

@CURX, @CURY — Set disassembly addressge Down Code window

Syntax @CURX
@CURY

Menu None

Description The @CURX and @CURY commands are variables that return the current cursor position. CURX returns the current column value, and CURY returns the current line value.

Example

```
CM 10 100 20
CM x:10 x:100 20
V0=@CURX
print @cury
```

See also CLS, LOCATE

DASM — Set disassembly address or Scroll Page Down Code window

Syntax DASM [address]

Menu ---

Description The DASM command will set the starting address of the Code window to the address value. If no parameter is specified, this command will cause the Code window to scroll one page down. This command affects only the last selected Code window; other Code windows are not affected.

DEFINE — Define symbols and macros

Syntax

DEFINE -a [-f] *symbol* [space:] *value*
DEFINE -m [-f] *symbol*
...
...
ENDM

Menu

None

Description

In the first case, symbol refers to a memory address.symbol and value must be valid addresses.

The second case is used to define the macro called symbol.A macro is a collection of commands that can be executed via the DO command. The lines following the DEFINE command are read up to a line containing the keyword ENDM and form the macro body.

Notice that the macro body is stored in memory literally, without any interpretation, except that leading blanks are stripped, and lines beginning with the colon character (:) are discarded.Macro parameters can be embedded in macro body as “%n”, where n is a single digit between 0 and 7. They will be expanded at macro execution time with the literal value of the actual parameters passed to the macro.

- *symbol* can be any word composed of letters, numbers, "_" (underscore), "$" characters.
- Symbols are case sensitive.
- Unless the -f (shorthand for “force”) option is specified, if the symbol is already defined, an error message is issued and the definition of the symbol is not changed.
- If two address symbols have the same value, the most recently defined one will be used when the symbol table is searched for a value.
- Symbols are stored in memory for fast retrieval and alphabetical sorting. Each symbol requires memory that is allocated when the symbol is defined and released when the symbol is removed. Defining more symbols than allowed by the available memory will cause an “OUT OF MEMORY” error message.
- Address symbols may be used in any expression. The symbol will be replaced with its numeric value.

Example

```
define -a lab x:10
define -m LOADANDGO
reset
print "LOADING FILE %1"
load %1
break P:%2
go
endm
```

A macro, once defined, may be invoked using the DO command. In the above example, the LOADANDGO macro could be executed like this: do LOADANDGO prog_8 1234

The DSP will be reset, prog_8 loaded, run and finally stopped if address 1234 is reached.

See also ARCHIVE, DISASM, LISTSYMBOL, UNDEFINE.

DIR — List directory contents

Syntax DIR [directory name]

Menu None

Description The DIR command displays a directory listing in the display area of the COMMAND window. If you use the optional *directory name* parameter, the dubugger displays a list of the specified directory's contents. If you don't use the parameter, the dubugger lists the contents of the current directory.

DISASM — Display memory in mnemonic format

Syntax DISASM *addr* [count]

Menu None

Description The DISASM command allows the user to display the contents of memory at address addr, in symbolic format, i.e., in mnemonic assembler code.

If only addr is typed, the command iteratively displays the instruction at address addr. If the
key is then typed, it displays the next instruction, and so on.If before the
the new address is typed, the command starts disassembling instruction from the new address.To leave the command type "q".

See also ASM

DISCONNECT — Close connection to the target processor

Syntax DISCONNECT

Menu Options → Close connection

Description Close the hardware Interface.
The DISCONNECT command will disable the connection with the target processor. This allows the user to select another processor to be accessed (in case of the multiprocessor or remote access).

DISPLAY — Set memory address or Scroll Page Down Memory window

Syntax DISPLAY [address]

Menu View → Symbols dialog

Description The DISPLAY command will set the starting address of the Memory window to the address value. Address should have the following format: space:offset, where space is one of the allowed memory spaces for the processor (P, X, Y or L), and offset is the absolute value of the address offset. Address can be also specified in the form of an application symbol. If no parameter is specified, the command will cause the Memory window to scroll one page down. This command affects only the last selected Memory window; other Memory windows are not affected.

DM — Display memory in TERM window

Syntax DM address [end_addr]

Menu None

Description The DM command allows the user to display the memory contents on a word basis.

In the first case, the command displays the contents of memory at address addr.

In the second case, the command displays the contents of the memory block specified between addresses and end_addr (inclusive).

DO — Execute defined macro command

Syntax DO macro_name [actual_parameters_list]

Menu None

Description The DO command executes the macro macro_name with the specified parameters. The macro body is read by the command interpreter and commands are executed as if they were read from the keyboard.

Interactive commands, like ASM, take their input from the lines following the command in the macro body. They also work silently, i.e., they do not display any prompt. Before a line is read from the macro body, any instance of "%n" (where n is a single digit between 0 and 7) is substituted with the corresponding parameter taken from the actual_ parameter_list."%0" is the macro name itself, "%1" is the first parameter, and so on. Up to 7 parameters can be given in the parameter list. Parameters not defined are substituted with a null string.

Macro executions can be nested. Execution can be interrupted with <Esc> key

Example:

```
define -m LOADANDGO
reset
print "LOADING FILE %1
"load %1
break %2
go
endm
do LOADANDGO prog_8 1234
```

DR — Display register values in TERM window

Syntax DR [reg [last_reg]]

Menu None

Description The DR command displays the content of the DSP registers.

Used withoutany arguments, it displays all of the DSPs registers.

If a single register is specified, only that register will be displayed.

In the form: "DR reg last_reg" the registers between reg and last_reg (inclusive) are displayed, one per line.

@EOF — Test end of Input channel

Syntax IF @EOF (channel) JUMP *label*

Menu None

Description The @EOF command tests if the end of an opened file has been reached.

The function returns FALSE if the end of the file has not been reached, otherwise it returns TRUE. The function works like the C-language function, meaning that it returns TRUE when the program has tried to read after the end of the file; in this case, the value read is undefined.

Example

```
de -m test
open 1 %1 input ascii 2
.loop
v1=#1
if @EOF(1) jump endfile
print v1
jump loop
.endfile
print "end of file encountered"
close 1
endm
do test datas.2
```

See also OPEN, CLOSE

EVAL — Evaluate expression

Syntax EVAL *[lval =]expression*

Menu None

Description This command allows the user to evaluate any “C type” expression, with option to store the result in the “lval” destination. The destination can be any register, memory location or system variable (v0..v1d). Use the “?” command to display the result of the evaluation.

Example

```
EVAL R4=*ptr+4
```

See also ?

FM — Fill memory with pattern

Syntax FM address_range pattern

Menu None

Description The FM command fills the memory block indicated by the address range with the values indicated in pattern

Up to 8 values can be indicated. They will be repeatedly put in the memory in the same order until the block is filled.

Example

```
FM x:100#200 1234 5678
```

FOR — Loop command execution

Syntax

FOR *var start end* [step]
...
...
ENDFOR

Menu None

Description The FOR command loops from start value to end value in increments of step. The parameter var must be a valid variable (v0 to v1d).

The default value for increment is 1.

Example :

```
Set memory between 10 and 20 to zero
FOR v0 10 20
print x:v0
ENDFOR
```

FORCE — Stop or reset targetl

Syntax FORCE R | B

Menu None

Description This command allows the user to stop program execution (Break parameter), or to reset the target processor (Reset parameter).

Example

```
Force r
```

See also HALT, RESET

GO — Start executing target code up to address

Syntax GO [address]

Menu Process → GO, F5

Description The GO command executes code up to a specific address. If the address is not supplied, code will execute until it reaches one of the breakpoints, or it is stopped with the HALT command.

GRAPH — Display memory in the graphical format

Syntax GRAPH [mem_address]

Menu View → Graph

Description The GRAPH command will set the starting address of the Graph window to the mem_address value. If no parameter is specified, the command will cause the Graph window to scroll one page down. This command affects only the last selected Graph window; other Graph windows are not affected. If there is no graph window opened, the command will open a new window in the graph format.

HALT — Halt target processor

Syntax HALT

Menu Process → HALT, Shift-F5

Description The HALT command will place the target processor into debug mode, allowing the user to access the target's resources.

HELP — Get help on a specific item

Syntax HELP [keyword]

Menu Help, F1

Description The HELP command will invoke the help file associated with the debugger application. If keyword is one of the valid topics, it will be automatically displayed.

IDCODE — Read JTAG IDCODE

Syntax IDCODE

Menu ---

Description The IDCODE command will read and display IDCODE of the target processor. IDCODE is a 32 bit number representing the manufacturer, processor type and version.

IF — Conditional operator

Syntax

IF *expr command*

IF *expr*

...

[ELSE]

...

ENDIF

Menu None

Description The IF command conditionally executes one or more commands, depending on the value of expr. If value of expr is not zero, then:

- in the first case the single following command is executed.
- in the second case all following commands up to ELSE or ENDIF are executed.

If the value of expression is zero and ELSE is present, then only the commands between ELSE and ENDIF are executed.

IF commands can be nested up to 10 levels. Nesting more than 10 IFs will result in an error message.

If the command is read from the keyboard, the prompt displays the current nesting level by adding one colon for each level. Hitting will result in an exit from all nesting.

Example

```
de -m -f run_and_see
reset
break P:110
v0=1
.loop
go
print x:(%1+v0)
eval v0=v0+1
if (v0<=A) jump loop
endm
```

INPUT — Define file for data input

Syntax INPUT [#(file_number)] [address] OFF | TERM | filename [-dec | -fra | -hex | -bin] [-count]

Menu None

Description The input command is used to open a text file and pass data to the target DSP chip. The data is passed to the target DSP when the user's program reaches a software breakpoint. The text file lists the data sequentially.

The INPUT command with no parameters displays all open input files.

file#	file_number is the number of the file opened (multiple files can be opened simultaneously). The file number is optional, the range is 1 to 99.
addr	address is the address where the DSP breakpoint is halted in order to perform the file input (always in the P: space). Assigning the address of the DEBUG instruction in the INPUT command defines the direction of data transfer when user program reaches the DEBUG opcode. The address is optional if one of the direction bits in register R1 is set.
OFF	closes an opened input file. If a file is not specified, all opened files are closed.
TERM	uses the terminal/keyboard (instead of a text file) to input data to the target DSP.
fname	is the name of the file used for data input.
-count	specifies number of bytes read from INPUT file for every word to be written (1..4). This parameter applies only to the binary file format
-bin	specifies binary file format
-dec	specifies ASCII decimal data representation
-fra	specifies ASCII fractional data representation
-hex	specifies ASCII hexadecimal data representation. This is the default.

Parameters specified by the user's DSP program.

Since multiple files can be opened, your DSP program must specify the file number. This is accomplished by placing the file number in the most significant 8 bits of register X0.

The user program must also specify the number of words to transfer. This is accomplished by placing that number in the low 16 bits of register X0. The low 8 bits of register X0 are used for the 16-bit DSPs.

The user program must also specify the address where the data will be placed. This is accomplished by placing that address in register R0.

The user program must specify the address space (P:, X: or Y:) where the data will be placed. This is accomplished by placing a value in the register R1. A 0 is used for P:, 1 for X: and 2 for Y: memory space.

The direction of the data transfer may be specified (optional) by placing a flag in register R1. If the most significant bit (8000 hex) is 1, the direction is into the DSP. If the 14th bit (4000 hex) of R1 is 1, direction is out of the DSP. If neither bits are set, the address of the DEBUG instruction is used.

The breakpoint must be reached by a user induced DEBUG instruction..

Details on the text file

The text file is always ASCII, with the data listed sequentially. BoxView provides ways to simplify the file editing (for example, if one word must be send to the DSP repeatedly, you can have a repeat code instead of typing an infinitely long file.

The following are the codes that may be used in the ASCII file

Specifies the number of times a data word is repeated.

() Parenthesis are used to group words. If parenthesis are preceded by **#** the whole group is repeated. If **#** doest not precede the parenthesis, the group of words is repeated indefinitely.

Example 1
```
0.1  0.2  0.3  0.4
```
Send data words 0.1, 0.2, 0.3, 0.4.

Example 2
```
0.1#10
```
Send data words 0.1 10 times.

Example 3
```
0.1 0.2)#10
```
Send 0.1, 0.2, 0.1, 0.2 ....

Example 4
```
7fff)a
```
Send 7fff indefinitely

Example 5
```
1 2 3 5#2 (10 20)#3 (10)
```
send 1, 2, 3, 5, 5, 10, 20, 10, 20, 10, 20, 10, 10, 10, 10, ...

Examples on entering the input command

Example 1
```
INPUT
```
Display all currently open input files.

Example 2
```
INPUT #1 P:100 DATA.IO -dec
```
Open “DATA.IN” file, label it file number 1, and send the data to the DSP when the breakpoint at address P:100 is reached. Data in “DATA.IO” is in decimal.

Example 3
```
INPUT P:100 DATA.IN2
```
Open “DATA.IN2" file, label the file automatically, and send the data to the DSP when the breakpoint at address P:100 is reached. Data in ”DATA.IN" is in hexadecimal.

Example 4
```
INPUT P:100 TERM
```
When the breakpoint at P:100 is reached, read data from the terminal (keyboard) and send it to the DSP.

Examples of DSP code to support the INPUT command

Example 1

```
MOVE        $010010,X0  ; 16 WORDS FROM FILE NUMBER 1
MOVE        $0000,R0    ; PLACE DATA AT ADDRESS 0
MOVE        $0001,R1    ; PLACE THE DATA IN SPACE X:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 2

```
MOVE        $020012,X0  ; 18 WORDS FROM FILE NUMBER 2
MOVE        $0020,R0    ; PLACE DATA AT ADDRESS 20
MOVE        $0000,R1    ; PLACE THE DATA IN SPACE P:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 3

```
MOVE        $030080,X0  ; 128 WORDS FROM FILE NUMBER 3
MOVE        $0200,R0    ; PLACE DATA AT ADDRESS 200 HEX
MOVE        $8002,R1    ; PLACE THE DATA IN SPACE Y:
                        ; (INPUT DIRECTION)
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

JUMP — Go to label

Syntax JUMP [condition] *label*

Menu None

Description The JUMP command conditionally changes program's execution to the specified label if the condition is true (!= 0). If no condition is specified, then it performs an unconditional jump.Labels can be defined in a command file. They must start with a period(.) and can be up to 80 characters long. If present, a label MUST be the only entry on the line.

Example

```
JUMP display_regs
JUMP m(0)==2 error_label
.display_regs
.error_label
```

LISTSYMBOL — Display symbols and macros

Syntax LISTSYMBOL {-a[space] | -m}

LISTSYMBOL {-a[space] | -m} *pattern*

Menu None

Description The LISTSYMBOL command lists all the symbols of the given type (address or macro) in alphabetical order along with their definition (value). To list symbols in a specific memory space, specify the memory space as "x:", "y:" or "p:"

The expression pattern is built with normal symbol names and characters:

"*" matches any sequence of characters;

"?" matches anysingle character.

Example

```
li -a (displays X, Y and P space symbols)
li -ax: (displays X space symbols)
li -ax: lab (displays X space symbol "lab")li -m
(displays macros)
li -a:p _str* (displays P space symbols _str...)
li -a:x _std?? (displays P space symbols _stdin ...)
```

See also ARCHIVE, DEFINE, UNDEFINE.

LOAD — Load program and symbols

Syntax LOAD filename [P/X/Y]

Menu File → Load

Description The LOAD command loads an object file to the target processor and its associated symbol table into memory. The LOAD command is equivalent to the RELOAD and SLOAD commands. If the filename has no extension, the system will automatically add the .cld extension. The LOAD command clears the old symbol table.

LOG — Enable logging

Syntax LOG [filename | OFF]

Menu None

Description The log command allows the user to open or close the log file. All output to the terminal window is automatically saved to the log file.

LOCATE — Set cursor position

Syntax LOCATE *line column*

Menu None

Description The LOCATE command positions the cursor.line must be in the range 1 to 25.

column must be in the range 1 to 80.

Example

```
locate 3 15
```

MAIL — Send command to another BoxView application

Syntax MAIL dspID command

Menu None

Description The MAIL command allows to execute commands by another BoxView client connected to the same emulator through the BoxServer. This allows to initialize and start selected processors from the single script file.

dspId is a DSP number, as defined by connections on the JTAG scan-chain. The same number is used for the MDEVICE command.

command any of the BoxView command, which will be executed by the destination debugger.

Synchronization of the execution of the remote commands is possible with the WAIT MAIL command. WAIT will return control to the command script processor, after every mail receives its acknowledge.

The text output of the command is sent back and is displayed in the Terminal Window. If logging is enabled, text output will be also stored in the log file.

Example:

```
RESET
LOAD app1
MAIL 0 LOAD app2
MAIL 1 LOAD app2
MAIL 2 load app31
WAIT mail
MDEVICE 0 1 2 3
MGO
```

MAP — Set/Remove/Display memory mapping

Syntax MAP {ON/OFF/-r} {memBLOCK mode/-r}

Menu None

Description The MAP command sets mapping for memory address ranges. Enabling mapping option, will enable READ and/or WRITE accesses to selected memory areas, while undefined memory blocks will not be accessed by the emulator.

ON	enable memory mapping
OFF	disable memory mapping
-r	remove specified meory block or all blocks
memBlock	address range to apply mapping mode
mode	type of access for the specified block. Supported modes are: RAM, ROM, READ, WRITE

MDEVICE — Set/display device list for multi-DSP operations

Syntax MDEVICE [id1 [id2 ...]]

Menu None

Description The MDEVICE command will set or display a list of devices to be used for the multi-DSP operations. This command applies only to the multi-DSP emulators (SB-56K, etc.). Multi-DSP commands allow for simultaneous execution of the following functions: GO (command MGO), HALT (MHALT) and STEP (MSTEP). Simultaneous operation means that all specified devices are synchronized to the single edge of the emulation clock (JTAG's TCK signal).

The list initialization is required for the above three commands. Each device needs to have opened its own user interface (separate BoxView application). Control applications are linked to the target through the BoxServer, so they can be executed on single or multiple workstations.

Other multi-DSP related commands: MGO, MHALT and MSTEP

```
MDEVICE 0 3 4  ;select three devices for multi-DSP execution
MSTEP 4        ;execute 4 single steps on those 3 devices
MDEVICE 1 2    ;select other two devices on the scan-chain
MGO            ;start simultaneous code execution on devices 1 & 2
```

MEM — Set memory address or Scroll Page Down Memory window

Syntax MEM [address]

Menu View → Symbols Dialog

Description The MEM command will set the starting address of the Memory window to the address value. If no parameter is specified, the command will cause the Memory window to scroll one page down. This command affects only the last selected Memory window; other Memory windows are not affected.

MGO — Execute GO for multi-DSP configuration

Syntax MGO

Menu None

Description The MGO command will simultaneously start execution for all devices enabled with the MDEVICE command.

This command applies only to the multi-DSP emulators (SB-56K, etc.). If the program counter of one of the target devices is at the software breakpoint, the opcode at the breakpoint address will be first executed, before starting of all of the devices enabled for multi-DSP operation. This will allow for reinserting the software breakpoint before full-speed execution.

Other multi-DSP related commands: MDEVICE, MHALT and MSTEP

MHALT — Execute HALT for multi-DSP configuration

Syntax MHALT

Menu None

Description The MHALT command will simultaneously stop code execution for all devices enabled with the MDEVICE command.

This command applies only to the multi-DSP emulators (SB-56K, etc.).

Other multi-DSP related commands: MDEVICE, MGO and MSTEP

MIX — Change Code window display to the mixed mode

Syntax MIX

Menu Local code window menu

Description The MIX command will change display mode of the last selected Code window to mixed mode. If the source line information is not available, this command will have no effect.

MSTEP — Execute STEP for multi-DSP configuration

Syntax MSTEP [count]

Menu None

Description The MSTEP command will simultaneously execute a single step for all devices enabled with the MDEVICE command. The optional count allows the user to specify how many times the MSTEP command will be repeated.

This command applies only to the multi-DSP emulators (SB-56K, etc.).

Other multi-DSP related commands: MDEVICE, MGO and MHALT

NEXT — Execute next statement

Syntax NEXT [count]

Menu Process → Next, F10

Description The NEXT command will execute a single step, or jump over subroutine calls. The NEXT command can also be accessed with the F10 function key. Count specifies the number of statements to be executed before refreshing the debugger windows.

OPEN — Open I/O channel

Syntax OPEN *channel file mode format* [length]

Menu None

Description The Open command associates an input or output channel with a file, for subsequent reading/writing data from/to the file.

- *channel* can be a decimal number from 1 to 10.
- *file* may be the name of any accessible file or the reserved word TTY, in which case the keyboard or the terminal, whichever is appropriate, is used.

mode

- INPUT the channel is opened for reading from the file;
- OUTPUT the channel is opened for writing to the file;
- APPEND the channel is opened for writing at the end of the file.

format

- ASCII data read or written should be interpreted in ASCII for-mat;
- BINARY data should be interpreted in binary format.
- length expresses the number of bytes read from or written to the file. It can be 1, 2, 3 or 4; default value is 1.

The content of a file read through an ASCII channel should be a sequence of hexadecimal numbers separated by one or more spaces, tabs or newline characters.When reading a TTY channel, the numbers will be read from the keyboard one for each line. Data read or written are truncated if their value exceeds the length given with command.Once opened for INPUT, a channel may be read by using the # operator followed by the channel number (e.g., #1). The data read may be assigned to any variable handled by the expression evaluator.

Example

```
de -m test_auto
open 1 address input ascii 2
open 2 result output binary
for v0 1 A 1
eval v1=#1
eval #2=x:v1
endfor
close 1
close 2
endm
```

OUTPUT — Define file for data output

Syntax OUTPUT [#(file number)] [address] OFF | TERM | filename [-dec | -fra | -hex | -bin] [-Ovr] [-count]

Menu None

Description The output command is used to open a text file and use the file to save data passed from the target DSP chip. The data is passed to the file when the user's DSP program reaches a software breakpoint.

The OUTPUT command with no parameters displays all open output files.

file#	is the number of a file opened (multiple files can be opened simultaneously). The file number is optional, the range is 1 to 99. The ASCII text file lists the data sequentially.
addr	is the address of the debug instruction (always in the P: space). The address is optional if one of the direction bits in register R1 is set.
OFF	closes an opened output file. If a file is not specified, all opened files are closed.
TERM	uses the terminal/monitor (instead of a text file), and display the data.
fname	is the name of the file used for data output.
-dec	specifies decimal data representation
-fra	specifies fractional data representation
-hex	specifies hexadecimal data representation. This is the default.
-bin	specifies binary file format (LSB written first)
-Ovr	Over-write file if the file exists.
-count	Column count for ASCII representations or byte count to store to the OUTPUT file for every word read from the target processor (1..4). If the count is less than word length, high bytes will be truncated. If the count is greater than word length, extra bytes will be filled with 0s (no sign extension).

Parameters specified by the user's DSP program.

Since multiple files can be opened, your DSP program must specify the file number. This is accomplished by placing the file number in the most significant 8 bits of register X0.

The user program must also specify the number of words to transfer. This is accomplished by placing that number in the low 16 bits of register X0. 8 bits are used for the 16-bit DSPs.

The user program must also specify the address where the data will be read from. This is accomplished by placing that address in register R0.

The user program must specify the address space (P:, X: or Y:) where the data will be placed. This is accomplished by placing a value in the register R1. A 0 is used for P:, 1 for X: and 2 for Y: memory space.

The direction of the data transfer may be specified (optional) by placing a flag in register R1. If the most significant bit (8000 hex) is 1, the direction is into the DSP. If the 14th bit (4000 hex) of R1 is 1, the direction is out of the DSP. If neither bit is set, the address of the DEBUG instruction is used.

The breakpoint must reached by a user induced DEBUG instruction.

Details on the text file

The text file is always ASCII, with the data listed sequentially, in the format specified but the OUTPUT command.

Examples on entering the input command

Example 1 `OUTPUT`

Display all currently open output files.

Example 2 `OUTPUT #1 P:100 DATA.OUT -rd`

Open "DATA.OUT" file, label it file number 1, and use the file to save data when the breakpoint at address P:100 is reached. Data is saved in decimal.

Example 3 `OUTPUT P:100 DATA.OUT`

Open "DATA.OUT" file, label the file automatically, and send the data to the file when the breakpoint at address P:100 is reached. Data in "DATA.OUT" is in hexadecimal.

Example 4 `OUTPUT P:100 TERM`

When the breakpoint at P:100 is reached, read data from the DSP and send it to the terminal (screen).

4

Examples DSP code to support the OUTPUT command

Example 1

```
MOVE        $010010,X0  ; 16 WORDS to FILE NUMBER 1
MOVE        $0000,R0    ; READ DATA FROM ADDRESS 0
MOVE        $0001,R1    ; READ DATA FROM SPACE X:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 2

```
MOVE        $020012,X0  ; 18 WORDS to FILE NUMBER 2
MOVE        $0020,R0    ; READ DATA AT ADDRESS $20
MOVE        $0000,R1    ; READ DATA FROM SPACE P:
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

Example 3

```
MOVE        $030080,X0  ; 128 WORDS to FILE NUMBER 3
MOVE        $0200,R0    ; READ DATA AT ADDRESS $200
MOVE        $4002,R1    ; READ DATA FROM SPACE Y:
                        ; (OUTPUT DIRECTION)
DEBUG                   ; BREAK AND ENTER THE DEBUG MODE
```

PATH — Define, display or remove search paths

Syntax PATH [pathname[,OFF]]

Menu ---

Description The PATH command with no parameter will display a list of the defined search paths for the system. Search paths are used to locate object files, source files and alias commands. If the pathname is specified, it will be added to the list of the search paths. The OFF parameter will remove the specified pathname or clear the path list if the pathname was empty.

PAUSE — Pause for a number of seconds

Syntax PAUSE *secs*

Menu None

Description Temporization command, waits for *secs* seconds.

Example

```
Pause 10
```

PRINT — Print strings and values

Syntax

Print [-x | -d | -o] [-n] [-r] [expr] ...

Print [-x | -d | -o] [-n] [-r] -f "C-like-format" [expr] ...

Menu

None

Description

The PRINT command allows the user to print (display) strings and values. It can be used in macros to print messages and to convert the base of numbers.

-x specifies the hexadecimal output format.

-d specifies the decimal output format.

-o specifies the octal output format.

Note that expressions in the list of arguments are always interpreted according the current base, independently from options -x, -d and -o.

-n omits a newline to be printed at the end of a line.

-r displays in reverse video mode.

-f "C-like-format" allows to format output. The same format is applied to all *expr* parameters. Follows the ANSI C syntax.

Example

```
print -r "TITLE IN REVERSE"
print -f "value = %04x" 0x0A --> value = 000A
print -f "value = %04d" 0x0A --> value = 0010
print -f "value = %4x" 0x0A --> value = A
print -f "value = %4d" 0x0A --> value = 10
print "value of stack pointer register =" SP
print -d "LOOP COUNTER VALUE=" LOOPCNT
--> print value (in decimal format) at the address
defined by symbol LOOPCNT
```

QUIT — Exit application

Syntax QUIT

Menu File → Exit

Description The QUIT command will close the target connection (if opened), save current windows configuration, and close the application.

RELOAD — Reload program code (no symbols)

Syntax RELOAD [filename]

Menu ---

Description The RELOAD command will load last loaded code to the target processor. The RELOAD command leaves the symbol table intact.

RESET — Reset target processor

Syntax RESET

Menu Process → Reset

Description The RESET command will reset the target processor to debug mode. This is a hardware reset using the physical processor reset signal. In multiprocessor systems it can affect other processors. For the software reset only, the appropriate macro command should be executed.

RESTART — Reset program to its entry point

Syntax RESTART

Menu Process → Restart

Description The RESTART command resets the program to its entry point. This command assumes that program is already loaded with the LOAD command, or that symbolic information is entered with the SLOAD command.

RETURN — Exit from subroutine

Syntax RETURN

Menu Process → Return

Description The RETURN command will execute code in the current C function, and halts when execution reaches the caller.

RUN — Execute multiple instructions

Syntax RUN [count]

Menu Process → GO, F5

Description The RUN command executes count instructions using the target's "trace counter". If the count is omitted, the RUN command will behave as the GO command with no parameter.

SAVE — Save memory blocks to file

Syntax SAVE address_range [,address_range] filename

Menu None

Description The SAVE command saves the memory blocks specified by one of more definitions of the address_range to the ASCII file in the OMF format (extension .LOD). This data file can be loaded back to the processor memory with the LOAD command.

SEARCH — Search a pattern in memorys

Syntax SEARCH [space] *from to pattern*

Menu None

Description The SEARCH command searches for the value pattern in memory, between from and to addresses.space may be “x:”, “y:” or “p:”. Default value for space is “p:”, i.e., program space.

The pattern can consist of 1 to 8 words

If the pattern is found, the display is preceded by the address of memory where it has been found. The format is similar to that of DM command.

Example

```
SE x:100 200 FF00
```

SET — Set/reset options

Syntax SET [option argument]

Menu None

Description The SET command displays or changes the emulator's settings.

- ECHO ON | OFF
  When set to OFF, no message is echoed on the screen. This command must be used in a command file.
  When set to ON, all messages are echoed again on the screen.
- LOG *file* | OFF
  If *file* is specified this option makes a typescript of everything printed at the user's terminal in *file*.
  When argument is OFF the current LOG file is closed
- VERBOSE ON | OFF
  When set to ON, any command, either read from the keyboard, from a macro, or from a command file is echoed before being executed.
- TAB value
  Number of spaces every tab character is expanded to.
- TIME24 ON|Off
  Time format for DIR command
- TAB count
  Sets the TAB spacing for the source file display.
- WATCH time_msec
  Sets the delay in milliseconds for display of the "instant" watch value. Instant watch is available for any expression within the code window. If no text is highlited, evaluated will be a word under the mouse cursor.
  Default value is 300 ms
- TIMER time_msec
  Sets the application main timer. The main timer is used to perform status check of the target. Default value is 100 ms. Decreasing this value will speed up debugger response for the target events.
- TIME24 ON|OFF
  Sets the mode for the display of files directory with the DIR command.
- ERRORS ON|OFF
  Disabling this option will allow the command scripts to execute after detecting the error.

- TTYLOG ON|OFF

  Enabling this option, will display all commands and command output within the terminal window. If the TTYLOG option is off, commands will be displayed in the terminal window, only if the logging is enabled (LOG command)

- TTYWRAP ON|OFF

  Enabling TTYWRAP option, will cause TERMINAL display to insert the newline character, if the length of the displayed line exceeds the window width.

- GOFROM ON|OFF

  This option will set the mode of the interpretation the parameter of the GO command. If GOFROM is set to ON, address specified with the GO command will be written to the PC, before target processor is started. If the option is OFF, address will be interpreted as a “goto” address. The temporary breakpoint will be set at the address.

- PRINTTERM SW|HW|OFF

  Sets the mode of operation of the stdout functions within Tasking C programs.

  Enabling PRINTTERM SW, will set a software breakpoint at the “_simo” or the “_fss_break” symbol. If the breakpoint is hit, the associated text will be displayed. The code at the breakpoint is executed after any stdout (printf, puts, ...) function is called.

  Enabling PRINTTERM HW (available for the SB-56K only), will set the hardware breakpoint, and all data transfer will be performed by the emulators firmware, which make the printterm operation much faster

- SCROLL hexValue

  Sets the range of the addresses for the code and memory window scrolling. If this value is set, the thumbtrack will be located in the middle of the scroll bar, and its movement will limit the address changes to the specified value.

Example

```
set verbose on
set log trace
set
time
set echo off
go
set echo on
time
set log off
```

SLOAD — Load symbol table

Syntax SLOAD filename

Menu Load → File

Description The SLOAD command loads the symbol table from the specified object file. SLOAD clears the existing symbol table before loading the new one. If the filename has no extension, the system will add automatically the .cld extension.

SRC — Change code window display to the source mode

Syntax SRC

Menu Local code window menu

Description The SRC command will change display mode of the last selected Code window to the source mode. If the source line information is not available, this command will have no effect.

STATUS — Check target status

Syntax STATUS

Menu ---

Description The STATUS command checks the target processor status. Status is also displayed on the right side of the status bar, on the bottom of the application window.

STEP — Single step single or multiple instructions

Syntax STEP [count]

Menu Process → Step

Description The STEP command will single step the target code. If count is not specified it will step a single instruction. For the multi-instruction step, the debugger windows will be refreshed after every step. To execute multiple instructions without windows refresh, use the RUN command.

TCLOCK — Set/get JTAG clock frequency

Syntax TCLOCK [value]

Menu None

Description The TCLOCK command allows the user to set the JTAG clock frequency used to access target DSPs. Default TCK frequency is about 2 MHz, but this can be too high for processors running on slow clocks. The frequency specified does not have to be accurate. The value will be adjusted to the closest available setting for the emulator. If no value is specified, the current TCK frequency will be displayed.

Example

```
TCLOCK 32000
```

TILE — Tile windows

Syntax TILE

Menu Window → Tile

Description The TILE command will arrange all of the opened windows in the tile format.

TIME — Display time informationl

Syntax TIME [mode]

Menu None

Description The TIME command allows the user to display the current system time, or the time in milliseconds from last TIME command usage.

The mode parameter allows for different display formats:

0	Display full time and date (default)
1	Display time and delta time from last TIME (hh:mm:ss.mmm (msec))
2	Display delta time from last TIME command (milliseconds)
3	Clear millisecond delta timer (initialize)

TIMER — Set/get windows idle timer

Syntax TIMER *[value]l*

Menu None

Description The TIMER command allows the user to adjust the application idle timer. This timer is used to periodically check for the target processor status.

Value represents the idle time in milliseconds.

UNALIAS — Remove alias for given name

Syntax UNALIAS [name]

Menu ---

Description If name is present, the UNALIAS command will remove that name alias. All system aliases are removed if the parameter is empty.

UNDEFINE — Remove symbols or macros

Syntax UNDEFINE {-a | -m} symbol...
UNDEFINE {-a | -m} -all

Menu None

Description The UNDEFINE command removes the symbol of the given type from the symbol table, or clears the entire symbol table if the -all option is specified.

-a Memory address.

-m Macros.

Example

```
un -a lab
un -m LOADANGO
```

See also ARCHIVE, DEFINE, LISTSYMBOL

USE — Define, display or remove search paths

Syntax USE [pathname[,OFF]]

Menu ---

Description The USE command with no parameter will display a list of the defined search paths for the system. Search paths are used to locate object files, source files and alias commands. If the pathname is specified, it will be added to the list of the search paths. The OFF parameter will remove specified pathname or clear the entire path list if the pathname is empty.

VERSION — Display program version

Syntax VERSION

Menu ---

Description The VERSION command will display version numbers of all active components.

WAIT — Wait specified time or for debug mode

Syntax WAIT *[DEBUG | time]*

Menu None

Description This command allows the user to suspend macro command execution for the specified number of milliseconds, or wait until the target device is in debug mode (reaches one of the breakpoints)

Example

```
Load test.cld
Break main
Go
Wait debug
beep
```

WATCH — Add watch variable

Syntax WATCH address | register | expression [,label[,mode]]

Menu View → Symbol

Description The WATCH command will add or remove watch variable from the Watch window.

address should have the following format: space: offset, where space is one of the allowed memory spaces for the processor (P, X, or Y), offset absolute value of the address offset. Address can be also specified in the form of the application symbol.

register is a valid name of one of the target registers

expression can be any C-type expression. Symbols are evaluated to the corresponding values. The meory space of the symbol can be overriden with the "(space:)" cast, where space is P, X, Y or L.. The memory space cast can be only entered at the beginning of the line.

label is a string which will be displayed instead of the watch.
address mode is one of the following:

c - ASCII character
d - decimal
e - exponential floating point
o - octal
p - address
s - ASCII string
u - unsigned decimal
x - hexadecimal
r - remove watch from the watch window.

WINDOW — Open new window

Syntax WINDOW win_type

Menu View → win_type

Description Window command allows to open new window of the specified type. Window type can be specified with the partial name only.

4