WIND RIVER

Wind River® Workbench for On-Chip Debugging

COMMAND REFERENCE

2.6.1

Corporate Headquarters
Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

http://www.windriver.com

For information on how to contact Customer Support, please visit the following URL:

http://www.windriver.com/support

Wind River Workbench for On-Chip Debugging Command Reference , 2.6.1

19 Mar 07
Part #: DOC-15985-ND-00

Contents

1
Introduction

1.1 Overview

This manual provides a reference guide for all low-level Wind River ICE SX and Wind River Probe commands. The commands are listed alphabetically and include descriptions, syntax information, and examples. Note that some commands apply only to certain emulators; those that are specific to one emulator are so marked, for example **Wind River ICE SX Only**.

The Wind River ICE SX emulator provides Ethernet support and allows for remote operation on any TCP/IP network. Network operation has its own subset of commands, which are described in*5. Wind River ICE Network Operation Commands*.

Wind River ICE SX and Wind River Probe use a standard ASCII protocol with XON/XOFF flow control.There are two 512-character buffers, one for commands and the other for responses. A command is not interpreted until a termination character is received. At most, only two commands are stored in the character buffer at a given time: the command that is executing, and the one that is about to be executed.

Character processing is case insensitive, with the exception of code and data symbols. A command name is at least two characters in length. Most commands can accept optional arguments or parameters. A space character is used as the delimiting character between command names and arguments.

2
Operational Modes

2.1 The Operational Modes of Wind River Emulators

Wind River emulators can operate in any one of four main operational modes. Each mode is easily identified by the command prompt that is displayed on your host or terminal screen.

Background Mode: >BKM>

When the **>BKM>** prompt displays, Background mode is both enabled and active, meaning that no code is currently executing and the target is stopped. At the prompt, users can enter and execute any valid Wind River low-level commands. You can enter Background mode any time you connect to your target, any time you use the **IN** or **INN** command, or any time you stop your target from running by typing **Ctrl+C** or **Ctrl+X**. Your target is also placed in Background mode any time a software breakpoint is hit.

Target Running in Real-Time: >RUN>

The **>RUN>** prompt only displays after you issue a **GO** command. The target executes application code in Real-Time; Background mode is enabled, but not currently active.

In > **RUN** > mode, you can enter commands to capture a target snapshot. A snapshot is a view of the data on your target retrieved by the debugger by instantaneously forcing the target into Background mode and then returning it to Real-Time execution.

Error Mode: >ERR>

Error mode occurs when the unit fails to establish debug mode communications. There are several possible reasons for this; see the *Establishing Communications* chapter of your emulator's Hardware Reference for more information.

Profiling Mode: >PFA>

The **>PFA>** profiling prompt indicates that the emulator is profiling your code (the performance analysis features are activated with Graphics mode turned off). To return to a **>BKM>** prompt, type **Ctrl+C** or **Ctrl+X**. In Profiling mode, you can type a **DIP** command to view the performance data.

Network Mode: >NET>

(Wind River ICE SX only)

The **>NET>** prompt appears in the **Terminal** view of Wind River Workbench when you have made a serial connection to the Wind River ICE SX,or in a Telnet window when you have made a remote connection. Type **BKM** at the **>NET>** prompt to bring up a **>BKM>** prompt, and type **Ctrl + D** to return to the **>NET>** prompt.

3
Low-Level Commands

3.1 Low-Level Commands

The following commands are commands that can be used in the **OCD Command Shell** in Wind River Workbench. They are listed alphabetically, and each command includes a description, syntax, and example where relevant.

Most of these commands are valid only at a **>BKM>** prompt. However, any commands that supply version information are also valid at an **>ERR>** prompt.

3.1.1 Inline Assembler (ASM)

Use the **ASM** command to write to memory using instructions instead of opcode.

Syntax

There are three different ways to use this command. Use the first syntax to specify the start address and the instruction on the same line.

`ASM` *start_addr instruction*

start_addr — The address to write to.

instruction — The instruction used to write to that address.

Use the second syntax to specify only a start address, and then you are placed into **ASM** mode where you can enter instructions on sequential lines of memory.

`ASM` *start_addr*

start_addr — The address to begin writing at.

The third syntax requires no parameters, and merely places you into **ASM** mode, taking the address of your PC as the start address.

```
ASM
```

Entering a period (**.**) at the **>ASM>** prompt returns you to a **>BKM>** prompt.

```
ASM> .
```

Examples

The first example displays memory at 0 and then uses the first syntax for the **ASM** command. Then the code is disassembled at 0.

```
>BKM>dm 0
00000000: 0000 0000 0000 0000 0000 0000 0000 0000     ................
>BKM>asm 0 addi r2,r2,0x1
>BKM>di 0 1
$00000000 : 0x38420001 :ppc addi     R2,R2,0x1
```

The second example places the emulator into **ASM** mode, at starting address 20. Entering a period (**.**) at the **>ASM>** prompt returns you to a **>BKM>** prompt. Then the code is disassembled at 20.

```
>BKM>asm 20
$00000020 : 0x00000000 :ppc dc.l       0x0 ASM>  addi r2,r2,0x1
$00000024 : 0x00000000 :ppc dc.l       0x0 ASM>  .
>BKM>di 20 1
$00000020 : 0x38420001 :ppc addi       R2,R2,0x1
>BKM>
```

The third example places the emulator into **ASM** mode, taking the current address of the PC as the starting address. The code is then disassembled at the starting address of the PC, in this case 50.

```
>BKM>asm
$00000050 : 0x00000000 :ppc dc.l       0x0                        ASM>  addi
r2,r2,0x1
$00000054 : 0x00000000 :ppc dc.l       0x0                        ASM>  .
>BKM>di 50 1
$00000050 : 0x38420001 :ppc addi       R2,R2,0x1
>BKM>
```

See also the **DM** (Display Memory) and **DI** (Disassemble) commands.

3.1.2 Boot Line Parameters (BL)

The Boot Line (**BL**) commands are used to set Linux structures and registers. There are several **BL** commands with various functions, as described below.

NOTE: To use the **BL** commands you must have both the **CF BL** and **CF MMU** options set to **ENABLE**.

BL ADD REGISTER

This command will enter and present a menu prompting for additional entries for items to be added into the REGISTER table held in NVRAM. Sequentially, each entry field will be prompted with the current contents. A new value may be entered. An **RTN/ENTER** will keep the existing value and advance to the next field. To exit the menu, type a period**.**

Information can be added to the **BL ADD REGISTER** command to add a new item at the end of the REGISTER table without going through the menu:

Syntax:

```
BL ADD REGISTER description type value
```

description is an ASCII string representing the name of the entry.

type should be left blank. REGISTER table entries have UINT32 type by default.

value is an ASCII string containing a Hex value or a string that is resolved and loaded dynamically.

Example:

```
>BKM>BL ADD REGISTER R6 E(R3) + S(0:19) + S(1)
```

R6 is the CPU Register selected; E(R3) stands for the value of the entry called R3 in the REGISTER table; S(0:19) is the sum of the sizes of entries 0 through 19 in the STRUCTURE table; and S(1) is the size of entry 1 in the STRUCTURE table.

All these arguments must be valid. That is, if there is no entry called R3 in the REGISTER table, the **BL ADD REGISTER** command will abort and return a syntax error.

BL ADD STRUCTURE

This command will enter and present a menu prompting for additional entries for items to be added into the STRUCTURE table held in NVRAM. Sequentially, each entry field will be prompted with the current contents. A new value may be entered. An **RTN/ENTER** will keep the existing value and advance to the next field. To exit the menu, type a period**.**

Information can be added to the **BL ADD STRUCTURE** command to add a new item at the end of the STRUCTURE table without going through the menu:

Syntax:

`BL ADD STRUCTURE` *description type value*

description is an ASCII string representing the name of the entry.

type can be U32 for UINT32, U16 for UINT16, U8 for UINT8, and char for CHAR.

value--the entered value must comply with the specified type.

Example:

```
>BKM>BL ADD STRUCTURE MemSize U32 0x04000000
```

BL DELETE

Use this command to delete the REGISTER and STRUCTURE tables held in NVRAM.

Syntax:

`BL DELETE [REGISTER,STRUCTURE]`

Entering **BL DELETE** without specifying a table will delete both tables.

BL DISPLAY

Use this command to display the REGISTER and STRUCTURE tables held in NVRAM.

Syntax:

`BL DISPLAY [REGISTER,STRUCTURE]`

Entering **BL DISPLAY** without specifying a table will display both tables.

BL INIT

This command will load the **BL** parameters into target memory and registers without performing an **IN** command. This is useful in cases where the bootROM is run to initialize the target board and the emulator is used to establish the boot line parameters.

BL MODIFY

This command will display and prompt to modify the field contained in the specified entry item. Menus are similar to the **ADD** command menus. Press **ENTER** to go on to the next menu item without modifying the current value. Type a period (**.**) to exit.

Syntax:

`BL MODIFY` [**`REGISTER`**, **`STRUCTURE`**] *entryIndex*

BL UPLOAD

Use this command to upload and display the STRUCTURE and REGISTER tables held in NVRAM in playback-command format. This enables the user to cut and paste into a host-resident file for later playback as an emulator command-script file. This same file can be concatenated with the typical target initialization register file normally used to set up the primitives of the target.

Syntax:

`BL UPLOAD` [**`REGISTER`**, **`STRUCTURE`**]

Entering **BL UPLOAD** without specifying a table will upload and display both tables.

3.1.3 Breakpoint Disable (BD)

This command disables all or one of the currently enabled software breakpoints. If a breakpoint is disabled, it is not installed in target memory prior to execution. Using the **BD** command allows the emulator to remember the breakpoint for future use, unlike the **RB** (remove breakpoint) command, which deletes the breakpoint entirely.

Syntax

BD *address*

address — The address where the breakpoint is installed. Use the **DB** command to display the address. Previously defined symbol names are allowed for address. If address is not specified, the **BD** command will disable ALL breakpoints.

Example

The following example disables a breakpoint at address EA0000. First, display all breakpoints, including the current status. Then disable the breakpoint. Finally, verify the status of the breakpoint by issuing a **DB** command. Notice the status change of the **cmp** flags.

```
>BKM>DB
1. 0Ea0000 data mask = 000ff cmp_flags = enabled Word_cmp_BEQ
>BKM>BD EA0000
>BKM>DB
1. 0EA0000 data mask = 000ff cmp_flags = disabled Word_cmp_BEQ
>BKM>
```

Refer to **SB** (set breakpoint) command and **DB** (display breakpoints) command for information on those commands.

3.1.4 Breakpoint Enable (BE)

This command enables all or one of the currently disabled software breakpoints.

Syntax

BE *address*

address — This is the address where the breakpoint is installed (use the DB command to display the address). Previously defined symbol names are allowed for address. If address is not specified, the **BE** command will enable all breakpoints.

Example

For this example, enable the breakpoint at address EA0000, which was previously disabled. First, display all breakpoints and their status. Then enable the breakpoint. Finally verify the status of the breakpoint by issuing a **DB** command. Notice the status change of the **cmp_flags**.

```
>BKM>DB
1. 0EA0000 data mask = 000ff cmp_flags = disabled Word_cmp_BEQ
>BKM>BE EA0000
>BKM>DB
```

```
1. 0EA0000 data mask = 000ff cmp_flags = enabled Word_cmp_BEQ
BKM>
```

See the **SB** (set breakpoint) command and the **DB** (display breakpoints) command for more information.

3.1.5 Block Fill (BF)

This command fills a block of units from *start_addr* to *end_addr* with data.

Syntax

BF*unit start_addr end_addr data*

unit — This can be either **B** (byte), **W** (word), or **L** (long). If no unit is specified, the default is **W** (word).

start_addr — This is the address at which to begin filling.

end_addr — This is the end of the address range to fill.

data — This is a data pattern to be placed into the block range specified.

start_addr and *end_addr* can be previously defined symbols.

Example 1

In this example, fill a block of memory, 32 bytes (20 Hex) in length with a word of *data* = 4121. First, display memory contents. Then fill the block and verify the command.

```
BKM>DM 1000 20
001000 0000 0000 0000 0000 0000 0000 0000 0000 ....................
001010 0000 0000 0000 0000 0000 0000 0000 0000 ....................
>BKM>BF 1000 1020 4121
>BKM>DM 1000 20
001000 4121 4121 4121 4121 4121 4121 4121 4121 ....................
001010 4121 4121 4121 4121 4121 4121 4121 4121 ....................
>BKM>
```

Example 2

In this example, change the command unit size from the last example of a word to a size **L** (long):

```
>BKM>DM 1000 20
001000 0000 0000 0000 0000 0000 0000 0000 0000 ....................
001010 0000 0000 0000 0000 0000 0000 0000 0000 ....................
>BKM>BFL 1000 1020 4121
>BKM>DM 1000 20
```

```
001000 0000 4121 0000 4121 0000 4121 0000 4121 ..A!..A!..A!..A!..A!
001010 0000 4121 0000 4121 0000 4121 0000 4121 ..A!..A!..A!..A!..A!
BKM>
```

See also the information provided on the **DM** (display memory) command.

3.1.6 Block Move (BM)

The **BM** command copies blocks of data from one address to another.

Syntax

BM*unit source_start_addr source_end_addr dest_start_addr*

unit — This can be either **B** (byte), **W** (word), or **L** (long.) By default it is set to **W** (word).

source_start_addr — Start address of the memory to move.

source_end_addr — End address of memory to move.

dest_start_addr — Start address of the location where the memory is moving to.

Previously defined symbols can be used for any of the above addresses.

Example

In this example, move a 32 byte (20 Hex) block from one area of memory to another. Memory is displayed before and after the move.

```
>BKM>DM 1000 20

001000 5448 4953 2049 5320 4120 5445 5354 2121 THIS IS A TEST!!
001010 0000 0000 0000 0000 0000 0000 0000 0000 ................
BKM>BM 1000 1010 1010
BKM>DM 1000 20

001000 5448 4953 2049 5320 4120 5445 5354 2121 THIS IS A TEST!!
001010 5448 4953 2049 5320 4120 5445 5354 2121 THIS IS A TEST!!
BKM>
```

See also **DM** (display memory) command.

3.1.7 Cache Access Command (CA)

The Cache Access (**CA**) commands are used to view and alter the state of the Instruction and Data caches of the processors. Note that certain **CA** commands are processor specific, as indicated in the command descriptions. There are several **CA** commands with various functions, as described in the sections that follow.

Syntax

CA

Issuing the **CA** command without any arguments lists all of the Cache Access commands and their syntax.

Example

```
>BKM>ca
CA dis                       : Disassemble all VALID I cache data.
CA dump <sw>                 : Display ALL cache sets for I, D or L caches.
CA dump <addr> <sw>          : Display the appropriate cache SET for all
                               WAYs.
CA dump <addr> -il           : Display true target i-cache tags and data
                               directly.
CA dump <addr> -dl           : Display true target d-cache tags and data
                               directly.
CA dump <addr> -ll           : Display true target L2-cache tags directly.
CA read <sw>                 : Display VALID cache sets for I, D or L caches.
CA search <addr..addr> <sw> : Searches for <addr> in the I, D or L caches.
CA enable <sw>               : Enables both or just I or D caches.
CA disable <sw>              : Disables both or just I or D caches.
CA inv <sw>                  : Invalidate both or just I or D caches.
CA -s                        : Display the state for both caches.
CA diff <sw>                 : Display differences between Cache and memory.

SW -d data, -i instruction cache only, -l L2 cache, -l3 L3 cache>BKM>
```

The **CA** commands that appear above are described in the following sections.

CA dis

This command displays all valid lines in the instruction cache in a disassembly format. The example displays all the cache information related to where this data was obtained from in the cache, and what the instructions are for each line.

Example:

```
>BKM>ca dis
Way0 Set 000 $00005000 : 0x48000005 :ppc bl         0x5004
[V,LRU 4]   $00005004 : 0x7C6802A6 :ppc mflr       R3
             $00005008 : 0x3C801122 :ppc lis        R4,0x1122
             $0000500c : 0x60843344 :ppc ori        R4,R4,0x3344
             $00005010 : 0x90830040 :ppc stw        R4,0x40(R3)
             $00005014 : 0xB0830050 :ppc sth        R4,0x50(R3)
             $00005018 : 0x98830060 :ppc stb        R4,0x60(R3)
             $0000501c : 0x80A30040 :ppc lwz       R5,0x40(R3)
Way0 Set 001 $00005020 : 0xA0A30050 :ppc lhz       R5,0x50(R3)
[V,LRU 4]   $00005024 : 0x88A30060 :ppc lbz       R5,0x60(R3)
             $00005028 : 0x7C0004AC :ppc sync
             $0000502c : 0x4BFFFFD4 :ppc b         0x5000
             $00005030 : 0x80020010 :ppc lwz       R0,0x10(R2)
             $00005034 : 0x80000000 :ppc lwz       R0,0x0(R0)
             $00005038 : 0x80010000 :ppc lwz       R0,0x0(R1)
             $0000503c : 0x80081082 :ppc lwz       R0,0x1082(R8)
```

CA dump -i

This command displays all of the processor instruction cache lines regardless of the state of each line. This example shows a processor with an instruction cache structure that has 5 WAYS and 128 sets per WAY. The entry at way0/set0 is marked as valid with the /v flag, and all other entries are marked as invalid /i.

Example:

```
>BKM>CA dump -i
Set    Way0          Way1          Way2          Way3          Way4
000 0x00005000/v 0x3C332000/i 0x57FFB000/i 0xB24FC000/i 0xDEE20000/i
001 0xFFFFF020/i 0xFFFFF020/i 0xFFFFF020/i 0x1DAF8020/i 0xE7B28020/i
002 0xAAFE9040/i 0x5BC1D040/i 0x5213E040/i 0x53788040/i 0xC875D040/i
003 0xCC7EC060/i 0x2D662060/i 0x952F6060/i 0x1BB7D060/i 0x18093060/i
004 0xD91BB080/i 0x5A2AB080/i 0x6BDE3080/i 0xED2A2080/i 0x51E43080/i
005 0xD27950A0/i 0x98C0E0A0/i 0x4E5E00A0/i 0x094B40A0/i 0x3A5290A0/i
006 0x3BB400C0/i 0x347A20C0/i 0xA36180C0/i 0xA23300C0/i 0x5099A0C0/i
etc ............
126 0xD27950A0/i 0x98C0E0A0/i 0x4E5E00A0/i 0x094B40A0/i 0x3A5290A0/i
127 0x3BB400C0/i 0x347A20C0/i 0xA36180C0/i 0xA23300C0/i 0x5099A0C0/i
>BKM>
```

CA dump -d

This command displays all of the processor data cache lines regardless of the state of each line. This example shows a processor with a data cache structure that has 4 WAYS and 128 sets per WAY. The entry at way0/set2 and set3 are marked as valid with the /v flag, and all other entries are marked as invalid /i. There is a second flag associated with the data cache. This flag indicates if the data is dirty (most recent) or clean (same as memory). If the entry is marked as clean with the /c flag, then contents in the cache and the contents in memory are the same. If the entry is valid and it is marked as dirty with a /d flag, then the data in the cache is the most recently used data and the value in memory is no longer valid.

Example:

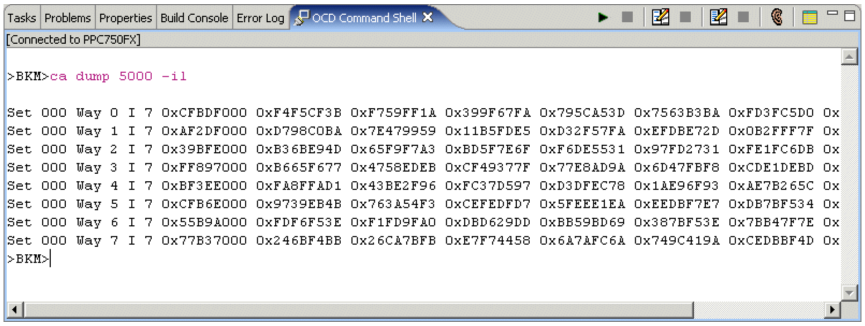
```
>BKM>CA dump -d
Set    Way0            Way1            Way2            Way3
000 0xE17AF000/i/c 0x80517000/i/d 0xCEF44000/i/c 0x37ECB000/i/d
001 0xBAF43020/i/c 0x56560020/i/d 0x384A2020/i/c 0x0A27A020/i/d
002 0x00005040/v/d 0x79F9B040/i/c 0xCB2BC040/i/d 0x80A73040/i/c
003 0x00005060/v/d 0x2F6B2060/i/c 0xED936060/i/d 0x58252060/i/c
004 0xCA098080/i/d 0x7CD4C080/i/c 0xC3BAD080/i/d 0x63B2C080/i/c
005 0xEF70F0A0/i/c 0x1AF4B0A0/i/c 0x1016A0A0/i/d 0x525E00A0/i/c
006 0xF9D5D0C0/i/d 0x458F20C0/i/d 0xAF17F0C0/i/c 0xAE8B70C0/i/c
etc ..............
125 0xD27DFFA0/i/d 0xD8C3FFA0/i/d 0x1D8B9FA0/i/c 0x7B3D3FA0/i/d
126 0x58F00FC0/i/c 0xC5AE3FC0/i/d 0x4512CFC0/i/c 0x9B7ACFC0/i/c
127 0xE94B6FE0/i/d 0x3CECFFE0/i/d 0x6EE65FE0/i/c 0xC14DDFE0/i/d
```

CA dump *addr* -il

This command displays all of the possible ways that a specified address could be found in the instruction cache. The **-il** switch is used to indicate a long format for the instruction cache. This example shows all WAYS for set0. The address TAG of 0x5000 is marked as valid, and all other ways are marked as invalid.

Example:

Figure 3-1 **CA dump 5000 -il**



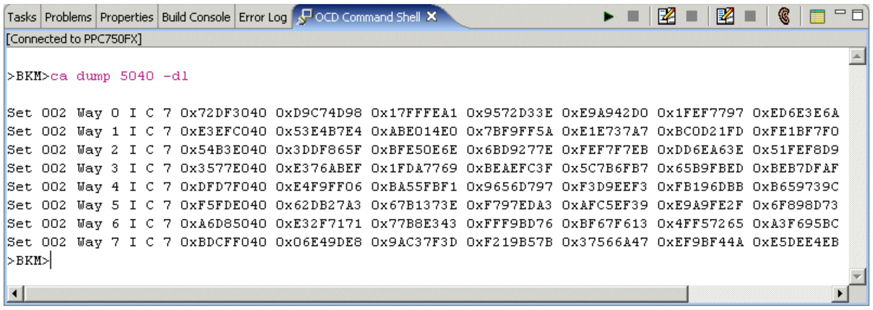
```
[Connected to PPC750FX]

>BKM>ca dump 5000 -il

Set 000 Way 0 I 7 0xCFBDF000 0xF4F5CF3B 0xF759FF1A 0x399F67FA 0x795CA53D 0x7563B3BA 0xFD3FC5D0 0x
Set 000 Way 1 I 7 0xAF2DF000 0xD798C0BA 0x7E479959 0x11B5FDE5 0xD32F57FA 0xEFDBE72D 0x0B2FFF7F 0x
Set 000 Way 2 I 7 0x39BFE000 0xB36BE94D 0x65F9F7A3 0xBD5F7E6F 0xF6DE5531 0x97FD2731 0xFE1FC6DB 0x
Set 000 Way 3 I 7 0xFF897000 0xB665F677 0x4758EDEB 0xCF49377F 0x77E8AD9A 0x6D47FBF8 0xCDE1DEBD 0x
Set 000 Way 4 I 7 0xBF3EE000 0xFA8FFAD1 0x43BE2F96 0xFC37D597 0xD3DFEC78 0x1AE96F93 0xAE7B265C 0x
Set 000 Way 5 I 7 0xCFB6E000 0x9739EB4B 0x763A54F3 0xCEFEDFD7 0x5FEEE1EA 0xEEDBF7E7 0xDB7BF534 0x
Set 000 Way 6 I 7 0x55B9A000 0xFDF6F53E 0xF1FD9FA0 0xDBD629DD 0xBB59BD69 0x387BF53E 0x7BB47F7E 0x
Set 000 Way 7 I 7 0x77B37000 0x246BF4BB 0x26CA7BFB 0xE7F74458 0x6A7AFC6A 0x749C419A 0xCEDBBF4D 0x
>BKM>
```

CA dump *addr* -dl

This command displays all of the possible WAYS where a specified address could be found in the data cache. The **-dl** switch is used to indicate a long format for the data cache. This example shows all 8 WAYS for set2. The address TAG of 0x5040 is marked as valid, and all other WAYS are marked as invalid.

Example:

Figure 3-2 **CA dump 5040 -dl**



```
[Connected to PPC750FX]

>BKM>ca dump 5040 -dl

Set 002 Way 0 I C 7 0x72DF3040 0xD9C74D98 0x17FFFEA1 0x9572D33E 0xE9A942D0 0x1FEF7797 0xED6E3E6A
Set 002 Way 1 I C 7 0xE3EFC040 0x53E4B7E4 0xABE014E0 0x7BF9FF5A 0xE1E737A7 0xBC0D21FD 0xFE1BF7F0
Set 002 Way 2 I C 7 0x54B3E040 0x3DDF865F 0xBFE50E6E 0x6BD9277E 0xFEF7F7EB 0xDD6EA63E 0x51FEF8D9
Set 002 Way 3 I C 7 0x3577E040 0xE376ABEF 0x1FDA7769 0xBEAEFC3F 0x5C7B6FB7 0x65B9FBED 0xBEB7DFAF
Set 002 Way 4 I C 7 0xDFD7F040 0xE4F9FF06 0xBA55FBF1 0x9656D797 0xF3D9EEF3 0xFB196DBB 0xB659739C
Set 002 Way 5 I C 7 0xF5FDE040 0x62DB27A3 0x67B1373E 0xF797EDA3 0xAFC5EF39 0xE9A9FE2F 0x6F898D73
Set 002 Way 6 I C 7 0xA6D85040 0xE32F7171 0x77B8E343 0xFFF9BD76 0xBF67F613 0x4FF57265 0xA3F695BC
Set 002 Way 7 I C 7 0xBDCFF040 0x06E49DE8 0x9AC37F3D 0xF219B57B 0x37566A47 0xEF9BF44A 0xE5DEE4EB
>BKM>
```

CA read -i

This command displays only the processor instruction cache lines that are valid. This example shows that TAG address 0x5000 from way0/set0 is the only valid entry found. The LRU flag indicates the Least Recently Used WAY is way0.

Example:

```
>BKM>ca read -i
Set/Way     Tag
000/000   0x00005000/lru 0
CA read -d
```

This command displays only the processor data cache lines that are valid. This example shows that TAG address 0x5040 and 0x5060 are the only valid entries found.

```
>BKM>CA read -d
Set/Way    Tag[status]:  Data+0     Data+4     Data+8     Data+C     Data+10
Data+14    Data+18    Data+1C
002/000   0x00005040[d]:0x80001000 0x11223344 0x00000400 0x00000402 0x08004000
0x33440000 0x00080080 0x00000000
003/000   0x00005060[d]:0x00001000 0x44000C02 0x00100400 0x00000000 0x08000000
0x00000000 0x00400000 0x00000010
>BKM>
```

CA search *start_addr*..*end_addr* **-i**

This command searches the instruction cache for valid cache lines. The command expects a starting and ending range. The example shows a search of the instruction cache starting from address 0x5000 to 0x6000, and has found 2 valid cache lines within this address range.

```
>BKM>CA search 5000..6000 -i
Set/Way     Tag
008/000  0x00005100
009/000  0x00005120
>BKM>
```

CA search *start_addr*..*end_addr* **-d**

This command searches the data cache for valid cache lines. The command expects a starting and ending range. This example shows a search of the data cache starting from address 0x5000 to 0x6000, and has found 2 valid cache lines within this address range.

```
>BKM>ca search 5000..6000 -d
Set/Way     Tag
010/000  0x00005140
011/000  0x00005160
>BKM>
```

CA -s

This command displays the state of each cache. L2 and L3 caches are level 2 and 3 caches found in the high end PowerPC processors.

```
>BKM>ca -s
Instruction Cache: Enabled
Data Cache       : Enabled
L2 Cache         : Disabled
L3 Cache         : Disabled
>BKM>
```

CA diff -i

This command displays all differences between the instruction cache and main memory for all valid cache lines. This example displays one cache line where the top values are the cache values, and the bottom values are main memory content. The dash lines indicate that there were no differences found.

```
>BKM>CA diff -i
Set/Way     Tag         Tag+00      Tag+04      Tag+08     Tag+0C
Tag+10      Tag+14      Tag+18      Tag+1C
001/000  0x00005020  0xA0A30050  0x88A30060  0x7C0004AC  0x4BFFFFD4
0x80020010  0x80000000  0x80010000  0x80081082
----------  ----------  ----------  ----------  0x00020080  0x00000400
0x00010000  0x00081100
>BKM>
```

CA diff -d

This command displays all differences between the data cache and main memory for all valid cache lines. This example displays one cache line where the top values are the cache values, and the bottom values are main memory content. The dash lines indicate that there were no differences found.

```
>BKM>CA diff -d
Set/Way     Tag         Tag+00     Tag+04     Tag+08     Tag+0C     Tag+10
Tag+14      Tag+18     Tag+1C
002/000  0x00005040  0x80001000  0x11223344  0x00000400  0x00000402
0x08004000  0x33440000  0x00080080  0x00000000
----------  0xAABBCCDD  0x00000000  0x00000000  0xFFFFFFFE  0xFFFFFFFE
0x00100100  ----------
>BKM
```

3.1.8 Configure Parameters (CF)

The **CF** command is a low-level ASCII command that configures important emulator and system-level parameters. The **CF** parameter table remains set until modified with subsequent **CF** commands.

For any given processor, entering a **CF** command without parameters at any **>BKM>** prompt will list all of the executable **CF** options for the processor type you are using. The following example shows available **CF** command options and parameters for an MPC 8260 board.

To make a change to any of these values, type **CF** and the **CF** command option you wish to modify, followed by the specific parameter you wish to configure, all on the same command line.

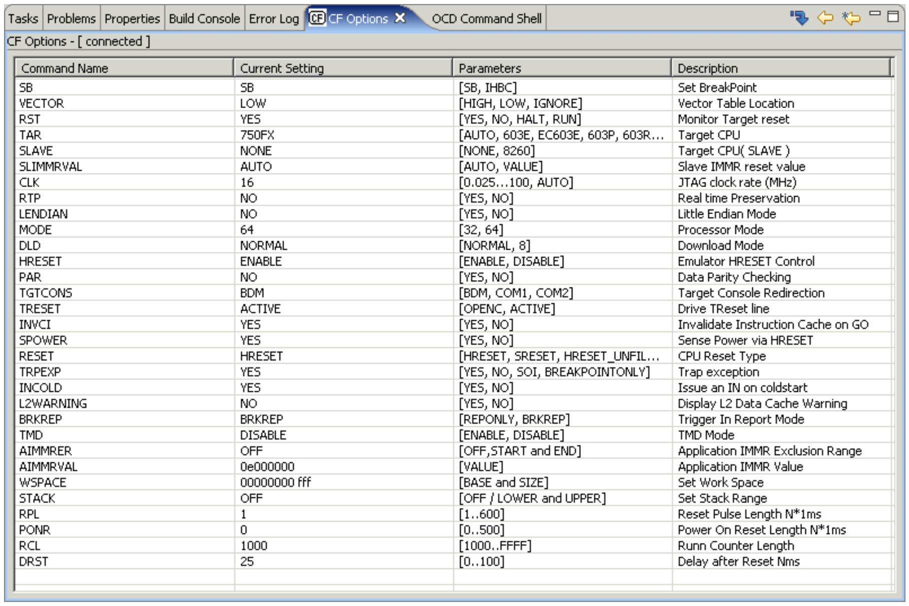
For example, to configure your system for a MPC8245 target, you would type

```
BKM>CF TAR 8245
```

You can use the **CF** command from the **> BKM >** or **>ERR>** prompt, but not from either a **> RUN >** or a **>TRC>** prompt.

Although the options vary depending on your target architecture, typing **CF** at the **>BKM>** prompt will provide a list of the options that are available for your target.

Figure 3-3 **CF Options View**



Command Name	Current Setting	Parameters	Description
SB	SB	[SB, IHBC]	Set BreakPoint
VECTOR	LOW	[HIGH, LOW, IGNORE]	Vector Table Location
RST	YES	[YES, NO, HALT, RUN]	Monitor Target reset
TAR	750FX	[AUTO, 603E, EC603E, 603P, 603R...	Target CPU
SLAVE	NONE	[NONE, 8260]	Target CPU(SLAVE)
SLIMMRVAL	AUTO	[AUTO, VALUE]	Slave IMMR reset value
CLK	16	[0.025...100, AUTO]	JTAG clock rate (MHz)
RTP	NO	[YES, NO]	Real time Preservation
LENDIAN	NO	[YES, NO]	Little Endian Mode
MODE	64	[32, 64]	Processor Mode
DLD	NORMAL	[NORMAL, 8]	Download Mode
HRESET	ENABLE	[ENABLE, DISABLE]	Emulator HRESET Control
PAR	NO	[YES, NO]	Data Parity Checking
TGTCONS	BDM	[BDM, COM1, COM2]	Target Console Redirection
TRESET	ACTIVE	[OPENC, ACTIVE]	Drive TReset line
INVCI	YES	[YES, NO]	Invalidate Instruction Cache on GO
SPOWER	YES	[YES, NO]	Sense Power via HRESET
RESET	HRESET	[HRESET, SRESET, HRESET_UNFIL...	CPU Reset Type
TRPEXP	YES	[YES, NO, SOI, BREAKPOINTONLY]	Trap exception
INCOLD	YES	[YES, NO]	Issue an IN on coldstart
L2WARNING	NO	[YES, NO]	Display L2 Data Cache Warning
BRKREP	BRKREP	[REPONLY, BRKREP]	Trigger In Report Mode
TMD	DISABLE	[ENABLE, DISABLE]	TMD Mode
AIMMRER	OFF	[OFF,START and END]	Application IMMR Exclusion Range
AIMMRVAL	0e000000	[VALUE]	Application IMMR Value
WSPACE	00000000 fff	[BASE and SIZE]	Set Work Space
STACK	OFF	[OFF / LOWER and UPPER]	Set Stack Range
RPL	1	[1..600]	Reset Pulse Length N*1ms
PONR	0	[0..500]	Power On Reset Length N*1ms
RCL	1000	[1000..FFFF]	Runn Counter Length
DRST	25	[0..100]	Delay after Reset Nms

The **CF** options that are available vary widely from target to target. For descriptions of the **CF** options for the target architectures that are currently supported by Wind River, see the *Wind River Workbench On-Chip Debugging Configuration Options Reference*.

NOTE: Some of the **CF** options may be the same from target to target. Be aware, however, that there are options that are unique to specific processor families. Be sure to refer to the section that is appropriate for the target that you are using.

3.1.9 Configure Register Groups (CF GRP)

Use this command to enable or disable register groups.

Syntax

```
cf grp
```

Example

```
>BKM>cf grp
Group          (CF GRP (M/S)    Name = ENABLED/DISABLED
SYS-CFG                               (0=Disable  1=Enable)     Enabled  >
```

The name of the first register group is displayed, along with its current status (either **ENABLED** or **DISABLED**).

Type **0** to disable the group or **1** to enable it.

To leave the setting as it is and advance to the next register group, press the **ENTER** key without typing **0** or **1**. Continue through the list of register groups enabling and disabling them as required.

When all register groups are enabled or disabled, type **CF UPLOAD GROUP** at the **>BKM>** prompt. This displays a list of all of the register groups on your target with their current settings as shown below:

```
>BKM>cf upload group
CF GRP                  SYS-CFG ENABLED            ;  GROUP
CF GRP                  INT0 ENABLED               ;  GROUP
CF GRP                  INT1 ENABLED               ;  GROUP
CF GRP                  GLOBAL_ACK ENABLED         ;  GROUP
CF GRP                  CS0-5 ENABLED              ;  GROUP
CF GRP                  SDRAM ENABLED              ;  GROUP
CF GRP                  REAL_TIME_CLOCK DISABLED   ;  GROUP
CF GRP                  PINT_TIMERS DISABLED       ;  GROUP
CF GRP                  DMA_TIMERS DISABLED        ;  GROUP
CF GRP                  PWM DISABLED               ;  GROUP
CF GRP                  EDMA_CTRL DISABLED         ;  GROUP
CF GRP                  EDMA_CHAN0-3 DISABLED      ;  GROUP
CF GRP                  EDMA_CHAN4-7 DISABLED      ;  GROUP
CF GRP                  EDMA_CHAN8-11 DISABLED     ;  GROUP
CF GRP                  EDMA_CHAN12-15 DISABLED    ;  GROUP

>BKM>
```

3.1.10 Chip Selects (CS)

NOTE: This command is processor dependent and not available across all families.

This command gives you a table-driven method for setting up the chip-selects for your processor. This table is stored in non-volatile memory and is automatically downloaded to the proper target location (via Background Mode memory sets) after every initialization sequence (**IN** command). To initialize Background Mode communications without writing the chip-select table and SIM registers (**SC** commands), use the **INN** command.

Syntax

```
CS arg
```

arg — This is the specific chip-select to modify.

If no *arg* is specified, the emulator will present a table of all of the chip selects.

Example

When the emulator is configured for an MPC 8260 target, a **CS** command with no arguments will display the following:

```
>BKM>cf tar 8260
>BKM>cs
Name BA        AM        AT ATM PS PARE WP MS V CSNT/SAM ACS  BI SCY SETA TRLX
CS0  FFC00000  FFC00000  0  0   16  No  RW GP Y Normal   1/2  Y  6ws Int. Norm
CS1  00000000  00000000  0  0   32  No  RW GP Y Normal   0    N  0ws Int. Norm
CS2  00000000  FFC00000  0  0   32  No  RW UB Y I.M.AMB  0    N  0ws Int. Norm
CS3  00000000  00000000  0  0   32  No  RW GP N Normal   0    N  0ws Int. Norm
CS4  00000000  00000000  0  0   32  No  RW GP N Normal   0    N  0ws Int. Norm
CS5  00000000  00000000  0  0   32  No  RW GP N Normal   0    N  0ws Int. Norm
CS6  00000000  00000000  0  0   32  No  RW GP N Normal   0    N  0ws Int. Norm
CS7  00000000  00000000  0  0   32  No  RW GP N Normal   0    N  0ws Int. Norm
```

The above chip-select table shows the factory default settings. To modify any of these chip-selects, type the **CS** command followed by the name of the specific chip-select to modify. The emulator displays the following for the **CS0** example, one line at a time, which shows you the options and current settings, and then prompts you to make individual changes.

```
>BKM>cs cs0
00000000 -> FFFF8000                |      Base Register = FFC00000     >
00000000 -> FFFF8000                |      Address Mask = FFC00000      >
0 -> 7                              |      Address Type = 0             >
0 -> 7                              |      Address Type Mask = 0        >
(0-2)=32, 8, 16 bits / 3=Rsvd       |      Port Size = 16 Bits          >
0 = Disabled, 1 = Enabled           |      Parity Enable = Disabled     >
0 = Read/Write, 1 = Read Only       |      Write Protect = Read/Write   >
(0-3) = GPCM, Rsvd, UPMA, UPMB      |      Machine Select = GPCM        >
0 = Not Valid, 1 = Valid            |      Valid state = Valid          >
(0-3)=Norm, Rsvd, 1/4 clk, 1/2 clk  |      Address/CS Setup = 1/2 clk   >
```

After you use the **CS** command to make a change in the emulator's chip select table, issue an **IN** command in order to download the table, and thus the changes to the target.

3.1.11 Chip Selects —Target (CST)

The **CST** command is similar to the **CS** command in that it displays a table of chip-selects. The difference is that the **CST** command provides a snapshot view of the chip-selects that are currently set on your target (rather than in the emulator). This is a useful command because it allows you to see whether or not your target is still synchronized with the chip-select table that you have programmed into your emulator.

If your target is no longer synchronized with your emulator chip-select table, an **IN** command will cause the target chip-select values to be overwritten by the ones stored in the emulator.

Syntax

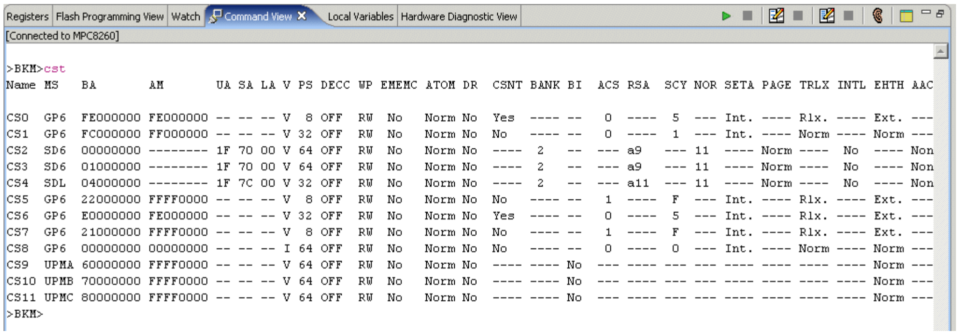
```
CST
```

There are no parameters associated with this command.

Example

The following example displays a target chip-select table for a Wind River 8260 reference design.

Figure 3-4 **Chip Select Table**



```
[Connected to MPC8260]

>BKM>cst
Name MS   BA       AM       UA SA LA V PS DECC WP EMEMC ATOM DR  CSNT BANK BI  ACS RSA  SCY NOR SETA PAGE TRLX INTL EHTH AAC

CS0  GP6  FE000000 FE000000 -- -- -- V  8 OFF  RW  No   Norm No  Yes  ---- --   0  ----  5  --- Int. ---- Rlx. ---- Ext. ---
CS1  GP6  FC000000 FF000000 -- -- -- V 32 OFF  RW  No   Norm No  No   ---- --   0  ----  1  --- Int. ---- Norm ---- Norm ---
CS2  SD6  00000000 -------- 1F 70 00 V 64 OFF  RW  No   Norm No  ----  2   --  --- a9   --- 11  ---- Norm ----  No  ---- Non
CS3  SD6  01000000 -------- 1F 70 00 V 64 OFF  RW  No   Norm No  ----  2   --  --- a9   --- 11  ---- Norm ----  No  ---- Non
CS4  SDL  04000000 -------- 1F 7C 00 V 32 OFF  RW  No   Norm No  ----  2   --  --- a11  --- 11  ---- Norm ----  No  ---- Non
CS5  GP6  22000000 FFFF0000 -- -- -- V  8 OFF  RW  No   Norm No  No   ---- --   1  ----  F  --- Int. ---- Rlx. ---- Ext. ---
CS6  GP6  E0000000 FE000000 -- -- -- V 32 OFF  RW  No   Norm No  Yes  ---- --   0  ----  5  --- Int. ---- Rlx. ---- Ext. ---
CS7  GP6  21000000 FFFF0000 -- -- -- V  8 OFF  RW  No   Norm No  No   ---- --   1  ----  F  --- Int. ---- Rlx. ---- Ext. ---
CS8  GP6  00000000 00000000 -- -- -- I 64 OFF  RW  No   Norm No  No   ---- --   0  ----  0  --- Int. ---- Norm ---- Norm ---
CS9  UPMA 60000000 FFFF0000 -- -- -- V 64 OFF  RW  No   Norm No  ---- ---- No  --- ---- --- --- ---- ---- ---- ---- Norm ---
CS10 UPMB 70000000 FFFF0000 -- -- -- V 64 OFF  RW  No   Norm No  ---- ---- No  --- ---- --- --- ---- ---- ---- ---- Norm ---
CS11 UPMC 80000000 FFFF0000 -- -- -- V 64 OFF  RW  No   Norm No  ---- ---- No  --- ---- --- --- ---- ---- ---- ---- Norm ---
>BKM>
```

See also the information provided on the **CS** command.

3.1.12 Display Breakpoint (DB)

This command displays all software breakpoints and their status. The status includes whether the break is enabled or disabled, and the conditions on which to break. For viewing convenience, each breakpoint is assigned a number. In addition, after a breakpoint has occurred, a flag indicates which breakpoint has interrupted code execution.

Syntax

DB

Response: *addr break_conditions status*

addr — the address at which the breakpoint is set

break_conditions — conditional information that can be set using the **SB** command (such as running through code count number of times before hitting the breakpoint)

status — states whether the breakpoint is enabled or disabled

Example

```
>BKM>db
    Software Code Breakpoints
1.  00040418  count = 0001 actual = 0000 enabled
2.  0004042C  count = 0001 actual = 0000 enabled
3.  0004044C  count = 0001 actual = 0000 enabled
4.  00040494  count = 0001 actual = 0000 enabled
!INFO! - [msg82001] No internal hardware breakpoints installed
>BKM>
```

See also the **SB** (Set Breakpoint) command for more information.

3.1.13 Display Configuration (DC)

This command displays the current hardware and firmware levels that your emulator is using.

Syntax

DC

Example

```
>BKM>dc
Firm Rev = vn1.0m
UJD Rev = 1.0h
PDI Version 1.1 WIND POWER ICE PPC82XX (type = 0x30)
>BKM>
```

3.1.14 Diagnostic Function (DF)

This command runs a pre-written emulator diagnostic function. This section describes the build-in diagnostic functions supported by the emulator which are run from a **>BKM>** prompt, using the low-level command language built in to the emulator, rather than using a graphical user interface. Their primary function is to expose memory problems, both in RAM and ROM, as well as any download problems.

The following pages serve as a reference for the available diagnostics, all of which are accessed with the same command prefix:

`DF` *Diagnostic_Function_Code*

For greater detail on the functions performed by each test, type **HE DF** *Diagnostic_Function_Code* at the **>BKM>** prompt in the **OCD Command Shell**.

Simple RAM Test: Single Pass — DF 0 Command

The **DF 0** command runs a simple RAM test for a single pass.

Syntax

`DF`*units* **`0`** *args*

units — The units can be either **B** (byte), **W** (word), or **L** (long). By default, this parameter is set to Word.

args — This is an address range, spanning from *begin_addr* to *end_addr*.

Example 1

A simple RAM test is executed on a 128 word memory space. There will be no errors.

```
>BKM>DFW 0 00000 000FF
simple ram test running
test complete
>BKM>
```

Example 2

A simple RAM test is executed on a 128 word memory space. There will be a bad memory bit 12 at address 0000E. A bit will be stuck low.

```
>BKM>DFW 0 00000 000FF
memory failure: $0000E=$4555 not $5555 complete
```

Simple RAM Test: Continuous — DF1 Command

The **DF 1** command runs a simple RAM test continuously. This test can be stopped by typing **Ctrl+C**.

Syntax

DF*units* **1** *args*

units — These can be either **B** (Byte), **L** (Long), or **W** (Word). By default, this parameter is set to Word.

args — This parameter requires an address range from *begin_addr* to *end_addr*.

Example 1

A simple RAM Test is executed continuously on a 128 word memory space. There are no errors.

```
>BKM>DFW 1 00000 000FF
!ABORT! - [msg72000] User Command Abort
>BKM>
```

Example 2

A simple RAM test is executed continuously on a 128 word memory space. There is a bad memory bit 12 at address 0000E. A bit is stuck low.

```
>BKM>DFW 1 00000 000FF
memory failure: $0000E=$4555 not $5555
PASS# = 1
memory failure: $0000E=$4555 not $5555
PASS# = 2
UNTIL ^C
```

Complete RAM Test: Single Pass — DF 2 Command

The **DF 2** command runs a complete RAM Test for a single pass.

Syntax

DF*units* **2** *args*

units — This parameter can be either **B** (Byte), **W** (Word), or **L** (Long). By default, it is set to Word.

args — This is an address range that is specified from *begin_addr* to *end_addr*.

Example 1

A complete RAM test is executed on a 128 word memory space. There are no errors.

```
>BKM>DFW 2 00000 000FF
complete ram test running
test complete
>BKM>
```

Example 2

A complete RAM test is executed on a 128 word memory space. There is a bad memory bit 12 at address 0000E. A bit is stuck low.

```
BKM>DFW 2 00000 000FF
memory failure: $0000E=$efff not $ffff memory failure:
$0000E-$0000 not $1000
complete
BKM>
```

Complete RAM Test: Continuous — DF 3 Command

The **DF 3** command runs a complete RAM test continuously. This test can be stopped by typing **Ctrl+C**.

Syntax

DF*units* **3** *args*

units — This parameter can be either **B** (byte), **W** (word), or **L** (long).

args — This is an address range, specified from begin_addr to end_addr

Example 1

A complete RAM test is executed continuously on a 128 word memory space. There are no errors.

```
>BKM>DFW 3 00000 000FF
complete ram test running
>BKM>
```

Example 2

A complete RAM test is executed on a 128 word memory space. There is a bad memory bit 12 at address 0000E. A bit is stuck low.

```
>BKM>DFB 3 00000 000FF
complete ram test running
```

```
memory failure: $0000F=$ef not $ff memory failure:
$0000F=$00 not $10
PASS# = 1
memory failure: $0000F=$ef not $ff memory failure:
$0000F=$00 not $10
PASS# = 2
.
.
UNTIL ^C
```

CRC Test — DF 4 Command

The **DF 4** command runs a CRC Test over a specified range of memory.

Syntax

DF 4 *args*

args — An address range, specified from *begin_addr* to *end_addr.*

Example

A CRC Test is executed on a 128 word memory space.

```
>BKM>df 4 00000 000FF
CRC-16 test running
Completed... CRC-16 Value = 166D
>BKM>
```

Scope Loop: Read from Location — DF 5 Command

The scope loop routines are useful when troubleshooting with an oscilloscope. Read/write continuously from/to an address and Write Then Read data are supported routines.

The **DF 5** command consecutively reads from the specified address. This test can be stopped by typing **Ctrl+X**.

Syntax

DF*unit* **5** *args*

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long)

args — This specifies a base address *base_addr.*

Example

Scope Loop reading a location.

```
>BKM>df 5 E800000
DF 5 reading a location press ^X to abort
>BKM>
```

Scope Loop: Write to Location — DF 6 Command

The **DF 6** command consecutively writes a specified pattern to the address specified. This test can be stopped by typing **Ctrl+X**.

Syntax

DF*unit* **6** *args*

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long).

args — This parameter consists of a base address *base_addr* and a hex string *hex_string*.

Example

Scope loop writing data to a location.

```
>BKM>df 6 e800000 5555
DF 6 Writing a location press ^X to abort
>BKM>
```

Scope Loop: Write and Complement — DF 7 Command

The **DF 7** command consecutively writes a pattern to the address specified and then writes its complement. This test can be stopped by typing **Ctrl+X**.

Syntax

DF*unit* **7** *args*

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long).

args — This parameter consists of a base address *base_addr* and a hex string *hex_string*.

Example

Scope loop writing data then its complement to a location.

```
>BKM>DF 7 E80000 5555
DF 7 writing value then complementing press ^x to abort
>BKM>
```

Scope Loop: Write Rotating Value — DF 8 Command

The **DF 8** command writes a pattern to the address specified and rotates the pattern. The test can be stopped by typing **Ctrl+X**.

Syntax

DF*unit* **8** *args*

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long).

args — This parameter consists of a base address *base_addr* and a hex string *hex_string*

Example

Scope Loop writing rotating data to a location.

```
>BKM>df 8 e800000 0001
DF 8 Writing Rotating Value On Location press ^X to abort
>BKM>
```

Scope Loop: Write Then Read — DF 9 Command

The **DF 9** command consecutively writes a specified pattern to the address specified and then reads it back. This test can be stopped by typing **Ctrl+X**.

Syntax

DF*unit* **9** *args*

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long).

args — This parameter consists of a base address *base_addr* and a hex string *hex_string*

Example

Scope Loop writing data to a location, and then reading it back.

```
BKM>DF 9 E80000 5555
DF 9 Writing and reading location press ^x to abort
>BKM>
```

Bus Test: Address — DF A Command

The **DF A** command tests the address bus for the purpose of locating system shorts on your target board.

Syntax

DF*unit* **A** *start_address*

This command must be executed at the **> BKM >** prompt.

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long). Do not leave a space between **DF** and *unit*.

Example

```
>BKM>df a 1000 2000
Address Bus Test at 00001000 to 00002000. Bus size 16 bits
.........................
TEST PASSED
>BKM>
```

Bus Test: Data — DF D Command

The **DF D** command tests the data bus for the purpose of locating system shorts on your target board.

Syntax

DF*unit* **D** *address*

This command must be executed at the **> BKM >** prompt.

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long). By default, this option is set to Word. Do not leave a space between **DF** and *unit*.

address — This is the address of the data bus to test.

Example

```
>BKM>df d 100
Data bus test at 00000100 size 16 bits
.................................
TEST PASSED
>BKM>
```

3.1.15 Disassemble (DI)

This command disassembles the target code beginning at a specified address or code symbol. If no address or symbol is given, **DI** disassembles code starting at the program counter. **DI** disassembles one screen of opcodes (20 lines) as a default, or you can include an optional count parameter. Pressing **ENTER** immediately following a **DI** command disassembles the next count or page of instructions in memory.

Syntax

DI *hex_addr count*

hex_addr — This is an address in target memory at which to begin disassembling. *hex_addr* may also be a previously defined code symbol. The default for *hex_addr* is the program counter.

count — This is the number of instructions (in hex) that you wish to disassemble. The default is one screen, or approximately 20 lines.

Pressing **ENTER** immediately following a **DI** command repeats the command for the next *count* instructions.

Example

In this example, eight lines of code located at address 10000 are disassembled. The display shows the address, the opcodes, and the disassembled instructions.

```
>BKM>di 40400 8
$00040400 : 0x3D600004 :ppc lis R11, 0x4
$00040404 : 0x382B2800 :ppc addi R1, R11, 0x2800
$00040408 : 0x3DA00005 :ppc lis R13, 0x5
$0004040C : 0x39AD9720 :ppc addi R13, R13, -0x68e0
$00040410 : 0x3C400005 :ppc lis R2, 0x5
$00040414 : 0x38429720 :ppc addi R2, R2, -0x68e0
$00040418 : 0x48000165 :ppc bl 0x4057c
$0004041C : 0x48000000 :ppc b 0x4041c
>BKM>
```

See also the **SI** (Single Step) command.

3.1.16 Disassemble Without Opcode (DIO)

As with the **DI** command, this command disassembles the target code beginning at a specified address or code symbol. The difference between the **DI** command and the **DIO** command is that with the **DIO** command, no opcode row value is displayed. If no address or symbol is given, **DIO** disassembles code starting at the program counter. **DIO** disassembles one screen (20 lines) as a default, or you can include an optional count parameter. Pressing **ENTER** immediately following a **DIO** command disassembles the next count or page of instructions in memory.

Syntax

DIO *hex_addr count*

hex_addr — This is an address in target memory at which to begin disassembling. *hex_addr* may also be a previously defined code symbol. The default for *hex_addr* is the program counter.

count — This is the number of instructions (in hex) to disassemble. The default is one screen, or approximately 20 lines.

Pressing **ENTER** immediately following a **DIO** command repeats the command for the next *count* instructions.

Example

```
>BKM>DIO 40400 8
$00040400 lis R11, 0x4
$00040404 addi R1, R11, 0x2800
$00040408 lis R13, 0x5
$0004040C addi R2, R2, -0x68e0
$00040410 lis R2, 0x5
$00040414 addi R2, R2, -0x68e0
$00040418 bl 0x4057c
$0004041C b 0x4041c
>BKM>
```

See also the **DI** (Disassemble) and the **SI** (Single Step) commands.

3.1.17 Disassemble with Code Coverage (DIP)

This command disassembles the target code and shows the corresponding profiling data for each instruction. After each instruction, the emulator displays the number of samples that were taken in which the program counter (at the time of the sample) is equal to the value of the particular disassembled instruction. The **DIP** command disassembles starting at either the program counter, a specified *addr*, or a previously defined code symbol, for *count* number of instructions. The default count is one page of disassembled instructions.

Also included in the **DIP** output are symbol boundaries and tabulated profile information for each symbol. The sum of the number of hits and percentages on the instructions inside each symbol boundary will equal the hits and percentages shown for the symbol. The numbers shown are accumulated over total profile time, and not for the last sample period, which can be shown by utilizing the PFA graphical display.

Syntax

DIP *hex_addr count*

hex_addr — address or symbol in target memory to begin disassembling. The program counter is the default.

count — This is the number of instructions (in hex) that you wish to disassemble. The default is one screen.

Pressing **ENTER** immediately following a **DIP** command repeats the command for the next *count* instructions.

NOTE: In order to have meaningful output with the **DIP** command, first run your code under the PFA profiling (see **PF RUN** command for additional information). Then use the **DIP** command.

NOTE: If no samples were taken on a particular instruction, no data will be shown for the number of hits and percentage of time.

NOTE: Percentages are calculated to two decimal places, unlike the PFA graphical display, where percentages are calculated using integer math.

Example

In this example, disassemble code while displaying instruction-level profiling data, starting at the address of *addone* (a previously defined symbol). In addition to the number of hits and percentages shown for each instruction, the **DIP** command tallies the total hits for each symbol encountered during the disassembly.

```
BKM>dip addone
-----------------------------------------------------------------------------
addone                                                          1389
             (0.148%0
-----------------------------------------------------------------------------
00008242    link.w   a6,#$0           641          (0.068%)
00008246    move.w   $8(a6),d0        F1           (0.07%)
0000824A    addq.w  #$1,d0            107          (0.011%)
0000824C    unlk     a6               351          (0.037%)
```

```
0000824E    rts                      219          (0.023%)
-----------------------------------------------------------------------------
main                                              591014       (63.279%)
-----------------------------------------------------------------------------
00008250     link.w    a6,#$FFFE
00008254     lea.l     $84(a5),a2
00008258     moveq.l   #$11,d1
0000825A     move.l    d1,(a2)
0000825C     bra.w     $82EE
00008260     move.b    $C(a5),d1                 220   (0.023%)
00008264     dc.w             $49C1              104  (0.011%)
00008266     cmp.l            (a2),d1             72   (0.07%)
00008268     bge.w            $826E              289  (0.030%)
0000826C     clr.l            d2                  67   (0.07%)
0000826E     clr.w            d3                  78   (0.08%)
BKM>
```

3.1.18 Display Memory (DM)

This command returns the contents of the memory location(s) requested. The command will display not only the address and the Hex data, but also the ASCII representation of that data. In addition, using the syntax indicated below, the **DM** command allows both memory-relative and register-relative addressing.

Syntax

DM*unit args*

unit — This parameter can be either **B** (byte), **W** (word), **L** (long), or **S** (string). By default, this is set to Word. Do not leave a space between **DM** and *unit*.

args — *base_addr number_of_units*. Leave a space between *base_addr* and *number_of_units*.

base_addr may be a previously defined symbol.

number_of_units is a hex value. This argument is optional.

- If *unit* is set to **B** (byte), **W** (word), or **L** (long), and you do not enter a value for *number_of_units*, then *number_of_units* defaults to 8.
- If *unit* is set to **S** (string), and you do not enter a value for *number_of_units*, then *number_of_units* defaults to 90 (5A in hex.)
- If *unit* is set to **S** (string), and you do enter a value for *number_of_units*, then the number of characters you specify are displayed unless a null character (**0x00**) is encountered first. A null character will always terminate the output.

Response: *addr hex_data ascii_data*

The **DM** command can also take an indirect address and a register-relative address as parameters. The syntax is as follows:

DM*unit (*addr) number_of_units*
DM*unit (offset register) number_of_units*

Example 1

In this example, display 32 (20 Hex) words of memory starting at location 200010:

```
>BKM>DM 200010 20
00200010: FFFF DFFF FFFF FFFF FFFF FFFF FFFF FFFF     ................
00200020: FFFF FFFE FFFF FFFD FFFF FFFF FFFF FFFF     ................
00200030: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF     ................
00200040: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF     ................
>BKM>
```

Example 2

In this example, display five bytes of memory starting at location 300300:

```
>BKM>dmb 300300 5
00300300: FF FF FF FF FF     .....
>BKM>
```

Example 3

In this example, display 50 bytes of memory at the relative-register address of 1116:

```
>BKM>dmb (*1116) 50
00000100: FD F0 BF AC B7 7C FA BF 9A 7C 4E 74 BA EC 3B FF.....|...|Nt..;.
00000110: FC FC FD EF 4F F8 9F F5 FF DC FF FF BA 7C B2 39....O........|.9
00000120: 7B 70 A7 85 EE FC F7 2F DA FC 67 C6 7D F4 FF FE {p...../..g.}...
00000130: E8 7C FE ED 9F F8 3B FE E7 7C F5 BE DF BC 7F 8F .|....;..|......
00000140: 87 D8 B2 6F F9 FC B3 FF DB 8C ED BF EF F4 FE 6D ...o...........m
>BKM>
```

Example 4

In this example, display 10 bytes of memory at the memory-relative address of A2:

```
>BKM>dmb (a2) 10
    00001064: 00 08 00 1E 00 00 46 45 42 00 00 00 00 00 00 00 .....FEB....
BKM>dr a2

    00001068: 00 00 46 45 42 00 00 00 00 00 00 00 00 00 00 00 ..FEB.......
BKM>
```

3.1.19 Display Memory Double (DMD)

The **DMD** command is only for relevant architectures. This command obtains data via full 64-bit read operations.

Syntax

`DMD` *args*

args — *base_addr number_of_units*; a previously defined symbol may be used for *base_addr.*

Response: *addr hex_data ascii_data*

The **DMD** command can also take an indirect address and a register-relative address as parameters. The syntax is as follows:

DMD *(*addr) number_of_units*

DMD [*offset*] *(register) number_of_units*

Example 1

In this example, display memory starting at location 200010:

```
>BKM>dmd 200010 20
00200010: FFFFDFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
00200020: FFFFFFFEFFFFFFFD FFFFFFFFFFFFFFFF     ................
00200030: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
00200040: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
00200050: FFFFFFFFFFFFFFFF FDFFFFFDFFFFFFBF     ................
00200060: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
00200070: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFF7F     ................
00200080: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFE     ................
00200090: FFFFFFFFFFFFFFF7 FFFFFFFFFFFFFFFF     ................
002000A0: FFFFFFFFFFFFFFFF F7FFFFFFFFFFFFFD     ................
002000B0: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
002000C0: FFFFFFFFDFFFFFFF FFFFFFFFFFFFFFFF     ................
002000D0: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
002000E0: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
002000F0: FFFFFFFBFFFFFFFF FFFFFFFFFFFFFFFF     ................
00200100: FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF     ................
>BKM>
```

Example 2

In this example, display memory at the relative-register address of 1116:

```
>BKM>dmd (*1116) 100
00000000: 9421FFE090610008 9041000000000000     .!...a...A......
00000010: 0000000000000000 0000000000000000     ................
00000020: 0000000000000000 0000000000000000     ................
00000030: 0000000000000000 0000000000000000     ................
```

3.1.20 Display Registers (DR)

The **DR** command displays the contents of a particular register or registers. To display all registers, enter **DR ALL**. If no arguments are specified, then the general purpose registers for most processor families will be displayed/returned. The active stack is flagged with an asterisk *.

Syntax

DR *options*

options can be either *reg_name*, the name of the register you wish to display the value for, or *group_name*, the name of the register group whose registers you want to display. If you do not specify an option then all of the registers are displayed. You can specify more than one register at a time, leaving a space between each *reg_name* or *group_name*.

Response: *option = value*

Example 1

Displaying all registers with the **DR ALL** command would take up too much space here. In this example, the **DR** command is used to display all general purpose registers.

```
>BKM>dr
R00     = 00E17F2C R01     = 00EFFF30 R02     = 00E07FF0 R03     = 00000001
R04     = 00000000 R05     = 00EFFF38 R06     = 00000001 R07     = 00EFFF3C
R08     = 000D1788 R09     = 00000008 R10     = 00000000 R11     = 00000000
R12     = 00000000 R13     = 00E22C00 R14     = 00E2847C R15     = 00000000
R16     = 00000000 R17     = 00E294F8 R18     = E396FFBE R19     = ADAEF77B
R20     = 762EFEDD R21     = F813F3A2 R22     = F3E71E7F R23     = A836CDEE
R24     = 7B7F9EFE R25     = 593A65BF R26     = 2513F4FB R27     = E2FB608F
R28     = EC6DCFDF R29     = A9F275EB R30     = 00E23144 R31     = 00E24310
CR      = 20000000 MSR     = 00009042 LR      = 00E19870 SRR0    = 00E1986C
SRR1    = 00009002 SPRG0   = 00000000 SPRG1   = 00000000 SPRG2   = 00000000
SPRG3   = 00000000 XER     = 00000300 CTR     = 00000000 PC      = 00E198BC
>BKM>
```

Example 2

In this example, set values in some registers. Then display these registers in a different order:

```
>BKM>SR R00 1000000 R03 120000 R08 1234 R16 55
>BKM>DR R08 R16 R00 R03
R08    = 00001234 R16     = 00000055 R00     = 01000000 R03     = 00120000
>BKM>
```

See also the **SR** (Set Register) command.

3.1.21 Display Trace Non-Real-Time (DT)

The **DT** command is used to display both real-time and non-real-time trace. The options for the **DT** command are different for the real-time modes.

The non-real-time trace buffer only has valid history if tracing was previously enabled, either directly with the Trace Enable command or indirectly by setting a data breakpoint, for example. The trace buffer contains PC-only history information.

Syntax

DT *arg1 arg2 num*

arg1 — This can be specified as either **B** (backwards), or **F** (forwards). By default this option is set to **B**.

arg2 — This option is set to **F** - full trace with data movements.

num — This is the number of locations to be displayed. By default, this is 20 lines of information.

Example

In this example, the non-real-time trace display of three instructions that were executed are shown:

```
BKM>DTB 3

19.      00ea6070 2080      move.b     d1,(a0)
18.      00ea6074 10BC00AA  move.l     #$AA,(a0)
17.      00ea6076 2210      move.l     (a0),d1
BKM>
```

See also the **CF** (Configure) and the **TE** (Trace Enable) commands.

3.1.22 Display License Key String (ESTKEY)

The **ESTKEY** command displays the actual encrypted license key string for your emulator.

Syntax

ESTKEY

Example

```
>BKM>estkey
```

```
AC1WXYJNB140G7C6GFD3H28DBK909VUODBO (*) Main Key
>BKM>
```

3.1.23 Display License Key Support Information (ESTKEY DISPLAY)

The **ESTKEY DISPLAY** command returns the current Wind River license support information.

Syntax

`ESTKEY DISPLAY`

Example

Below is an example of this command executed on a Wind River ICE SX unit.

```
>BKM>estkey display
                 Key In Use: Main Key
           FLEXLM licensing: Enabled
              Serial Number: W0000000
                   Group Id: 0
>BKM>
```

3.1.24 Fast Step (FS)

The Fast Step command allows you to step continuously through a specified code range. To use this command, first make sure that your PC is within the code range that you specify for fast stepping. If you do not specify a code range to Fast Step through, the **FS** command reverts to a simple single step, stepping one instruction at the current program counter.

Syntax

`FS` *start_address end_address*

In this case, *start_address* and *end_address* specify the address range for the debugger to continue stepping until the program counter is outside the range.

Both *start_address* and *end_address* are optional parameters.

If neither parameter is specified, the **FS** command reverts to a simple single step, stepping one instruction at the current program counter.

If only one parameter is given with the **FS** command, for example:

`FS` *end_address*

...then the *start_address* is taken as the current program counter. In this case, **FS** steps until it is outside of the range of the current program counter, and the *end_address* is given.

Examples

In the following examples, memory is at 5000 and the current program counter is at 5000.

```
BKM>FS
@5004
BKM>FS 500E
@5010
BKM>FS 500E 5010
@5004
BKM>
```

3.1.25 Fast Step Next (FSN)

The Fast Step Next command is similar to a Fast Step in that it lets you step continuously through a specified code range. The difference between them is that instead of stepping into each line of code, **FSN** steps over each line. To use this command, first make sure that your PC is within the code range that you specify for fast stepping. If you do not specify a code range to Fast Step through, the **FSN** command reverts to a simple step over command, stepping over one instruction at the current program counter.

Syntax

FSN *start_address end_address*

In this case, *start_address* and *end_address* specify the address range for the debugger to continue stepping until the program counter is outside of the range.

Both *start_address* and *end_address* are optional parameters.

If neither parameter is specified, the **FSN** command reverts to a step over, stepping one instruction at the current program counter.

If only one parameter is given with the **FSN** command, for example:

FS *end_address*

then the start_address is taken as the current program counter. In this case, FSN steps until it is outside of the range of the current program counter, and the end_address is given.

Example

```
>BKM>fsn
@00040d88
>BKM>fsn 40d84 40e7c
>RUN>
@000406c8
>BKM>
```

See the **FS** command for more information.

3.1.26 Start Code Execution (GO)

The **GO** command deactivates debug mode and begins executing user code on the target. Code execution continues until either a breakpoint occurs, the system is reset, a double bus fault occurs, or you stop the target by typing **Ctrl+C** or **Ctrl+X**. If breakpoints are enabled, a **BKMD** instruction is inserted into the target code at each breakpoint address (**TRAP** instructions are used in simulation). This allows real-time execution of target code and still allows for breakpoints to be set.

If the location where execution begins contains a breakpoint, then that breakpoint is temporarily disabled until the program is stepped off of the breakpoint. If an assertion has been set then, the **GO** command enables the trace buffer and checks for break conditions after each instruction is executed. After the **GO** command is issued, the emulator displays the **>RUN>** prompt, indicating that the target is running in real-time.

Syntax

GO *flag addr*

flag — This parameter can be set to either **T** (Trace Enabled), or **D** (Trace Disabled. By default, this is set to *no change*.

The flag argument is a way to combine the Trace Enable or Trace Disable commands with the **GO** command. This is optional, and if it is not specified, then there is no change to the trace conditions previously set. The tracing mode will remain in effect until it is specifically changed by another **GO** command or by the Trace Enable or Trace Disable commands.

addr — This parameter is the hex address where execution mode will resume.

Example 1

In this example, examine the instructions located at address 10000. Set a breakpoint and verify that it is the only condition set up. Disable tracing. Finally, issue the **GO** command, allowing Execution Mode to operate in Real-Time. Notice the prompts that are displayed:

```
BKM>DI 10000 8
00100000  2200    move.l   D0,D1
00100002  4282    clr.l    D2
00100004  D401    add.b    1,D2
00100006  E289    lsr.l    1,D1
00100008  66FA    bne.b    $10004
0010000A  E20A    lsr.b    #1,D2
0010000C  55C2    scs      D2
0010000E  60FE    bra.b    $1000E

BKM>SB 1000E

BKM>DB

Code Break-Points

1. 01000E count = 0001 actual = 0000 enabled

BKM>GO 10000

.RUN>

!BREAK! - Breakpoint at 001000E

BKM>
```

Example 2

In this example, assume only one address breakpoint is set at location 1000E. First, issue a **GO** command with trace enabled. Once the breakpoint is taken, issue another **GO** command from the same place. Finally, explicitly disable tracing and issue the **GO** command again. Notice that tracing is still enabled for the second **GO** command:

```
BKM>GOT 10000

TRC>

!BREAK! - Code Breakpoint at 001000E

BKM>GO 10000

TRC>

!BREAK! - Code Breakpoint at 001000E

BKM>GOD 10000

RUN>

BREAK! - Breakpoint at 001000E
```

```
BKM>
```

See also **DI** (Disassemble) command, **SB** (Set Breakpoint) command, **DB** (Disable Breakpoints) command, **TE** (Trace Enable) command, and **TD** (Trace Disable) command.

3.1.27 Start Performance Analysis (GOP)

This command starts execution of the target system with Performance Profiling activated. You can only issue a **GOP** command while the target is in Background Mode. The **GOP** command begins the sampling process from the specified address or code symbol. Issuing a **GOP** command automatically opens the **PFA** graphic display (histogram) if the graphics switch is turned on in the **PF** command. If the graphics switch is turned off, the **GOP** command issues the **>PFA>** prompt. At this prompt, issue a **DIP** (Disassemble with Code Coverage) command to view the code profiling information by disassemble instruction or by symbol, respectively.

Syntax

GOP *addr*

By default, *addr* is a program counter. A previously defined symbol may be substituted for *addr*.

Examples

Table 3-1 **GOP Examples**

BKM>GOP	Starts profiling at the current program counter and opens the **PFA** histogram and chart display.
BKM>GOP 1000	Starts profiling at 1000 and opens the **PFA** display if the graphics are on in the **PF** command.
BKM>GOP *main*	Starts execution and profiling at main and issues the **>PFA>** prompt if graphics are turned off.

NOTE: The **GOP** command only operates from a stopped system (>**BKM**> prompt). To start performance analysis from a running system (>**RUN**> prompt), use the **PF RUN** command.

NOTE: You must disable any active breakpoints before executing the **GOP** command. Use the **DB** (Disable Breakpoints) command or **RB** (Remove Breakpoints) command.

NOTE: Performance Analysis does not operate in Trace mode. You must explicitly disable tracing (using the **TD** command).

NOTE: The **GOP** command either directly opens the **PFA** graphic display or issues the **>PFA>** prompt, depending on whether or not the graphics switch is turned on or off in the **PF** command (option 5). If graphics are turned on, the **GOP** command opens the display with parameters (**PFA** range, graph type, sampling period) that are currently set in non-volatile memory. To change these parameters, use the **PF** command, or change them from within the graphic display.

NOTE: From the **>PFA>** prompt, use the **DIP** command to disassemble code with profiling data. Additionally, issue a **DM**, **SM**, **DR**, **SR**, or **DI** command in the form of a special target snapshot from the **>PFA>** prompt. You cannot issue a trace, breakpoint, or configuration command.

NOTE: Enter **Ctrl+C** from the **>PFA>** prompt to discontinue profiling, stop the target, and return to a **>BKM>** prompt. Exiting the graphic display with the **F9** key returns you to the **>PFA>** prompt, so you must use **Ctrl+C** to stop the code profiling.

3.1.28 Synchronized Start Code Execution (GOS)

Syntax

```
GOS
```

Resume code execution for the group of cores defined by the **SCTRL** command. The resume operation is synchronized on the same edge of the JTAG clock signal.

This command also activates hardware polling and the low latency breakpoint cross-triggering as configured by the **SCTRL** command.

This is a multi-core command and is only supported for the Wind River ICE SX.

See also the **HALTS** and **SCTRL** commands.

3.1.29 Halt (HA)

Syntax

`HA`, `HALT`, or `Ctrl+C`

When entering background mode fails due to a register value set incorrectly by a software application, the emulator will wait for the amount of time set in the **CF TOUT** option before issuing a non-maskable break to the target. If this occurs, the following message is displayed:

```
>RUN>
!BREAK! - [msg12007] User forced halt; PC = 0x000142F0
>BKM>
```

You can stop the target running by typing **HA** or **HALT** at the **>RUN>** or **>TRC>** prompt. You can also stop the target or a command that is currently executing by typing **Ctrl+C** or **Ctrl+X**.

3.1.30 Synchronized Halt (HALTS)

Syntax

`HALTS`

Halt code execution for the group of cores defined by the **SCTRL** command. The halt operation is synchronized on the same edge of the JTAG clock signal.

This is a multi-core command and is only supported for the Wind River ICE SX.

See also the **GOS** and **SCTRL** commands.

3.1.31 Help Command (HE)

The **HE** command displays the emulator's help menu. The menu shows the syntax of the command set. You can see additional help for each command by typing the command name as an argument.

Syntax

`HE` *command*

command is the command for which help will be displayed.

Entering **HE** with no arguments displays all available help.

Example 1

Display all commands:

```
>BKM>he
==== MISCELLANEOUS ===    = SOFTWARE BREAKPOINTS =    == REGISTERS & MEMORY =
SH: Show History           DB: Display Breakpoints    DR: Display Registers
CF: Configure              BD: Disable Breakpoints    SR: Set Registers
CS: Chip Selects           BE: Enable Breakpoints     DM: Display Memory
TF: Target Flash           RB: Remove Breakpoints     MM: Memory Modify
IN: Initialize             SB: Set Code Breakpoint    SM: Set Memory
SC: System Configuration                              BF: Block Memory Fill
                                                      BM: Block Memory Move
                                                      SS: Search for String

===== EXECUTION =======   = STEPPING & DISASSEMBLY =  ====== TRACING ======
GO   : Start Execution     SI : Step Instructions     TD: Trace Disable
^C/^X: Stop Execution      FSN: Step Over             TE: Trace Enable
HALT : Stop Execution      DI : Disassemble Code      DT: Display Trace
DF   : Run Diagnostics     FS : Fast Step In          SB: Set a Trace Point
                           ST : Stack Trace
                           STO: Step Out
                           ASM: Inline Assembler

= HARDWARE BREAKPOINTS =    == PERFORMANCE ANALYSIS==
HBC :Set HB on Code         PF : Performance Analysis
HBD :Set HB on Data         DIP: Dis. with Perf. Anal.
HBTC:Set HB on N Cycle
  For Help on a specific command, type:  HE {command}{parm1}{cr}

>BKM>
```

Example 2

In this example, display help for the **ASM** command:

```
>BKM>he asm
Command         Description                 Syntax
=======    ===================  ====================================
ASM         Inline Assembler     Form 1   ASM <start_addr> <instruction>
                                 Form 2   ASM <start_addr>
                                            then  <instruction>.
                                            or  <enter>.......Skip this addr
                                            or  <.>...........Stop
                                 Form 3   ASM <no addr>
                                          takes the PC as the starting addr.
>BKM>
```

3.1.32 Hardware Interface Configuration (HIC)

(Wind River Probe only)

Use the **HIC** command to change the properties of the Wind River Probe electrical interface.

Syntax

```
HIC
```

Entering **HIC** with no arguments returns a list of all user-changeable electrical properties:

```
>BKM>hic
Clock Frequency in KHz                      CLKK[25...100000] = 16000
Clock Phase                                 CLKPHASE[-180..+900] = 0
Clock External synchronization              RTCK[ENABLE,DISABLE] = DISABLE
CLK Drive Strength                          CLKSTRENGTH[1..4] = 2
TMS Drive Strength                          TMSSTRENGTH[1..2] = 1
TDI Drive Strength                          TDISTRENGTH[1..2] = 1
Target Interface voltage Override           VOLTAGE[3.3V,2.5V,1.8V,VIO] = VIO
Target Interface termination voltage       TERMVOLTAGE[2.5V-3.3V,1.8V-2.5V] =
2.5V-3.3V
CLK Termination             CLKTERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = ACTIVE_HI
TMS Termination             TMSTERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = ACTIVE_HI
TDI Termination             TDITERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = ACTIVE_HI
TDO Termination             TDOTERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = ACTIVE_HI
GPIO0 Termination         GPIO0TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = ACTIVE_HI
GPIO1 Termination         GPIO1TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = ACTIVE_HI
GPIO2 Termination         GPIO2TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = PASSIVE
GPIO3 Termination         GPIO3TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = PASSIVE
GPIO4 Termination         GPIO4TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = PASSIVE
GPIO5 Termination         GPIO5TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = PASSIVE
GPIO6 Termination         GPIO6TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = PASSIVE
GPIO7 Termination         GPIO7TERM[ACTIVE_HI,ACTIVE_LO,PASSIVE] = PASSIVE
GPIO Output Enable Value  (GPIO7..GPIO0) GPIOXOE[VALUE] = 0x01
GPIO Output Value         (GPIO7..GPIO0) GPIOXOV[VALUE] = 0x03
GPIO Intput Value         (GPIO7..GPIO0) = 0x27
>BKM>
```

Example

In this example, set the strength of the TCK signal to 4.

```
>BKM> HIC TCKSTRENGTH 4
```

When all parameters are configured, you can save the configuration in XML format using the command **HIC BRDDUMP**:

```
>BKM>hic brddump
<‘  CLKK>16000</CLKK>
<CLKPHASE>0</CLKPHASE>
<RTCK>DISABLE</RTCK>
```

```
<CLKSTRENGTH>2</CLKSTRENGTH>
<TMSSTRENGTH>1</TMSSTRENGTH>
<TDISTRENGTH>1</TDISTRENGTH>
<VOLTAGE>VIO</VOLTAGE>
<TERMVOLTAGE>2.5V-3.3V</TERMVOLTAGE>
<CLKTERM>ACTIVE_HI</CLKTERM>
<TMSTERM>ACTIVE_HI</TMSTERM>
<TDITERM>ACTIVE_HI</TDITERM>
<TDOTERM>ACTIVE_HI</TDOTERM>
<GPIO0TERM>ACTIVE_HI</GPIO0TERM>
<GPIO1TERM>ACTIVE_HI</GPIO1TERM>
<GPIO2TERM>PASSIVE</GPIO2TERM>
<GPIO3TERM>PASSIVE</GPIO3TERM>
<GPIO4TERM>PASSIVE</GPIO4TERM>
<GPIO5TERM>PASSIVE</GPIO5TERM>
<GPIO6TERM>PASSIVE</GPIO6TERM>
<GPIO7TERM>PASSIVE</GPIO7TERM>
<GPIOXOE>0x01</GPIOXOE>
<GPIOXOV>0x03</GPIOXOV>
<GPIOXIV>0x27</GPIOXIV>

>BKM>
```

Copy this output to a board file that you can use for every connection.

3.1.33 Internal Code Breakpoint (IHBC)

The **IHBC** command is used to set an internal hardware breakpoint on code execution. This command initializes a program counter comparator with the address of the breakpoint. The processor breaks prior to executing the instruction at the breakpoint address.

Unlike software breakpoints (which require the emulator to overwrite the breakpoint instruction with a **HALT** or Software Emulation Exception Instruction and can only be set in read/write memory), internal hardware breakpoints can be used to set breakpoints in read only memory (Flash or ROM).

This command is only supported with microprocessors that have program counter comparators.

Syntax

`IHCB` *addr*

addr — This is the address of the instruction.

Example

Set an internal code breakpoint at address 0x1000.

```
>BKM>ihbc 1000
>BKM>
```

3.1.34 Internal Data Breakpoint (IHBD)

The **IHBD** command is used to set an internal hardware breakpoint on data accesses. The command initializes an address comparator with the address of the breakpoint. This command may have a data argument, and can be further qualified when the operand access is a read or write.

The **IHBD** command is only supported with microprocessors that have address and data comparators.

Syntax

IHBD *optional_qualifier addr optional_data*

optional_qualifier — This parameter is either **R** (read), **W** (write), or it can be left without any qualifier, which permits any access.

addr — This parameter is the address of the operand.

optional_data — This parameter is optional, and you can use it to specify a condition that you want met before a break occurs. Using this option you can cause the target to break only when a specified value is read or written at the address specified in *addr*.

Examples

Set an internal breakpoint at address 0x1000 and break on any access (read or write) to this address.

```
BKM>ihbd 1000
```

Set an internal data breakpoint on any read to address 0x1000.

```
BKM>ihbdr 1000
```

Set an internal data breakpoint on any write to address 0x1000.

```
BKM>ihbdw 1000
```

Set an internal data breakpoint on a read of value of 0x00000003 at address 0x1000.

```
BKM>ihbdr 1000 00000003
```

3.1.35 Initialize System (IN)

The **IN** command attempts to initialize Background Mode communications. Once proper communications are established, the emulator transfers the chip-select table (modifiable with the **CS** command) and the stored register settings (which can be modified with the **SC** command) to the target system using the emulator. The **IN** command is automatically sent after every emulator power-up or reset. To initialize Background Mode communications without writing the chip-select table and register settings to the target, use the **INN** command.

Syntax

```
IN
```

This command initializes and downloads the chip-select and register settings, and initializes communications with your target.

Initialization Messages

The emulator goes through the following sequence after the **IN** command is issued, assuming that it is able to properly establish background mode communications with the target:

```
>BKM>in
****************************************************************************
Wind River ICE Initialization Sequence.
Copyright (c) Wind River Systems, Inc., 1999-2004. All rights reserved.
****************************************************************************
Support Expires....... 5/17/07
Target Processor...... MPC8260
Wind River ICE             Group ID#= 0
Wind River ICE             Serial#= W0000000       Firmware= vn2.3a
Type CF For a Menu of Configuration Options
Initializing Background Debug Mode.............Successful
>BKM>
```

NOTE: The information displayed at an **IN** command varies slightly depending on the type of system you are using. For example, the support expiry date and the target processor may be different than what is displayed here.

3.1.36 Initialize Communications Only (INN)

The **INN** command is similar to the **IN** command in that it places a target in background mode. The difference is that the **INN** command does not download any of the register settings or the chip-select settings that are stored in the emulator. The **INN** command only places the target in background mode. If you use this command, your target settings and the settings stored in the emulator may not be the same. The next time you power on your emulator, an **IN** command is automatically performed and so your target settings are overwritten by the emulator settings. Make sure that you are aware of this issue and try to keep your target settings and the emulator settings in sync at all times.

Syntax

INN

Initialization Messages

The emulator goes through the same sequence as the **IN** command when the **INN** command is issued, assuming that it is able to properly establish communications with your target:

```
>BKM>inn
*****************************************************************************
Wind River ICE Initialization Sequence.
Copyright (c) Wind River Systems, Inc., 1999-2004. All rights reserved.
*****************************************************************************
Support Expires....... 5/17/07
Target Processor...... MPC8260
Wind River ICE            Group ID#= 0
Wind River ICE             Serial#= W0000000       Firmware= vn2.3a
Type CF For a Menu of Configuration Options
Initializing Background Debug Mode.............Successful
>BKM>
```

See the **IN** command for more information about initializing your system.

3.1.37 Initialize and Trap Exceptions (INE)

The **INE** command is a special initialization command which allows the emulator to trap and decode exceptions that are not properly handled by the application code. You may only use this command from a **> BKM >** prompt or **>ERR>** prompt. The **INE** command cannot be used as a target snapshot.

The **INE** command performs the standard initialization sequence (refer to the **IN** command for more information), including chip-select and register download. If valid communications are established with the target, the **INE** command also initializes all exception vectors, not including the restart vector at the first eight longs, by writing a general exception address in each location in the exception vector table.

After initializing with the **INE** command and issuing a **GO** command, if you hit a breakpoint, the emulator determines if the breakpoint was at the location pointed to by the general exception. This is the address at the end of the event to which all uninitialized vectors point. If the target breaks at this location, the emulator uploads the exception stack frame from the target and decodes it for you.

If you take the general exception and stopped at a breakpoint at the end of the event, the emulator displays:

- A !BREAK! - Exception Trap Handler message
- The location in memory of the exception stack frame
- The program counter where the exception took place and/or the faulted address, if they are not the same
- The exception type and vector offset
- The target registers at the time of the exception

The **INE** command assumes that the vector table is located at 0, unless you include a non-zero VBR as an optional parameter. If your vector table is in ROM, you can use the **INE** command by giving it an arbitrary RAM-based VBR, and then manually changing the VBR using the Set Register command (**SR VBR** *RAM_Address*).

You only need to use the **INE** command once after system power-up because the installed vectors remain loaded.

Syntax

INE *VBR_addr*

VBR_addr — this is an optional parameter that specifies a non-zero VBR.

Examples

Table 3-2 **INE Examples**

BKM>INE	Initializes Background Mode communications (displays the emulator banner), downloads the chip-select table and registers, sets all exception vectors (except the restart vector) to an address just after the EVT, and sets a breakpoint at that address. The VBR is assumed to be zero.
BKM>GO	If a bus error occurs after code starts executing (**GO**), the emulator traps the error and decodes the exception stack frame.

```
RUN>
!BREAK! - Exception trap handler

Exception Stack Frame Starts at $0000F4E8
Faulted Address = $0000FB90
Current Instruction Program Counter = $000500E
```

```
Special Status Word = 0225
Bus Error - Exception vector offset = 0008
(Displays Registers at the time of the Bus Error)
```

3.1.38 Memory Modify (MM)

Use the **MM** command to modify memory without a read verify.

Using the syntax described below, the **MM** command allows both memory-relative and register-relative addressing.

The **MM** command differs from the **SM** command in that the **MM** command only modifies the memory at a specified address; it does not read back the value to verify that it wrote correctly.

Syntax

MM*unit base_addr data*

unit — This can be either **B** (byte), **W** (word), or **L** (long). By default, the option is set to word. Do not leave a space between **MM** and *unit*.

base_addr — This is the beginning address of the memory that you want to modify.

data — This is the data to be written to the address specified by *base_addr*.

The Memory Modify command can also take an indirect address and a register-relative address as parameters. The syntax is as follows:

MM*(*addr) data*

Example 1

In this example, set a word at 1000 to 0C, set a long at 0FFE to 100, and set a byte at FB0 to 0A:

```
BKM>MM 1000 0C
BKM>DM 1000
    00001000:  000C 4E00 4645 4200 4D41 5200 4150 5200 ..N.FEB.MAR.APR.
BKM>MML 0FFE 100
BKM>DM 0FFE
    00000FFE:  0000 0100 4E00 4645 4200 4D41 5200 4150 ..N.FEB.MAR.APR
BKM>MMB FB0 0A
BKM>DM FB0
    00000FB0:  0A00 0000 0000 0000 0000 0000 0000 0000 . . . . . . . . . .
```

Example 2

In this example, modify the memory at the relative-register address of 1116:

```
>BKM>mm (*1116) 1234
>BKM>dm (*1116)
00000000: 1234 FFE0 9061 0008 9041 000C 7C7A 02A6     .4...a...A..|z..
>BKM>
```

3.1.39 Memory Modify — Double (MMD)

The **MMD** command is for specific architectures. This command modifies memory via full 64-bit read operations.

Syntax

MMD *args*

args — *base_addr* data; a previously defined symbol may be used for *base_addr*.

Response: *addr hex_data ascii_data*

The Memory Modify Double command can also take an indirect address and a register-relative address as parameters. The syntax is as follows:

MMD *(*addr) number_of_units*

Example 1

In this example, modify the memory starting at location 200010, and then display it using the **DMD** command:

```
>BKM>mmd 100020 1234
>BKM>dmd 100020
00100020: 0000000000001234 7C6802A690610018     .......4|h...a..|
00100030: 4800001D00E07FF0 00E22C0000E2852C     H........,....,
00100040: 00E02B1800000000 00E02A9C7C4802A6     ..+.......*.|H..
00100050: 80620008BC830010 9003000080810000     .b.............
>BKM>
```

Example 2

In this example, modify memory at the relative-register address of 1116 and display it using the **DMD** command:

```
>BKM>mmd (*1116) 1234
>BKM>dmd (*1116)
00000000: 0000000000001234 9041000C7C7A02A6     .......4.A..|z..
00000010: 7C5B02A690610010 904100147C6000A6     |[...a...A..|`..
00000020: 0000000000001234 7C6802A690610018     .......4|h...a..
00000030: 4800001D00E07FF0 00E22C0000E2852C     H........,....,
>BKM>
```

See also the **DMD** command.

3.1.40 Memory Management Unit (MMU)

NOTE: Use of this command requires an MMU-enabled license.

The Memory Management Unit commands allow you to add and manipulate data included in the **MMU** table. **MMU** commands are not available for all processors. If your processor does support **MMU** commands, you need to enable the **MMU** configuration option. See the **CF** command for more information on enabling the **MMU** commands.

NOTE: These **MMU** commands only set up the Memory Management table on the emulator unit. They do not set up this information on the target. Any information programmed into the Memory Management table should correspond to the information that your code is setting up on the target.

MMU Commands

Table 3-3 **MMU Commands**

MMUA	Add an entry to the MMU table.
MMUL	Displays the MMU table.
MMUD	Deletes entries from the table, either one at a time or all at once.

Each of the commands is described in the sections that follow.

Memory Management Unit — Add (MMUA)

This command lets you add an entry to the table in your Wind River emulator.

Syntax

There are two ways that you can add an entry to your table. The first way is to enter all of the data on the same line, as shown.

MMUA *logical_address physical_address mask*

logical_address — This is the virtual address that the applications use to locate a device on your target.

physical_address — This is the physical location of the device on your target.

The second method is to type **MMUA** and press **ENTER**.

```
MMUA
```

This method prompts you for each of the required parameters (*logical_address*, *physical_address*, and *mask*).

Examples

The first example illustrates adding an entry to the **MMU** table in the emulator using the **MMUA** *logical_address physical_address mask* syntax. The table is then displayed using the **MMUL** command.

```
>BKM>mmua c000000 0 ff000000
>BKM>mmul
Index |   Logical     |   Physical    |   Mask
------|---------------|---------------|-------------
01    |  0x0c000000   |  0x00000000   |  0xff000000
>BKM>
```

The second example illustrates adding an entry to the MMU table in the emulator using the **MMU ENTER** syntax.

```
>BKM>mmua
Logical Address     :d0000000
Physical Address    :100
Mask                :ff000000
>BKM>mmul
Index|   Logical  |   Physical    |    Mask
-----|------------|---------------|-------------
01   |  0x0c000000|  0x00000000   |  0xff000000
02   |  0x0d000000|  0x00000100   |  0xff000000
>BKM>
```

NOTE: The entry labelled 01 in the table is only included if you added the line to the table using the syntax in the first example. If you only worked through the second example, the line labelled 02 will be labelled 01, and will be the only entry in the table.

See also the **MMUL** command.

Memory Management Unit — List (MMUL)

This command displays the **MMU** table, showing any entries that are included in it.

Syntax

```
MMUL
```

No parameters are required for this command.

Example

This example displays the **MMU** table:

```
>BKM>mmul
Index |   Logical     |   Physical    |   Mask
------|---------------|---------------|-------------
01    |  0x0c000000   |  0x00000000   |  0xff000000
02    |  0x0d000000   |  0x00000100   |  0xff000000
>BKM>
```

See also the **MMUA** command.

Memory Management Unit — Delete (MMUD)

This command deletes an entry in the **MMU** table.

Syntax

MMUD *IndexNumber*

IndexNumber — This is the index number of the line that you want to delete from your **MMU** table. The index number is visible in the first column of the **MMU** table.

Entering **MMUD** without specifying an index number will delete the entire table.

Example

The following example displays the **MMU** table with two entries in it, deletes the second entry, and displays the **MMU** table again, this time with only one entry.

```
>BKM>mmul
Index |   Logical     |   Physical    |   Mask
------|---------------|---------------|-------------
01    |  0x0c000000   |  0x00000000   |  0xff000000
02    |  0x0d000000   |  0x00000100   |  0xff000000
>BKM>mmud 2
>BKM>mmul
Index |   Logical     |   Physical    |   Mask
------|---------------|---------------|-------------
01    |  0x0c000000   |  0x00000000   |  0xff000000
```

See also the **MMUL** command and the **MMUA** command.

3.1.41 Linux Virtual Memory Management (MMUOS)

The **MMUOS** commands allow the user to define and manage a set of addresses/offsets (symbols) to the Linux Page Directories, which the MMU engine of the emulator firmware (and associated plugin) uses to acquire the necessary information for Linux Virtual Memory Management emulation.

There are several **MMUOS** commands with various functions, as described below.

NOTE: To use the **MMUOS** commands, you must have the **CF MMU** option set to **ENABLE**.

MMUOS ADD

This command adds an **MMUOS** entry to the list.

Currently used **TYPE** defines are:

String: define the **MMUOS** entry as a string.

Constant: define the **MMUOS** entry as a constant hexadecimal based value.

Syntax:

```
MMUOS ADD
```

MMUOS DELETE

This command deletes an entry from the **MMUOS** list.

Syntax:

MMUOS DELETE *Name*

Name is the entry you want to delete. If you do not specify an entry, this command will delete the entire **MMUOS** list.

MMUOS DISPLAY

This command displays the **MMUOS** list.

Syntax:

```
MMUOS DISPLAY
```

Example:

```
NAME | TYPE | VALUE
-------------------------------------------
pidhash | Constant | 0xc01ab094
swapper_pg_dir | Constant | 0xc01ab094
kernelsp | Constant | 0x00000000
TASK_PId_Offset | Constant | 0x00000080
TASK_THREAD_Offset | Constant | 0x00000270
TASK_HNEXT_Offset | Constant | 0x000000b4
THREAD_PGDIR_Offset | Constant | 0x0000000c
HOOK_PageAlloc | Constant | 0xffffffff
```

MMUOS SET

This command sets the value of an entry on the **MMUOS** list.

Syntax:

MMUOS SET *Entry Value*

Entry is the entry you want to change.

Value is the value you want to change the entry to.

3.1.42 Performance Analysis (PF)

The **PF** command lets you specify performance analysis parameters (code profiling start and end range, graph type, display update mode, sample period and sample rate, and graphics on/off mode). These parameters are stored in non-volatile memory. When you start the PFA graphic display, it defaults to the settings last specified with the **PF** command, or last set inside the PFA display using the function keys.

Appending the **RUN** argument to the **PF** command causes the PFA profiling to start from a running system. You must have a **> RUN >** prompt in order to use the **PF RUN** command. Use the **GOP** command if you want to start performance profiling from a stopped state (**>BKM>** prompt).

Syntax

```
PF RUN
```

The optional **RUN** keyword starts performance profiling and opens the PFA display for a target system currently running in Real-Time (**>RUN>** prompt) with no breakpoints installed.

If the **RUN** keyword is not included, the emulator prompts you for a set of performance analysis parameters that are the defaults every time you start PFA with the **PF RUN** or **GOP** commands.

Examples

Table 3-4 **PF Examples**

>RUN>PF RUN	Starts profiling from a running system and starts the PFA histogram and chart display.
>BKM>PF	Prompts you to input the start-up performance analysis parameters as shown below.

```
1.....Performance Range       =       000A0000-000DFFFF
2.....Graph Type              =       sample last period/manual screen update
3.....Period                  =       1 Second
4.....Number of Samples/Sec   =       2K/sec
5.....Graphic Display         =       ON
6.....RUN
enter number to edit or 6 to start >
```

NOTE: Disable active breakpoints and tracing before using the **PF RUN** command. Use the **BD** (Breakpoint Disable) or **RB** (Remove Breakpoints) and **DT** (Disable Trace) commands.

NOTE: The maximum performance analysis range is 256K; only symbols that are mapped into the specified range will be shown in the PFA display. Use the PFA range to filter out data symbols.

NOTE: You can profile your code without using the graphics display. If you execute a **PF RUN** or **GOP** command with graphics off, the emulator issues a **> PFA >** prompt, signaling that the unit is sampling your code. From the **> PFA >** prompt, you may use the **DIP** command to disassemble code with profiling data. Additionally, you may issue a **DM**, **SM**, **DR**, **SR**, or **DI** command in the form of a special target snapshot from the **>PFA>** prompt. You may not issue a trace, breakpoint, or configuration command.

NOTE: Enter **Ctrl+C** from the **>PFA>** prompt to discontinue profiling, stop the target, and return to a **>BKM>** prompt.

NOTE: Choosing Option 6, **RUN**, is equivalent to using the **PF RUN** command.

See also the **GOP** (Start Performance Analysis) command, and the **DIP** (Disassemble with Code Coverage) command for more information.

3.1.43 Project Upload (PJ UPLOAD)

The **PJ UPLOAD** command displays all of the project settings that are currently loaded on your target board.

Syntax

```
PJ UPLOAD
```

Example

Output for the **PJ UPLOAD** command is several pages long, so only the first few lines of the command are shown here.

```
>BKM>PJ UPLOAD
REM ******************************************
REM CF CONFIGURATION
REM ******************************************
CF TAR                              8260    ;  OPERATION
CF SB                               SB      ;  OPERATION
CF VECTOR                           HIGH    ;  OPERATION
................................................contd.
```

3.1.44 Remove Software Breakpoint (RB)

The **RB** command deletes a software breakpoint from a specified address. If no arguments are specified, then all breakpoints are removed.

Syntax

RB *addr*

addr — This is the address of the breakpoint to remove.

If no *addr* is specified, then all breakpoints are removed.

Example

In this example, some previously set up breakpoints are displayed. Then the breakpoint at address 10000 is removed. Finally, the removal of the breakpoint is verified. Notice that the second code breakpoint becomes the first breakpoint after the **RB** command is issued.

```
>BKM>db
Software Code Breakpoints
1.  00010000  count = 0001 actual = 0000 enabled
2.  00010010  count = 0001 actual = 0000 enabled
!INFO! - [msg82001] No internal hardware breakpoints installed
>BKM>RB 10000
>BKM>db
Software Code Breakpoints
1.  00010010  count = 0001 actual = 0000 enabled
!INFO! - [msg82001] No internal hardware breakpoints installed
>BKM>
```

See also the **DB** (Display Breakpoints) command.

3.1.45 Initialize Communications with Multiple Processors (RST or RSTIN)

(Wind River ICE SX only)

This command is for use during multi-core debugging. If you are debugging multiple processors on your scan chain, you can only use the **IN** command to initialize one processor at a time. The **RST** or **RSTIN** command will initialize all of the processors included on the scan chain simultaneously. The **RST** command attempts to initialize Background Mode communications for each processor on the scan chain. Once proper communications have been established, Wind River ICE SX transfers the Chip Select Table (which can be modified with the **CS** command) and the stored register settings (which can be modified with the **SC** command) to the target system via Background Mode. To initialize Background Mode communications for multiple processors without writing the Chip Select Table and register settings, use the **RSTINN** command.

Syntax

```
RST
```

or

```
RSTIN
```

See the **IN** command for more information.

3.1.46 Only Initialize Communications with Multiple Processors (RSTINN)

(Wind River ICE SX only)

This command is for use during multi-core debugging. If you are debugging multiple processors on your scan chain, you can only use the **INN** command to put one processor at a time into background mode. The **RSTINN** command will put all of the processors included on the scan chain into background mode simultaneously. The **RSTINN** command is similar to the **RST** or **RSTIN** commands in that it places all of the processors on the scan chain in background mode. The difference is that the **RSTINN** command does not download any of the register settings or the chip-select settings that are stored in the Wind River ICE SX unit. The **RSTINN** command only places the target in background mode. If you use this command, your target settings and the settings stored in Wind River ICE SX may not be the same. Make sure that you are aware of this issue and try to keep your target settings and Wind River ICE SX settings in sync at all times.

Syntax

```
RSTINN
```

See also the **RST** command and the **INN** command.

3.1.47 Initialize and Trap Exceptions for Multiple Processors (RSTINE)

(Wind River ICE SX only)

This command is for use during multi-core debugging. The **RSTINE** command is similar to the **INE** command, a special initialization command that allows Wind River ICE SX to trap and decode exceptions that are not properly handled by the application code. The difference is that the **RSTINE** command performs the **INE** command on all of the processors in your scan chain simultaneously.

You can only use this command from a **>BKM>** prompt or an **>ERR>** prompt. The **RSTINE** command cannot be used as a target snapshot.

The **RSTINE** command performs the standard Background Mode initialization sequence (see the **RST** command for more information), including chip-select and **SC** register download. If valid Background Mode communications are established, the **RSTINE** command also initializes all exception vectors, not including the restart vector at the first eight longs, by writing a general exception address in each location in the exception vector table.

This address always points to the address immediately following the vector table. At this address,Wind River ICE SX sets a software breakpoint by inserting a **BGND** (Background Mode) instruction.

After initializing with the **RSTINE** command and issuing a **GO** command, if you hit a breakpoint, Wind River ICE SX will determine if the breakpoint was at the location pointed to by the general exception. This is the address at the end of the event to which all uninitialized vectors point. If the target has broken at this location, Wind River ICE SX will upload the exception stack frame from the target and decode it for you.

If you have taken the general exception and stopped at a breakpoint at the end of the event, Wind River ICE SX will display:

- A !BREAK! - Exception Trap Handler message
- The location in memory of the exception stack frame
- The Program Counter where the exception took place and/or the faulted address, if they are not the same
- The exception type and vector offset
- The target system's registers at the time of the exception

The **RSTINE** command assumes that the vector table is located at 0, unless you include a non-zero VBR as an optional parameter. If your vector table is in ROM, you can use the **RSTINE** feature by giving it an arbitrary RAM-based VBR, and then manually changing the VBR using the **Set Register** command (**SR VBR***RAM_address*).

You only need to use the **RSTINE** command once after system power-up because the installed vectors will remain loaded.

Syntax

```
RSTINE VBR_addr
```

VBR_addr = this is an optional parameter that specifies a non-zero VBR.

3.1.48 Set Breakpoint (SB)

The **SB** command is used to set software breakpoints. These breakpoints can be standard software breakpoints or conditional breakpoints. Set a breakpoint by specifying only the address of the breakpoint, or add a second parameter count so that the break occurs only when the specified address is encountered count times.

The following is a list of the conditions that may be checked using the conditional **SB** command:

- Check an address or data register for a specific value
- Check a memory location for a particular value

When entering the Execution Mode, the mode of operation message displayed by the **GO** command only indicates the mode that was started. If a conditional breakpoint is encountered during Execution Mode, the following message is returned:

`!BREAK! - Breakpoint at` *location*

Syntax

For setting a standard breakpoint:

SB *addr*

addr — the address where the breakpoint is set.

For setting a conditional breakpoint and checking a register value:

SB *addr count* **IF** (**R#** == *value*)

addr — This is the address where the breakpoint is set.

count — This is an optional parameter, and it causes a break only when the specified address has been encountered *count* times.

R# — This is number of the register to be checked, such as R9.

value — This is the value that the data in the register must be equal to for the breakpoint to be hit.

For setting a conditional breakpoint and checking a memory location:

SB *addr count* **IF** (**data* == *value*)

addr — This is the address where the breakpoint is set.

count — This is an optional parameter, and it causes a break only when the specified address is encountered count times.

**data* — This is the address of the data to be checked.

value — This is the value that the data at the specified address must be equal to for the breakpoint to be hit.

For setting a conditional breakpoint to enable/disable tracing:

SB *addr count* > **TE**

addr — This is the address at which the breakpoint is set.

count — This is an optional parameter, and it causes a break only when the specified address has been encountered count times.

Example

In this example, set a conditional breakpoint to trace a specific subroutine execution. First, set a conditional breakpoint that enables tracing during the subroutine. Also set a conditional breakpoint that disables tracing at the end of that subroutine. Finally, issue the **GO** command. Begin by displaying the code that will execute. Notice that when entering Execution Mode, the code will be running in real-time. Only during the subroutine will the code not run in real-time.

```
BKM>DI 6000 5
         00060000 7003      moveq.l    #3,D1
         00060002 7201      moveq.l    #1.D1
         00060004 D4006FF8  bsr.w      $7000
         00060008 E289      move.l     D0,D1
         0006000A D401      add.b      D1,D2

BKM>DI 7000 4
         0007000 D081       add.l      D1,D0
         0007002 E289       move.l     D0,D1
         0007004 D401       add.b      D1,D2
         0007006 4E75       rts
BKM>SB 7000 > TE
BKM>SB 7006 > TD
BKM>SB 600A
BKM>GO 6000
RUN>
!BREAK! - Breakpoint at 600A
BKM>DTB
         4.       0007000 D081    add.l    D1,D0
         3.       0007002 E289    move.l   D0,D1
         2.       0007004 D401    add.b    D1,D2
         1.       0007006 4E75    rts
BKM>
```

See also the **GO** command, **DI** (Disassemble) command, and **DT** (Display Trace Buffer) command.

3.1.49 Set Breakpoint —Temporary (SBT)

The **SBT** command sets a temporary breakpoint in your code. Once the breakpoint is encountered the first time, it disappears. In terms of syntax and usage, the **SBT** command can be used in exactly the same way as the **SB** command. See the **SB** command for more information about the syntax and use of this command.

3.1.50 System Configuration (SC)

Use the **SC** command to view and modify all of the internal registers in your system. The **SC** command operates on a host data file which retains register values, and it works with the **SC GRP** command in that the **SC GRP** command is needed to enable/disable register groups (including chip selects and register settings).

If there are multiple processors included in your scan chain, the **SC** command will apply to the first processor included in your scan chain. The System Configuration settings for the second processor can be used by replacing **SC** with **SC[1]**, and the settings for the third processor can be used by replacing **SC** with **SC[2]**. In the sections that follow describing all of the **SC** commands, make the replacement to **SC[1]** or **SC[2]** so that you can access the correct processor on your scan chain.

The following table lists the many features and options supported by the **SC** command.

Table 3-5 **Features and Options Supported by the SC Command**

SC	Lists all the values stored in the host data file that belong to currently active groups.
SC *reg_name*	Prompts the user for input by first displaying the location and value of the specified register.
SC *reg_name value*	Allows the user to specify a value for a register without being prompted for the input.
SC UPLOAD	Uploads all of the current settings that are stored in the emulator to the host. This includes values for all registers, both the enabled and the disabled groups.
SC ASM	This command generates the assembly code for the enabled groups that would be required to initialize the registers during run time. This code may be captured by the host, assembled, and linked in with the rest of the system boot code.
SC DEFAULT	Resets all of the emulator settings in the host data file to the default factory settings.

The **SC** command allows users to display the current register settings and target settings stored in the emulator. To display the current data file settings at the prompt, type **SC**. The emulator displays the internal registers, which include the name of the register, its target address, and the value stored in the emulator. These values are displayed by group, and only those groups that are enabled using the **CF GRP** command are displayed.

Syntax

SC

Example

The following example displays the configuration settings for an 8260 target.

```
>BKM>sc
****************** SIU              *****************
IMMR             0F0101A8  0F000000      SIUMCR           0F010000  0E240000
SYPCR            0F010004  FFFFFFC3      SWSR             0F01000E  0000
BCR              0F010024  00000000      PPC_ACR          0F010028  02
PPC_ALRH         0F01002C  01234567      PPC_ALRL         0F010030  89ABCDEF
LCL_ACR          0F010034  02            LCL_ALRH         0F010038  01234567
LCL_ALRL         0F01003C  89ABCDEF      TESCR1           0F010040  80020000
TESCR2           0F010044  00000000      LTESCR1          0F010048  00000000
LTESCR2          0F01004C  00000000      PDTEA            0F010050  00000000
PDTEM            0F010054  00            LDTEA            0F010058  00000000
LDTEM            0F01005C  00
------ CR for more information ------
****************** MEMC             *****************
OR0              0F010104  FE000856      BR0              0F010100  FE000801
OR1              0F01010C  FF000010      BR1              0F010108  FC001801
OR2              0F010114  FF000C80      BR2              0F010110  00000041
OR3              0F01011C  FF000C80      BR3              0F010118  01000041
OR4              0F010124  FFC01480      BR4              0F010120  04001861
OR5              0F01012C  FFFF03F6      BR5              0F010128  22000801
OR6              0F010134  FE000856      BR6              0F010130  E0001801
OR7              0F01013C  FFFF03F6      BR7              0F010138  21000801
OR8              0F010144  00000000      BR8              0F010140  00000000
OR9              0F01014C  FFFF0000      BR9              0F010148  60000081
OR10             0F010154  FFFF0000      BR10             0F010150  700000A1
OR11             0F01015C  FFFF0000      BR11             0F010158  800000C1
MAR              0F010168  00000200      MAMR             0F010170  00000000
MBMR             0F010174  00000000      MCMR             0F010178  00000000
MPTPR            0F010184  3200          MDR              0F010188  00000000
PSDMR            0F010190  418F48BA      LSDMR            0F010194  00000000
PURT             0F010198  08            PSRT             0F01019C  0E
LURT             0F0101A0  00            LSRT             0F0101A4  00
PCIBR0           0F0101AC  00000000      PCIBR1           0F0101B0  00000000
PCIMSK0          0F0101C4  00000000      PCIMSK1          0F0101C8  00000000
------ CR for more information ------
****************** ESTSDRAM         *****************
SDRMR            0F010C94  0000          SDPSRT           0F01019C  0E
PSDMR1           0F010190  016EB452      PSDMR2           0F010190  016EB452
```

```
PSDMR3           0F010190  296EB452      MEM1             00000000  FF
PSDMR4           0F010190  296EB452      PSDMR5           0F010190  096EB452
MEM2             00000000  FF            MEM3             00000001  FF
MEM4             00000002  FF            MEM5             00000003  FF
MEM6             00000004  FF            MEM7             00000005  FF
MEM8             00000006  FF            MEM9             00000007  FF
PSDMR6           0F010190  096EB452      PSDMR7           0F010190  196EB452
MEM10            00000000  FF            PSDMR8           0F010190  596EB452
PSDMR9           0F010190  418F48BA
------ CR for more information ------
****************** ESTLSDRAM        *****************
SDLSRT           0F0101A4  0E            LSDMR1           0F010194  2886A552
MEM11            04000000  FF            LSDMR2           0F010194  0886A552
MEM12            04000000  FF            MEM13            04000001  FF
MEM14            04000002  FF            MEM15            04000003  FF
MEM16            04000004  FF            MEM17            04000005  FF
MEM18            04000006  FF            MEM19            04000007  FF
LSDMR3           0F010194  1886A552      MEM20            04000000  FF
LSDMR4          0F010194  4086A552
>BKM>
```

Modifying Values with the SC Command

Any register can be modified by typing the **SC** command followed by the name of the register you want to modify. To set a value in Immediate Mode, the register name followed by a value will load the value into the host data file and then the emulator. By specifying the register name without a value, the emulator will prompt you for the new value by displaying the current settings and asking for input. You may also specify the form with which to input data. The default data input format is hex, but if you include a **/B** at the end of the command line, the input can be in binary format.

Syntax

SC *regname* **/b**

regname — the name of the register to be modified

/b — this is an optional parameter. If you do not include it, the data you enter for a register is in hex format. If you do include it, the data you enter for the register is in binary.

Example

To modify the module configuration register SYPCR in binary form, type:

```
>BKM>SC SYPCR /B
11111111111000011000000000000000
SYPCR 0F010004 FFFFFFC3>
```

The **>** positions the cursor just over the first bit of the SYPCR, and you can specify each bit by typing a **1** or a **0**, or by pressing the spacebar. The spacebar indicates that you do not want to change the current setting of that bit. You have the option of modifying all of the bits, or changing only one or two bits by using the spacebar.

Uploading the Emulator Settings

To upload the settings that are stored in the emulator to your host computer, use the **SC UPLOAD** command.

Syntax

```
SC UPLOAD
```

Generating Assembly Code using the SC Command

The **SC** command can be used to turn all of your system configuration settings into assembly code. The assembly code that is generated may need to be edited to suit your specific programming needs, but the **SC ASM** command provides a starting point for generating assembly code that can be inserted into the code that you are using to bring up your board, such as a Board Support Package (BSP).

Syntax

```
SC ASM
```

Example

The following example shows how the **SC ASM** command can be used.

```
>BKM>sc asm
lis    r4,0x0F00
ori    r4,r4,0x0000       # r4 is the IMMR Base Address
lis    r12,0x0000         # r12 is the SDRAM Base Address
lis    r3,0x0E24          # SIUMCR
ori    r3,r3,0x0000
lis    r7,0x0001
ori    r7,r7,0x0000
stwx   r3,r7,r4
lis    r3,0xFFFF          # SYPCR
ori    r3,r3,0xFFC3
lis    r7,0x0001
ori    r7,r7,0x0004
stwx   r3,r7,r4
li     r3,0x0000          # SWSR
....
```

Restoring the Default Configuration Settings (SC DEFAULT)

The **SC DEFAULT** command restores the default configuration settings for your processor.

Syntax

```
SC DEFAULT
```

⚠ **CAUTION:** This command causes any register settings or configuration options on your target to be overwritten by the emulator. Make sure that you save these settings to a file before issuing the **SC DEFAULT** command if you wish to preserve them.

3.1.51 System Configuration Add/Delete (SCA/SCD)

The **SCA** command is used to add custom registers to the existing set of system configuration registers. Custom registers can be any memory mapped peripheral register. Custom registers will be initialized with the **IN** command if the custom register group is enabled (**CF GRP** command). Custom registers are initialized in the sequence that they are originally created with the **SCA** command. The size (byte, word, or long word) of the register is determined by the number bytes in the value argument. Custom registers can be saved in the emulator with the **SC SAVE** command.

Syntax

SCA *name_addr_value*

name = The name of the register.

addr = The address of the register.

value = The initial value of the register.

Examples

Create a custom register called IO1 at address 0x1000. Initialize this byte size register with a value of 0xde.

```
SCA IO1 1000 DE
```

Create a custom register called IO2 at address 0x1010. Initialize this word size register with a value of 0x0.

```
SCA IO2 1010 0000
```

Create a custom register called IO3 at address 0x1020. Initialize this long word size register with a value of 0x12345678.

```
SCA IO3 1020 12345678
```

Save these registers into the emulator NVRAM.

```
SC SAVE
```

Deleting Custom Registers (SCD)

The **SCD** command is used to delete custom registers.

Syntax

SCD *name*

name = The name of the register.

Example

Delete custom register IO3.

```
SCD IO3
```

Use the command **SC UPLOAD CUSTOM** to view the newly created registers.

3.1.52 System Configuration —Target (SCT)

The **SCT** command behaves similarly to the **SC** command, except that it affects only the system configuration settings on the target. It allows you to view and modify all of the internal registers on the target.

If you use the **SCT** command to configure your target, be aware that any time you issue an **IN** command or power cycle the emulator, all of your target values get overwritten by the information that is stored in the emulator. For that reason it is best to make sure that your target settings and the settings stored in the emulator are the same whenever possible.

The following table lists the many features and options supported by the **SCT** command.

Table 3-6 **Features and Options Supported by the SCT Command**

SCT	Lists all the current target settings for the enabled groups. Any value displayed that differs from the value stored in the emulator is highlighted with an asterisk.
SCT COPY	Copies the values that are stored on the target into the emulator.
SCT *regname*	Prompts the user for input by first displaying the current target settings. Entering a value changes the value on the target but not in the emulator.
SCT *regname value*	Sets the target register to the value specified.
SCT DIFF	This command displays the differences between the current target values and the values stored in the host data file.
SCT UPLOAD	Uploads the current target values to the host.
SCT ASM	Generates assembly code for the target settings.

Most of the **SCT** commands work similarly to the SC commands described in *3.1.50 System Configuration (SC)*, p.67. The only difference is that with the **SCT** commands the information is being taken from or added to the target rather than the emulator. See the **SC** commands for syntax.

The **SCT DIFF** command is different from the other commands described in the **SC** section. The **SCT DIFF** command displays both the values stored in the emulator data file and the target values that differ.

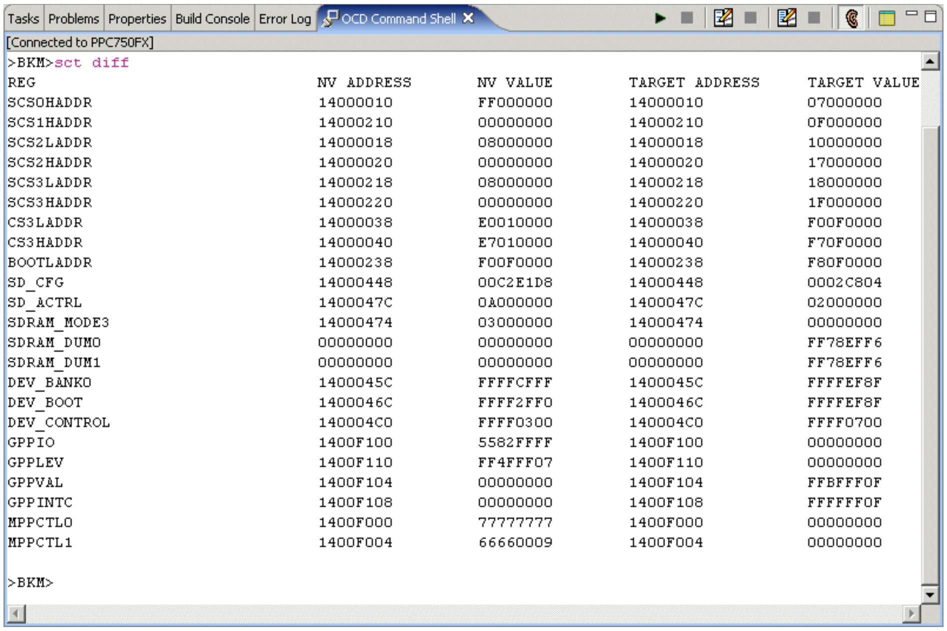
Syntax

```
SCT DIFF
```

Example

The following example shows output from the **SCT DIFF** command.

Figure 3-5 **SCT DIFF Command**



```
[Connected to PPC750FX]
>BKM>sct diff
REG                                NV ADDRESS       NV VALUE        TARGET ADDRESS      TARGET VALUE
SCS0HADDR                          14000010         FF000000        14000010            07000000
SCS1HADDR                          14000210         00000000        14000210            0F000000
SCS2LADDR                          14000018         08000000        14000018            10000000
SCS2HADDR                          14000020         00000000        14000020            17000000
SCS3LADDR                          14000218         08000000        14000218            18000000
SCS3HADDR                          14000220         00000000        14000220            1F000000
CS3LADDR                           14000038         E0010000        14000038            F00F0000
CS3HADDR                           14000040         E7010000        14000040            F70F0000
BOOTLADDR                          14000238         F00F0000        14000238            F80F0000
SD_CFG                             14000448         00C2E1D8        14000448            0002C804
SD_ACTRL                           1400047C         0A000000        1400047C            02000000
SDRAM_MODE3                        14000474         03000000        14000474            00000000
SDRAM_DUM0                         00000000         00000000        00000000            FF78EFF6
SDRAM_DUM1                         00000000         00000000        00000000            FF78EFF6
DEV_BANK0                          1400045C         FFFFCFFF        1400045C            FFFFEF8F
DEV_BOOT                           1400046C         FFFF2FF0        1400046C            FFFFEF8F
DEV_CONTROL                        140004C0         FFFF0300        140004C0            FFFF0700
GPPIO                              1400F100         5582FFFF        1400F100            00000000
GPPLEV                             1400F110         FF4FFF07        1400F110            00000000
GPPVAL                             1400F104         00000000        1400F104            FFBFFF0F
GPPINTC                            1400F108         00000000        1400F108            FFFFFF0F
MPPCTL0                            1400F000         77777777        1400F000            00000000
MPPCTL1                            1400F004         66660009        1400F004            00000000

>BKM>
```

3.1.53 System Configuration Group Add (SCGA)

This command is used to create registers within a specified register group. The register group may be one that already exists, in which case the new register is just added to it, or if the register group does not exist it is created with the new register in it. Up to 32 custom register groups can be created, a total of 960 custom registers.

Syntax

SCGA *GroupName RegisterName Address Data Options*

GroupName — This is the name of the register group that the new register is added to.

RegisterName — This is the name of the register that you are creating.

Address — This is the address where the new register is located.

Data — This is the data that is stored in the register you are creating.

Options — There are many options associated with the **SCGA** command. Table 3-7 describes them.

Table 3-7 **SCGA Options**

Option Name	Description
/cpur	This option specifies that the register you are creating is a CPU core register (that is, **SPR**, or other non-memory mapped register.)
/hide	This option means that the register is not visible when an **SC** or **DR** command is executed. It is only visible when an **SC/SCG UPLOAD** command is issued.
/lendian	This option signifies that the register you are creating is a little-endian register. It will only work on targets that are able to switch between little-endian and big-endian modes.
/memr	This option signifies that the register is a memory mapped register. It is the default for self-defined registers.
/no_in	This option means that the register you are creating does not get set on the target during an **IN** sequence.
/r, /rw	These options specify Read Only and Read/Write registers. The default is **/rw**.
/Sz:B, /Sz:W, /Sz:L, /Sz:D	These options force the size of the register to either Byte (8 bits), Word (16 bits), Long (32 bits), or Double (64 bits). The default register size is determined by the amount of characters used to specify the default value.
/va_dr	This option is used on anchor registers to make them available on a **DR** command.
/w, /rw	These options are Write and Read/Write flags. The default is **/rw.**
/wo	This option defines a fixed value register. That register is not affected by an **SCT COPY** command.
/w(nwf)	This option specifies a write cycle (next write first.) It indicates that to write a value to the register, you need to write the following register value to the target first.
/r(nwf)	This option specifies a read cycle (next write first.) It indicates that to read a register, you need to write the following register value to the target first.

Table 3-7 **SCGA Options**

Option Name	Description
/r(nwa)	This option specifies a write cycle (next write after.) It indicates that to write a value to this register, you need to write the next register value to the target afterwards.

Example

The first example creates a new group called **SIM_MMU**, with a core register **SIM_IBATOL** included in it.

```
>BKM>SCGA SIM_MMU SIM_IBATOL 4014 00000004 /cpur
>BKM>
```

The second example describes the **/w(nwf)** option. In this case, a register called **PCICMD** is created in register group **MPC_PCI**, however the option specifies that the register cannot be written to unless a write to the register **ADDR_04**, located in the same register group, is performed first.

```
SCGA MPC_PCI PCICMD 80000CFC 0600 /w(nwf) /r(nwf)
SCGA MPC_PCI ADDR_04 80000CF8 04000080 /wo /hide
```

See also the **SCGD** command.

3.1.54 System Configuration Group Delete (SCGD)

This command allows you to delete specific registers in a group or all registers in a group.

Syntax

There are two ways to use this command. The first way causes all of the registers included in a register group to be deleted. The register group will still be available.

SCGD *Groupname*

Groupname — This is the name of the register group that the registers are to be deleted from.

This second way to use the **SCGD** command is to specify a register within a register group to delete. In that case, only the specified register is deleted.

SCGD *Groupname Regname*

Groupname — This is the name of the group where the register to delete is located.

Regname — This is the name of the register to delete from register group *Groupname*.

Example

```
>BKM> SCGD SIM_MMU SIM_IBATOL
>BKM>
```

See also the **SCGA** command.

3.1.55 Synchronize Cores (SCTRL)

Syntax

SCTRL [*designator* | **ALL**] [**ENABLE** | **DISABLE**] [**ALL** | **GO** | **HALT** | **TRIGGER**]

This command allows you to configure a group of cores for synchronized operation. This is a multi-core command and is only supported for the Wind River ICE SX.

designator is the identification of the core you want to use, as specified in the board descriptor file. This option can also be set to **ALL**, meaning that all Microprocessor type devices being debugged at this time will be synchronized. For information on board descriptor files, see the *Wind River Workbench On-Chip Debugging Guide: Board Descriptor Files*.

Specify **ENABLE** to enable synchronized activity; specify **DISABLE** to disable it.

Specify **GO** to synchronize cores only on **GO** operations.

Specify **HALT** to synchronize cores only on **HALT** operations.

Specify **ALL** to synchronize cores on both **GO** and **HALT** operations.

The **TRIGGER** option is not currently used.

Example

```
SCTRL ALL ENABLE ALL
```

This command groups all Microprocessor-type devices on the scan chain for all synchronized actions.

See also the **GOS** and **HALTS** commands.

3.1.56 Set Verbose On (SET VERBOSE ON)

The **SET VERBOSE ON** command puts the emulator in verbose mode, which is useful when diagnosing communication problems with your target. When the emulator is not able to place your target in Background Mode after initialization, a failure message is displayed:

```
Initializing Background Debug Mode...... Failed
```

Additional information about the failure may be obtained by using the verbose mode at the **> ERR >** prompt. When in verbose mode, a number of pass/fail diagnostic messages are displayed as the emulator attempts to place the target in Background Mode. Using verbose mode will help you to diagnose why your target cannot be placed in Background Mode.

NOTE: The tests described may vary depending on your target architecture.

Initialization Tests for JTAG Targets

Using JTAG communications, the following output occurs when a **SET VERBOSE ON** command is issued followed by an **IN** command:

```
>BKM>set verbose on
>BKM>in
*****************************************************************************
Wind River ICE Initialization Sequence.
Copyright (c) Wind River Systems, Inc., 1999-2004. All rights reserved.
*****************************************************************************
Support Expires.......5/17/07
Target Processor...... MPC8260
Wind River ICE  Group ID#=0
Wind River ICE  Serial#=W0000000    Firmware= vp2.3a
Type CF For a Menu of Configuration Options
Testing Communications to Hardware Interface......Passed
Checking Paddle Board.............................Passed
Driving HRESET to be High.........................Passed
Driving HRESET to be Low..........................Passed
Waiting HRESET Low Acknowledge.................Passed
Attempting JTAG communication.....................Passed
Waiting for HRESET to be released.................Passed
Testing for target STOP State.....................Passed
Comparing target CPU with CF setting..............Passed
Waiting for HRESET High Acknowledge.............Passed
Checking IR length.............................Passed
Testing JTAG Communication........................Passed
Attempting to Locate IMMR register................Passed
Loading Internal Registers........................Passed
Testing JTAG Communication........................Passed
Attempting to restore CPU context.................Passed
>BKM>
```

The following is a brief description of the tests with some possible reasons why each test would fail.

Testing Communications to Hardware Interface

This tests the hardware connectivity, and examines the communications path between the host and the emulator. If the test fails, ensure that you have the power properly connected and turned on, that the emulator is correctly connected to the host computer, and that your emulator hardware is properly connected to the target.

Checking Paddle Board

If this test fails, verify that the paddle board is correctly connected to the emulator unit. If it fails, it is likely that there is a problem with this paddle board. Verify that the connection between the paddle board and the unit is secure, and that all of the pins are intact.

Driving HRESET to be High

This function tests the RESET signal of the JTAG connector to verify that it is high. The emulator is not driving the RESET signal during this test, so the target must drive the RESET signal via a pull-up resistor. If this test fails, check to see if the target board has a pull-up resistor on the RESET signal to the HRESET pin of the JTAG connector. Also, check the target board reset logic and verify that it is not continually driving RESET low.

Driving HRESET to be Low

The RESET signal is a bi-directional signal for your unit. The emulator drives the RESET signal low and clocks it back in to verify that it is low. If this test fails, you may have contention on your reset signal. Check to see if a device on your target board is continually driving reset high. Verify that the device on your target board that is driving the RESET signal is an open collector device with a pull-up resistor.

Attempting JTAG communication

During this test, the emulator stops the processor and attempts to establish JTAG communications. If this fails, check to see that your hardware is connected properly, and that the tests preceding this one passed accordingly. It is also possible that there is contention on your board.

Waiting for HRESET to be released

The emulator only drives RESET low for a specified period of time. After RESET is driven low for the allotted time, it tri-states the RESET driver and clocks the RESET signal back in to see if the RESET signal went high. It continues to check for RESET to go high until is sees it go high or until you type **Ctrl+X**. If this test fails, check to see if your target board reset logic is still driving the RESET signal low. Also check that your target board has a pull-up resistor to drive RESET high.

Testing for target STOP State

This test verifies that the processor stopped during the preceding JTAG Communications test by polling the processor status. If the target is still running, this test fails.

Comparing Target CPU With CF Setting

This test verifies that you are properly configured for the appropriate target processor by comparing the processor type on your target with the processor type specified in your board file. If the test fails, use the **CF TAR** command to properly configure your target. For example, if this test fails and you are using an MPC8260 board, execute a **CF TAR 8260** command sequence in the **OCD Command Shell**.

Testing JTAG Communication

This tests the JTAG communication using a slow internal clock rate.

Attempting to Locate IMMR register

This test only completes for PowerPC 82xx targets. It attempts to verify the location of the IMMR register, which serves as a pointer to all of the other registers. If it fails, none of the internal registers are accessible. If the test fails, check the reset configuration word, located in the Flash, and ensure that it is set to the correct value.

Loading Internal Registers

Once background communications are established, the emulator downloads register values from the debugger NV-RAM to the target. It will only download register values for those register groups that are enabled. The CF GRP command can be used to view and change the groups that are enabled. If this test fails, verify that your register values are correct.

Testing JTAG Communication

This test examines the JTAG Communication between the emulator and the target using the internal clock rate for which the emulator is configured. If this test fails, use the CF CLK command to lower the internal clock rate.

Attempting to restore CPU context

This test restores the processor scan chains.

Initialization Tests for BDM Targets

Using BDM communications, the following output occurs when a **SET VERBOSE ON** command is issued followed by an **IN** command:

```
>BKM>set verbose on
>BKM>in
*****************************************************************************
Wind River ICE Initialization Sequence.
Copyright (c) Wind River Systems, Inc., 1999-2004. All rights reserved.
*****************************************************************************
Support Expires.......5/17/07
Target Processor...... MPC860
Wind River ICE   Group ID#=0
Wind River ICE   Serial#=W0000000    Firmware= vp2.3a
Type CF For a Menu of Configuration Options
Testing Wind River ICE Communication...........Passed
Driving HRESET to be High...................Passed
Driving HRESET to be Low....................Passed
Waiting for HRESET to be Released...........Passed
Testing for target FREEZE State.............Passed
Attempting to Enable Background Mode........Passed
Testing BDM Communication...................Passed
Loading Internal Registers..................Passed
Testing BDM Communication...................Passed
Initializing CPU registers..................Passed
>BKM>
```

Here is a brief description of the tests with reasons why each test might fail.

Testing Communication

This item checks the hardware connectivity, and examines the communications path between the host and the emulator. If this fails, ensure that power is properly connected and turned on, that the emulator is correctly connected to the host computer, and that your emulator hardware is properly connected to the target.

Driving HRESET to be High

This function tests the RESET signal of the BDM connector to verify that it is high. The emulator is not driving the RESET signal during this test, so the target must drive the RESET signal via a pull-up resistor. If this test fails, check to see if the target board has a pull-up resistor on the RESET signal to the HRESET pin of the BDM connector. Also, check the target board reset logic and verify that it is not continually driving reset low.

Driving HRESET to be Low

The RESET signal is a bi-directional signal for your unit. The emulator drives the RESET signal low and clocks it back in to verify that it is low. If this test fails, you may have contention on your RESET signal. Check to see if a device on your target board is continually driving RESET high. Verify that the device on your target board that is driving the RESET signal is an open collector device with a pull-up resistor.

Waiting for HRESET to be Released

The emulator only drives RESET low for a specified period of time. After the debugger has driven RESET low for the allotted time, it tri-states the RESET driver and clocks the RESET signal back in to see if the signal went high. It continues to check for RESET to go high until is sees it go high or until you type **Ctrl+X**. If this test fails, check to see if your target board reset logic is still driving the RESET signal low. Also check that your target board has a pull-up resistor to drive RESET high.

Testing for target FREEZE State

This test determines whether or not the processor is in Freeze state. The emulator monitors the VFLS signals to see if the processor is frozen. These signals are high if the processor is in the Freeze state. If the Freeze signal is being used instead of the VFLS signals, then the emulator checks that the Freeze signal is high. This test usually fails if the Hardware Reset Configuration word is incorrect. Verify that bits D9 and D10 of the Reset Configuration Word are driven high and bits D11 and D12 are driven low during the reset. Bits D9 and D10 select the VFLS functionality on the appropriate multifunction pins.

Attempting to Enable Background Mode

The emulator checks to make sure the processor is in background mode. The emulator also initializes registers in accordance with its CF parameters.

Testing BDM Communication

The emulator tests the serial communications channel by writing a test pattern to register R00 and then reading register R00 to verify that the write was correct.

Loading Internal Registers

The emulator initializes the internal target registers. This test is not performed if an **INN** command is used to initialize the target. The emulator initializes all registers in enabled register groups. Register groups can be enabled or disabled with the **CF GRP** command.

Testing BDM Communication

The emulator test the serial communication channel a second time by writing a test pattern to register R00 and then reading register R00 to verify that the write was correct. This test is repeated because the debug port may be inadvertently disabled by the previous tests. If this test fails, check to make sure that the initial value for the SUIMCR register does not disable the debug port.

Initializing CPU registers

The emulator initializes CPU registers that are loaded. If this fails, check your register values.

3.1.57 Show History (SH)

The emulator automatically logs all of its command responses and error messages in an internal buffer. The **SH** command allows you to display this text buffer, thus displaying a history of the command responses.

Essentially, any ASCII characters that the emulator transmits to a host PC, either in direct communication mode or as a response to a source-level debugger command, are logged in this buffer. The system does not log commands that were received from the host, only responses. In most cases, the emulator echoes back the received command before it processes it, so that this echo of the original command is in the history buffer.

Binary interfaces to source-level debuggers denote a special case of the **SH** command. The emulator does not log its binary response to the source-level debugger command; however, before the unit sends its binary packet to the host, it logs the ASCII equivalent response in the history buffer. This can be extremely useful in trapping target problems that may be hidden by the source debugger interface. For example, the target may take a double bus fault, but the emulator has no way to communicate this specific problem to the host debugger. The **SH** command allows you to view the output history log, which in this case lets you display the entire ASCII double bus fault message as you would have seen it had you been in direct ASCII communication mode.

The emulator uses a 2K circular text buffer to log the history. The **SH** command allows you to display the log backward (**B**) or forward (**F**) in the buffer. The default is to display the log backward. The **SH** command also takes the number of lines you wish to display as an optional parameter. If you use this parameter, the **SH** command displays that number of lines forward or backward from the current buffer pointer. The **SH** command, without the lines parameter, resets the buffer pointer to the very last command that was logged. A carriage return immediately following an **SH** command displays the next page of the buffer, either forward or backward, depending on the previous **SH** command. The **SH** command is valid from a **> BKM >**, **> ERR >**, **>RUN>** or **>TRC>** prompt.

Syntax

SH*direction number_of_lines*

direction — This determines whether you display the log forward (**F**) or backward (**B**). By default this option is set to **B**. Do not leave a space between **SH** and the *direction* parameter.

number_of_units — This specifies the number of logged ASCII lines you wish to display.

A carriage return immediately following a **SH** command repeats the **SH** command, while updating the buffer pointer either forward or backward.

Examples

Table 3-8 **SH Examples**

>BKM>SH	Resets the buffer pointer to the last command that was logged and displays a full page of the log, backward.
>BKM>ENTER	Displays the next page of the log, backwards.
>BKM>SHF 8	Displays the next eight lines in the buffer, forward.

3.1.58 Single-Step Instruction(s) (SI)

The Single-Step command steps the emulator count number of target instructions. The default count is one instruction. There are optional arguments for displaying registers and disassembling the last instruction that was just executed.

Syntax

SI*arg1arg2 count*

arg1 — This argument is the **DR** command, which displays all registers.

arg2 — This argument is the **DI** command, which disassembles the last instruction executed.

count — This is the number of target instructions to execute.

Note that the positions of *arg1* and *arg2* can be reversed with no difference in the output.

Example

In this example, single-step the target for one instruction. View the contents of the registers and the instruction that was executed.

```
BKM>SIDIDR
D0 = 00000000 D1 = 00000000 D2 = 00000000 D3 = 00000000
D4 = 00000000 D5 = 00000000 D6 = 00000000 D7 = 00000000
A0 = 00200000 A1 = 00000000 A2 = 00000000 A3 = 00000000
A4 = 00000000 A5 = 00000000 A6 = 00000000 A7 = 00000000
USP= 00000000 SSP= 00000000 PC = 00000000 SR = 0000
VBR= 00000000 SFC = 0000 DFC = 0000

MOVE.W #$7F00,(A0)
BKM>
```

3.1.59 Set Memory (SM)

This command sets memory with data in byte, word (default), or long format, starting at *addr*.

Using the syntax indicated below, the **SM** command allows both memory-relative and register-relative addressing.

The **SM** command differs from the **MM** command in that the **MM** command only modifies the memory at a specified address whereas the **SM** command reads back the value to verify that it wrote correctly.

Syntax

SM*unit base_addr data1 optional_data2 optional_dataN*

unit — This parameter can be either **B** (byte), **W** (word), or **L** (long). By default, this option is set to Word.

addr — This parameter is the target address at which to begin setting data.

data — This parameter can be one or more Hex strings of data in units.

The Set Memory command can also take an indirect address and a register-relative address as parameters. The syntax is as follows:

SM(**addr)*

Example 1

In this example, display the contents of memory both before and after setting three bytes of memory to the specified values:

```
BKM>DMB 200010 3
200010: FF 0F 00
BKM>SMB 200010 0B 7F 34
BKM>DMB 200010 3
200010: 0B 7F 34
BKM>
```

There is also an optional Task Identifier (**tid=**) field for Linux. The Task Identifier relates the address to the Linux Process Identifier. Some PowerPC targets (such as the PPC405) employ a PID register, which is not the same as the Linux **pid** and **tid**. When you need to cross reference a Linux User Space address, assuming the MMUL tables have been preset to know the PIDs for a given operating system, you may enter a relative address with the value to access the correct physical address on the target. Up to five (0,1,2...4) PID cross-referenced addresses (in the MMUL table) are allowed.

Example 2:

```
    >BKM>sml tid=1 5500 12345678
```

In this example, the TID is 1, the Logical Address is 0x5500, and the *value* is 12345678.

NOTE: Linux assigns TIDs as needed. Use the Linux command **ps –a** at a Linux shell to get the list of Process IDs for all user processes.

See also the **DM** (Display Memory) command and the **MM** (Memory Modify) command.

3.1.60 Set Memory Double (SMD)

The **SMD** command is for supported architectures. This command is similar to the **SM** command in that it sets memory with data starting at *addr*. However, it will write data with full 64-bit write operations.

Using the syntax indicated below, the **SMD** command allows both memory-relative and register-relative addressing.

Syntax

SMD *addr data*

addr — This is the base address at which to start adding data.

data — This is the data to be added.

Example

The following example first reads the memory at location 200010, then writes some data to the location, and then reads it again.

```
>BKM>dmd 200010 3
00200010: FFFFDFFFFFFFFFFF FFFFFFFFFFFFFFFF    ................
00200020: FFFFFFFFFFFFFFFD    ................
>BKM>smd 200010 0B 7F 34
>BKM>dmd 200010 3
00200010: 000000000000000B 000000000000007F    ................
00200020: 0000000000000034    .......4........
>BKM>
```

3.1.61 Set Register (SR)

The **SR** command sets the register *reg_name* to the values designated by *hex_string*. The **all** parameter is useful when clearing all of the registers.

Syntax

SR *reg_name1 data1 reg_nameX dataX*

reg_name1 — The name of the first register to have data added to it.

data1 — The data to add to *reg_name1*.

The *reg_nameX dataX* can be repeated for as many registers as you want to modify at one time.

The other syntax that can be used for this command is shown below.

Example

In this example, display registers R00 and R01. Then modify the values of those registers using the **SR** command, then display them again to verify their contents.

```
>BKM>dr r00 r01
R00      = 00000000 R01      = 00001234
>BKM>sr r00 12345678 r01 88888888
>BKM>dr r00 r01
R00      = 12345678 R01      = 88888888
>BKM>
```

See also the **DR** (Display Registers) command.

3.1.62 Search for String (SS)

The Search for String command allows you to search for a string in memory. The string may be an ASCII string or Hex data. The search begins at *strt_addr* and ends at *end_addr*. If the string exists more than once in the block of memory, pressing **ENTER** continues the search.

Syntax

SS*unit strt_addr end_addr string*

unit — This parameter can be either **H** (hex) or **A** (ASCII). By default, this option is set to ASCII.

strt_addr — This option is the starting address of the search.

end_addr — This option is the ending address of the search.

string — This is the string to search for.

Example

In this example, search for the ASCII string hello between 10000H and 12000H. Find one string and continue the search until all of the locations are checked. There is only one occurrence of this string in the memory that is searched.

```
BKM>SSA 10000 12000 hello
String found at 1000f
BKM>cr
String not found
BKM>
```

3.1.63 SY Commands

The following **SY** commands are low-level system commands used to provide detailed control and diagnostic information obtained via the JTAG scan chain information.

NOTE: Some JTAG diagnostic commands are specific for different processors, and do not necessarily support them all. Refer to command descriptions for information on which processors each command supports.

SY

For PowerPC processors, entering **SY** with no arguments lists all available **SY** commands and their syntax. These will vary by target processor. The following example shows the output displayed after an **SY** command on a PPC750FX target.

```
>BKM>sy
0. SYNC                     : Synchronize the emulator with the target.
1. REV                      : Display the PVR Register value.
2. pci mapa                 : Poll all 32 devices on the MAP_A bus.
3. pci mapb                 : Poll all 32 devices on the MAP_B bus.
4. pci rega                 : Display the MAP_A PCI bridge Registers.
5. pci regb                 : Display the MAP_B PCI bridge Registers.
6. JTAG stat                : Display all types of status for multiple devices
7. JTAG rd                  : Read the Boundary scan chain for this device
8. JTAG ID                  : Read the device ID for this device
9. JTAG sca                 : Analyze the topology of a mutiple device chain
10. JTAG wr<v><l><c><d>     : Write BSDL bit <v> to level <l> control <c>
                                 direction <d>.
11. BS hrst                 : Display Boundary Info while HRESET is asserted
12. BS                      : Display Boundary Information.

13. BKM bsdl                : Toggle the BSDL external pins.
14. BKM                     : Toggle the JTAG external pins.
15. PROMPT                  : Force the prompt to BKM mode.
16. CHAIN                   : Display the length of the SELECTED chain.
17. MAP                     : Display CS mapping.
18. PROG1 {ADDR}            : Download basic program to target at ADDR.
19. PROG2 {ADDR}            : Download read/write program to target at ADDR.
20. PROG3 {ADDR}            : Download exception program to target at ADDR.
21. CMD {-l} <v>            : Issue a COP command v = COP command, -l = loop.
22. BIST {-l} <v>           : Issue a JTAG command v = COP command, -l = loop.
23. BAT                     : Display the BAT registers formatted.
24. RSTCONF <val><size><Ftype><PSpan> : Program the PQII Conf Word.
25. REPORT                  : Display COP/JTAG -> Core Information.
26. CLAMP INIT              : Freeze the cores clocks and release HRESET
27. CLAMP                   : Freeze the cores clocks and hold HRESET
28. CONNECT                 : Synchronize the emulator with the TAP controller
29. READFPU                 : Display Floating Point Registers
30. UMM <addr> <v>          : Unaligned Modify Memory Long (32bit only).
31. FREERUN <addr>          : Run the processor without trapping exceptions.
32. HRESET LOW              : Assert and HOLD the HRESET signals.
33. HRESET HIGH             : Release the HRESET signal.
34. 107 sdram               : Initialize MPC107 for SDRAM using mapB.
35. CACHE L1                : Display L1 cache data
36. CACHE L2                : Display L2 cache data
37. CACHE L3                : Display L3 cache data
```

```
38. PQII <immr location>   : Fined the IMMR and disable the watchdog timer.
>BKM>
```

SY BAT

This command obtains the BAT registers and formats them to determine memory blocks, sizes, and configurations. It is for use with PowerPC processors.

```
Name     PageIndex   BaseAddress    Size    W I M G Access Protection
IBAT0    fff00000    fff00000       16MEG   0 1 0 0 Supr & User Read Only
IBAT1    00000000    00000000       128MEG  0 0 1 0 Supr & User Read/Write
IBAT2    fe000000    fe000000       16MEG   0 1 0 0 Supr & User Read/Write
IBAT3    00000000    00000000       128K    0 0 0 0 Disabled
DBAT0    fff00000    fff00000       16MEG   0 1 0 0 Supr & User Read Only
DBAT1    00000000    00000000       128MEG  0 1 1 0 Supr & User Read/Write
DBAT2    fe000000    fe000000       16MEG   0 1 0 0 Supr & User Read/Write
DBAT3    00000000    00000000       128K    0 0 0 0 Disabled
```

SY BS

Scans out the boundary scan chain and displays most signals and their logic state. For use with PowerPC processors. The (**s**) indicates a static signal read directly from the pin. All others are based on the last clock before the processor has stopped, as shown below.

```
Scan Address      = ffffffff
QREQ              = LOW QACK(s)         = LOW
MCP(s)            = HGH SRESET(s)       = HGH
HRESET(s)         = HGH CKSTPIN(s)      = HGH
INT (S)           = LOW AACK(s)         = HGH
ABB               = HGH DBB             = HGH
DBG(s)            = HGH DBDIS(s)        = HGH
TS                = HGH TA(s)           = HGH
TEA(s)            = HGH BR              = HGH
BG(s)             = LOW GBL             = HGH
DRTRY(s)          = HGH ARTRY           = HGH
SMI(s)            = HGH TLBSYNC         = HGH
CKSTPOUT          = LOW
PLL0..3(s)        = 4    133Mhz
TSIZ              = 0
TTx               = 1f
```

SY CHAIN

This command scans and counts each bit throughout the scan chain to determine its exact length. This is helpful to determine which processor you are working with. Some PowerPC processors have scan chains that are not supported. This will be the quickest indication that the processor and emulator are not compatible. The command can be used with PowerPC processors.

```
Chain Length in bit = 7488
```

SY CMD *value*

This command provides a way to send JTAG commands directly to the JTAG interface. The *value(s)* are any of the JTAG commands supported by the processor you are working with. For use with PowerPC processors.

```
CPU is Stopped, BIST is Complete
```

SY PCI

Displays the MPC 106 configuration registers in big endian mode. This command also displays the current map being used. For use with PowerPC processors.

```
Using MAP b........
Device and Vendor ID       = 00021057   Memory Starting Address  = ffff0800
Memory Ending Address      = ffff0f07   Processor Configure #1   = ff341c48
Processor Configure #2     = 00000000   Memory Configure #1      = ffb80003
Memory Configure #2        = 0000020c   Memory Configure #3      = 02300000
Memory Configure #4        = 25402220   Memory Bank Enable       = 03
PCI Status/Command         = 00800006   Power Management         = dd000000
PCI Error Address          = 00000000
```

System Program (SY PROG)

The **SY PROG** command is used to write a tight loop of code at a given address. This command is used to see if the processor is operational (can execute code out of internal RAM, for example). The **SY PROG1** command writes a tight loop consisting of NOPs followed by a branch back. The **SY PROG2** command writes a tight loop to load and store instructions.

This command is only supported for PowerPC microprocessors.

Syntax

SY PROG*n*

n = 1, 2

Examples

Write a NOP loop to internal RAM, located at 0x2000 offset from IMMR register (0xff000000):

```
BKM>sy prog1 ff002000
```

Write Load/Store loop to external RAM located at 0x40000:

```
BKM>sy prog2 40000
```

SY REV

This command scans out the PVR register of the processor and formats the value into the product name and revision. For use with PowerPC processors.

Example

```
>BKM>sy rev
Part PVR  Value = 0x70000202, IBM 750FX2 Revision 2.2
>BKM>
```

SYNC or SY PROMPT

When the emulator enters error mode, the JTAG interface can re-synchronize and then the scan chain can be read. In some cases, this information can help diagnose the reason for an error condition. This can be used with PowerPC processors.

Example

```
>BKM>sy prompt
Synchronization Complete. Current PC = 0xfff00100
>BKM>
```

System Map (SY MAP)

The **SY MAP** command provides the chip select start and end addresses for each chip select when using the emulator with a PowerPC 8xx target.

Syntax

```
SY MAP
```

Example

```
BKM> sy map
IMMR = FF000000 -> FF003FFF
CS0  = FFF00000 -> FFFFFFFF
CS1  = FFE00000 -> FFE7FFFF
CS2  = 30000000 -> 3000FFFF
CS3  = 04000000 -> 0407FFFF
CS4  = 00000000 -> 003FFFFF
CS5  = DISABLED..
CS6  = DISABLED..
CS7  = DISABLED..
BKM>
```

3.1.64 Trace Disable (TD)

The **TD** command disables the non-real-time software trace. Software trace is disabled by default. Note that software trace is automatically activated (and a **>TRC>** prompt appears) if you set a software breakpoint on ROM code or a software data breakpoint.

Syntax

TD

No parameters are required for this command.

3.1.65 Target Diagnostic Functions (TDF)

The target diagnostic functions are similar to the diagnostic functions (**DF** commands) described previously in this chapter. The difference is that in this case, the diagnostics are run directly on the target instead of through the emulator.

Typing **TDF LIST** at the **>BKM>** prompt displays a list of all of the available target diagnostic functions. Typing **TDF VERSION** displays the versions of each of the target diagnostic functions.

The code that runs the diagnostic will be loaded at the memory address indicated by the **CF WSPACE** configuration option. Make sure you are not trying to test the same memory area that you are specifying in **CF WSPACE**.

There are five target diagnostic function tests available.

TDF 0 — Single-pass Simple Memory Test

This diagnostic function runs a simple RAM test through a single pass only.

Syntax

TDF*unit 0 startaddress endaddress*

unit — This can be either **B** (Byte), **W** (Word), or **L** (Long).

startaddress — This is the address where the diagnostic test is to start.

endaddress — This is the address where the diagnostic test is to finish.

Example

```
>BKM>tdf 0 1000 1100
Single-pass Simple Memory Test loaded. Now executing TDF.
Test Complete
>BKM>
```

TDF 1 — Continuous Single Memory Test

The **TDF 1** command runs a simple RAM test continuously. This test can be stopped by typing **Ctrl+X**.

Syntax

TDF*unit 1 startaddress endaddress*

unit — This can be either **B** (Byte), **W** (Word), or **L** (Long).

startaddress — This is the address where the diagnostic test is to start.

endaddress — This is the address where the diagnostic test is to finish.

Example

```
>BKM>TDF 1 1000 1100
Continuous Simple Memory Test loaded. Now executing TDF.
test looping press ^X to abort  pass count 1
test looping press ^X to abort  pass count 2
test looping press ^X to abort  pass count 3
test looping press ^X to abort  pass count 4
test looping press ^X to abort  pass count 5
test looping press ^X to abort  pass count 6
test looping press ^X to abort  pass count 7
test looping press ^X to abort  pass count 8
test looping press ^X to abort  pass count 9
test looping press ^X to abort  pass count 10
test looping press ^X to abort  pass count 11
test looping press ^X to abort  pass count 12
test looping press ^X to abort  pass count 13
test looping press ^X to abort  pass count 14
test looping press ^X to abort  pass count 15
test looping press ^X to abort  pass count 16
test looping press ^X to abort  pass count 17
>BKM>
```

TDF 2 — Complete Memory Test, Single Pass

The **TDF 2** command runs a complete RAM Test for a single pass.

Syntax

TDF*unit 2 startaddress endaddress*

unit — This can be either **B** (Byte), **W** (Word), or **L** (Long).

startaddress — This is the address where the diagnostic test is to start.

endaddress — This is the address where the diagnostic test is to finish.

Example

```
>BKM>TDF 2 1000 1100
Single-pass Complete Memory Test loaded. Now executing
TDF.
Test Complete
>BKM>
```

TDF 3 — Complete Memory Test, Continuous

The **TDF 3** command runs a complete RAM Test continuously.

Syntax

TDF*unit 3 startaddress endaddress*

unit — This can be either **B** (Byte), **W** (Word), or **L** (Long).

startaddress — This is the address where the diagnostic test is to start.

endaddress — This is the address where the diagnostic test is to finish.

Example

```
>BKM>tdf 3 1000 1100
Continuous Complete Memory Test loaded. Now executing TDF.
test looping press ^X to abort  pass count 1
test looping press ^X to abort  pass count 2
test looping press ^X to abort  pass count 3
test looping press ^X to abort  pass count 4
test looping press ^X to abort  pass count 5
test looping press ^X to abort  pass count 6
test looping press ^X to abort  pass count 7
test looping press ^X to abort  pass count 8
test looping press ^X to abort  pass count 9
>BKM>
```

TDF 4 — CRC Test

The **TDF 4** command runs a CRC test over a specified range of memory.

Syntax

TDF*unit 4 startaddress endaddress*

unit — This can be either **B** (Byte), **W** (Word), or **L** (Long).

startaddress — This is the address where the diagnostic test is to start.

endaddress — This is the address where the diagnostic test is to finish.

Example

```
>BKM>tdf 4 1000 1100
CRC-16 Test loaded. Now executing TDF.
Completed... CRC-16 Value =  101
>BKM>
```

3.1.66 Target Diagnostic Function, Double (TDFD)

The **TDFD** command is for JTAG processors only. This command performs the same functions as the **TDF** commands with full 64-bit read/write operations.

Syntax

The syntax for the **TDFD** command is the same as the syntax for all of the other **TDF** commands, except that the *unit* option is not available since the unit is already set to double with this command.

TDFD *testnumber startaddress endaddress*

testnumber — This is the number of the diagnostic test you wish to run. There are five target diagnostic tests available, labelled **0 - 4**.

startaddress — This is the address where the diagnostic test is to start.

endaddress — This is the address where the diagnostic test is to finish.

See the **TDF** commands for more information.

3.1.67 Trace Enable (TE)

The **TE** command enables the non real-time software trace. Software trace is disabled by default. Note that software trace is automatically activated (a **>TRC>**

prompt appears) if you set a software breakpoint on ROM code or a software data breakpoint.

Syntax

```
TE
```

No arguments are required for this command. See also the **CF** (Configure) command.

3.1.68 TF Flash Configure Command (TF)

The **TF** commands are associated with programming flash memory. There are a number of usable **TF** commands, each with specific functions, as described in this section.

Target Flash Configure (TF CONF)

Use the **TF CONF** command to configure the emulator for flash programming and erasing. This command allows you to specify the device type, target RAM workspace, and base address. The actual erase and program instructions are downloaded into the RAM workspace. The target processor then executes the erase and program algorithms out of the RAM workspace.

Syntax

There are two different ways to use the **TF CONF** command. The first way is to use the following syntax.

```
TF CONF
```

With this syntax, you are prompted to specify the start address of the RAM workspace, and the start address of the flash bank on your target.

NOTE: To use this command syntax to configure the emulator for flash programming, first use the **TF DEVICE** command to specify the flash device on your target and the **TF CONF SIZE** command to specify the RAM workspace size.

The syntax for the second method of using the **TF CONF** command is as follows:

TF CONF *device_number RAM_workspace_address workspace_size base_address*

device_number — This selects a device from a list of available devices. This is the same number that is used with the **TF DEVICE** command.

RAM_workspace_address — This is the base address for the target RAM workspace.

workspace_size — This is the size (in bytes) required for the erase and program algorithm in RAM.

base_address — The is the base address (in hex) of the Flash device.

Example

The first example assumes that flash device AMD 29F040 (512x8) 4 devices is selected using the **TF DEVICE** command, and uses the **TF CONF** syntax. Select 0xff002000 for the RAM workspace. The base address of the flash should be 0xffc00000.

```
>BKM>tf conf
- BDM TFlash programming Interface Settings -
Current device selected       :    AMD    29F040    (  512 x  8 )  4 Devices
Start of work space in target :    00000000 > 0FF00200
Start address of the flash    :    FFFFFFFF > FFC00000
>BKM>
```

The second example uses the **TF CONF** *device_number RAM_workspace_address workspace_size base_address* syntax. The device number for the AMD 29F030 (512x8) 4 devices is 12, the RAM workspace address is 0xFF002000, the workspace size is 1968 bytes, and the flash base address is 0xFFD00000.

```
>BKM>TF CONF 12 FF002000 1968 FFD00000
12 FF002000 1912 FFD00000
>BKM>
```

See also the **TF DEVICE** command and the **TF CONF SIZE** command.

Target Flash Configure - Size (TF CONF SIZE)

Use the **TF CONF SIZE** command to specify the RAM workspace size. This is the size (in hex) required for the erase and program algorithm in RAM.

Syntax

TF CONF SIZE *size_in_hex*

Example

```
>BKM>TF CONF SIZE 0xFFFFFFFF
>BKM>
```

Target Flash Device (TF DEVICE)

Use the **TF DEVICE** command to select the particular flash device for erasing and programming that is included on your target.

Syntax

```
TF DEVICE
```

When you enter this command, Workbench displays a list of all available flash devices, each with a corresponding device number. You can find your device on the list and enter the device number that appears beside it.

Example

Display the device list.

```
>BKM>tf device
00:  AMD    29LV004T (  512 x  8 )  2 Devices
01:  AMD    29LV004B (  512 x  8 )  1 Device
02:  AMD    29LV004B (  512 x  8 )  2 Devices
03:  AMD    29LV004B (  512 x  8 )  4 Devices
04:  AMD    29LV008BT( 1024 x  8 )  1 Device
05:  AMD    29LV008BB( 1024 x  8 )  4 Devices
06:  AMD    29F010   (  128 x  8 )  1 Device
07:  AMD    29F010   (  128 x  8 )  2 Devices
08:  AMD    29F010   (  128 x  8 )  4 Devices
09:  AMD    29F010   (  128 x  8 )  8 Devices
10:  AMD    29F040   (  512 x  8 )  1 Device
11:  AMD    29F040   (  512 x  8 )  2 Devices
12:  AMD    29F040   (  512 x  8 )  4 Devices   -> Selected <-
13:  AMD    29F040   (  512 x  8 )  8 Devices
14:  AMD    29F080/81( 1024 x  8 )  1 Device
15:  AMD    29F080/81( 1024 x  8 )  2 Devices
16:  AMD    29F080/81( 1024 x  8 )  4 Devices
17:  AMD    29F080/81( 1024 x  8 )  8 Devices
18:  AMD    29F016/17( 2048 x  8 )  1 Device
19:  AMD    29F016/17( 2048 x  8 )  2 Devices
20:  AMD    29F016/17( 2048 x  8 )  4 Devices
21:  AMD    29F016/17( 2048 x  8 )  8 Devices
-> More <-
```

Keep pressing **ENTER** to display additional pages of flash devices. At the end of the list, Workbench displays a **New Choice >** prompt.

```
286:  ATMEL  49BV16xT ( 1024 x 16 )  2 Devices
287:  ATMEL  49BV6416T( 4096 x 16 )  1 Device
288:  ATMEL  49BV6416 ( 4096 x 16 )  1 Device
289:  SST    39xF016  ( 2048 x  8 )  1 Device
290:  SST    39xF016  ( 2048 x  8 )  2 Devices
291:  SST    39xF040  (  512 x  8 )  1 Device
292:  SST    39xF040  (  512 x  8 )  2 Devices
293:  SST    39xF160  ( 1024 x 16 )  1 Device
```

```
294:  SST    39xF160  ( 1024 x 16 )  2 Devices
295:  STM   M58BW016BB(  512 x 32 )  1 Device
296:  STM   M58BW016BT(  512 x 32 )  1 Device
297:  FrSc 5554H7Fv135(  256 x 64 )  1 Device
298:  Toshiba 58FVB641( 8192 x  8 )  1 Device
299:  Toshiba 58FVT641( 8192 x  8 )  1 Device
300:  Toshiba 58FVT641( 4096 x 16 )  1 Device
301:  Toshiba 58FVT641( 4096 x 16 )  2 Devices
New Choice >
```

At the **New Choice >** prompt, enter the number of your selection and press **ENTER**. This specifies your flash device.

Target Flash Erase (TF ERASE)

Use the **TF ERASE** command to erase a flash device. Prior to using the **TF ERASE** command, configure the emulator properly for flash programming using the **TF DEVICE**, **TF CONF**, and **TF TIMEOUT** commands. The erase algorithm is downloaded into RAM workspace (which is specified using the **TF CONF** command) on the target.

Syntax

TF ERASE *optional_addr1 optional_addr2*

optional_addr1 — This is the first address to erase (it must be a sector boundary)

optional_addr2 — This is the last address to erase (it must be a sector boundary)

If no optional addresses are given, the entire flash device is erased.

Examples

Erase the entire flash device, which was configured previously using the **TF CONF** and **TF DEVICE** commands.

```
BKM>tf erase
AMD        29F016    ( 2048 x 8 )    2 Devices
Erasing Flash(s)...Done
BKM>
```

Erase the flash devices from sector boundary address 0xFFC00000 to 0xFFD00000

```
BKM>tf erase ffc00000 ffd00000
AMD 29F016    ( 2048 x 8 )    2 Devices
Erasing Flash(s)...Done
BKM>
```

See also the **TF CONF**, **TF DEVICE**, and **TF TIMEOUT** commands.

Target Flash Test (TF TEST)

Use the **TF TEST** command to test erasing and programming target flash. The benefit of **TF TEST** is that a file is not required to program flash. Instead, the emulator directly programs a 0x2ff Byte test pattern into the target flash. The **TF TEST** command downloads the erase and program algorithms into RAM workspace (as determined by the **TF CONF** command). After downloading these algorithms, execute the erase/program algorithms with the **GO** command. The processor executes a software breakpoint when the algorithm completes.

The test pattern is ***WRS_FLASH*_** *(ASCII format)* repeated for 0x2ff bytes.

Syntax

```
TF TEST
GO addr
```

addr — This is the start address of the erase and/or program algorithm.

Example

Test erasing and programming target flash:

```
BKM>tf test
The flash algorithm : AMD   29F080/81     ( 1024 x 8 )    4Devices
has been loaded. The remaining Workspace is filled with a pattern
Flash to be programmed starts at                            = $E0000000
End of current flash device selected at                     = $E03FFFFF
End of the workspace is at                                  = $00000EA0
Programming algorithm starts at (PC set to this address)    = $0000002C
Erase algorithm starts at                                   = $0000001C
Work space starts at                                        = $00000000
Work space ends at                                          = $00000EA0
Test pattern starts at                                      = $000004A0

BKM>go 0000001c
RUN>
!HALT! - [msg90004] Unexpected software breakpoint encountered; PC=00000298

BKM>dm e0000000
E0000000: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ..........................

BKM>tf test
The flash algorithm : AMD   29F080/81     ( 1024 x 8 )    4Devices
has been loaded. The remaining Workspace is filled with a pattern
Flash to be programmed starts at                            = $E0000000
End of current flash device selected at                     = $E03FFFFF
End of the workspace is at                                  = $00000EA0
Programming algorithm starts at (PC set to this address)    = $0000002C
Erase algorithm starts at                                   = $0000001C
Work space starts at                                        = $00000000
Work space ends at                                          = $00000EA0
```

```
Test pattern starts at                                   = $000004A0
BKM>go 0000002c
RUN>
!HALT! - [msg90004] Unexpected software breakpoint encountered; PC=0xFF0023DC
BKM>dm e0000000
E0000000: 2A45 5354 5F46 4C41 5348 2A5F 2A45 5354     *WRS_FLASH*_*WRS
BKM>
E0000010: 5F46 4C41 5348 2A5F 2A45 5354 5F46 4C41     _FLASH*_*WRS_FLA
BKM>
E0000020: 5348 2A5F 2A45 5354 5F46 4C41 5348 2A5F     SH*_*WRS_FLASH*_
BKM>
E0000030: 2A45 5354 5F46 4C41 5348 2A5F 2A45 5354     *WRS_FLASH*_*WRS
BKM>
E0000040: 5F46 4C41 5348 2A5F 2A45 5354 5F46 4C41     _FLASH*_*WRS_FLA
BKM>
E0000050: 5348 2A5F 2A45 5354 5F46 4C41 5348 2A5F     SH*_*WRS_FLASH*_
BKM>
E0000060: 2A45 5354 5F46 4C41 5348 2A5F 2A45 5354     *WRS_FLASH*_*WRS
```

Target Flash Timeout (TF TIMEOUT)

The **TF TIMEOUT** command determines the amount of time in seconds that Wind River Workbench waits when erasing flash. Larger flash devices require a longer timeout period.

Syntax

TF TIMEOUT *optional_seconds*

optional_seconds — This is the time in seconds that Wind River Workbench waits when erasing flash.

If no parameter is specified, this command shows you the present value for timeout. Timeout values for both erase timeout and program timeout are displayed.

The **TF TIMEOUT** command only allows you to modify the erase timeout value. The erase timeout value can be any number of seconds between 1 and 500. The program timeout value can be any number of seconds between 1 and 8. To modify the program timeout value, use the following syntax:

TF PROGTIMEOUT *seconds*

Examples

Set the erase timeout period to 30 seconds.

```
>BKM>tf timeout 30
>BKM>
```

Set the progtimeout period to 6 seconds.

```
>BKM>tf progtimeout 6
>BKM>
```

Check the present timeout periods.

```
>BKM>tf timeout
erasetimeout is 30 seconds
progtimeout is 6 seconds
>BKM>
```

Target Flash Upload Sector (TF UPLOAD SECTOR)

This command provides a list of the starting addresses of all of the different sectors of your flash device based on the flash algorithm that you have specified.

Syntax

```
TF UPLOAD SECTOR
```

Example

```
>BKM>tf upload sector
Sector   0 : 0x00000000
Sector   1 : 0x00040000
Sector   2 : 0x00080000
Sector   3 : 0x000C0000
Sector   4 : 0x00100000
Sector   5 : 0x00140000
Sector   6 : 0x00180000
Sector   7 : 0x001C0000
>BKM>
```

3.1.69 Trigger On Breakpoint (TRG)

(Wind River ICE SX only)

This command allows you to set a trigger on a breakpoint. You can use the **TRG** command to set a trigger that will be executed when the CPU hits an internal hardware or software code breakpoint.

Syntax

[**SB**, **IHBC**] *breakAddress* **> TRG**

SB sets the system to trigger on a software breakpoint.

IHBC sets the system to trigger on an internal hardware breakpoint.

breakAddress = the breakpoint address.

Example

In the following example, a conditional software breakpoint is set at address 41244. After 100 executions of code at this address, the target will break, at which point the unit will set the **trigger out** signal based on the parameters set in the **CF** options. An internal hardware breakpoint is also set at 40578 and when this is hit, it will break and set **trigger out** in the same manner as above.

```
BKM > SB 41244 100 > TRG
BKM > IHBC 40578 > TRG
BKM > DB
         Software Code Breakpoints
1.   00041233  count = 0100 actual = 0000 enabled Trigger is ENABLED

         Hardware Breakpoints
2.   HBC   00040578 BRK enabled (i0.0) Trigger is ENABLED

BKM > GO 40400
RUN >
!BREAK! - [msg11001] Internal hardware breakpoint; PC = 0x00040578
BKM > GO

RUN >
!BREAK! - [msg12000] Software breakpoint; PC = 0x00041244
BKM >
```

3.1.70 Trigger Pulse Out (TRGOUT)

(Wind River ICE SX only)

This command generates a pulse on the Out pin of the Trigger port on the Wind River ICE SX unit. It can be used to trigger devices such as oscilloscopes and logic analyzers.

The pulse that is generated has a fixed width and an amplitude of 3.3 volts. The pulse can have either a positive or a negative amplitude, which is set using the **CF TRGOUT** command and setting it to either **PULSEHI** or **PULSELO**. **PULSEHI** triggers a pulse with a positive amplitude, and **PULSELO** triggers a pulse with a negative amplitude.

Syntax

```
TRGOUT
```

See also the **CF** command.

4 Scripting Commands

4.1 Introduction

The commands described in this chapter can be included in a script, which you can then run in Wind River Workbench using the following steps:

1. In the Workbench toolbar, select **Window > Show View > OCD Command Shell**.
2. In the **OCD Command Shell**, click the **Settings** button.

   The **Settings** dialog appears.
3. Enter the full path to your script in the **PlayBack File** field, or click **Browse** to navigate to a desired script.

4. Click **OK**.

   You are returned to the **OCD Command Shell**.

5. In the **OCD Command Shell**, click the **Playback File** button.

   Workbench runs the specified script.

4.2 Initialization Commands

4.2.1 LOADREG

Syntax

LOADREG *filename*

Load register values from a saved register file to the emulator NVRAM.

4.2.2 LOADSIMREG

Same as **LOADREG**.

4.2.3 SAVEEMULATORREG

Syntax

SAVEEMULATORREG *filename*

Upload all register values from the emulator NVRAM to a specified file. This command saves values for all registers, from both enabled and disabled register groups.

Bear in mind that the register values on the target may not be identical to the register values in the emulator NVRAM. To compare the register values on the target with the register values in the emulator NVRAM, use the command **SCT DIFF**.

4.2.4 SAVESIMREG

Same as **SAVEEMULATORREG**.

4.2.5 SCUPLOAD

Same as **SAVEEMULATORREG**.

4.2.6 SAVETARGETREG

Syntax

SAVETARGETREG *filename*

Upload all register values from the target to a file. This command saves values for all registers, from both enabled and disabled register groups.

Bear in mind that the register values on the target may not be identical to the register values in the emulator NVRAM. To compare the register values on the target with the register values in the emulator NVRAM, use the command **SCT DIFF**.

4.2.7 SCTUPLOAD

Same as **SAVETARGETREG**.

4.2.8 SAVENVRAM

Syntax

SAVENVRAM *filename*

Run a **PJ UPLOAD** command and save the output to the specified file. The **PJ UPLOAD** command returns the emulator's current configuration settings and the state of register groups (enabled or disabled), as well as Memory Management Unit (MMU) and boot line (BL) information, if available.

This command returns information only from the emulator NVRAM; it does not return information from the target.

4.2.9 RESTORENVRAM

Syntax

RESTORENVRAM *filename*

Set the emulator′s configuration options and other project settings to the values from the specified file. Use this command to restore the emulator NVRAM to settings you have previously saved using the **SAVENVRAM** command.

4.3 Download Commands

4.3.1 LOAD

Syntax

LOAD *filename offset*

Load the specified file to target memory. *offset* is a value in hex added to the address built into the image.

4.3.2 DOWNLOAD

Same as **LOAD**.

4.3.3 DOWN

Same as **LOAD**.

4.3.4 LOADVERIFY

Syntax

LOADVERIFY *filename offset*

Downloads a specified file to the target and verifies all memory writes to the target. *offset* is a value in hex added to the address built into the image.

4.3.5 LOADANDVERIFY

Same as **LOADVERIFY**.

4.3.6 VERIFYONLY

Syntax

VERIFYONLY *filename offset*

Verifies the specified file against target memory. This command does not download the specified file. *offset* is a value in hex added to the address built into the image.

4.3.7 VERIFY

Same as **VERIFYONLY**.

4.3.8 LOADMACRO

Syntax

LOADMACRO *filename*

Loads all macros coded in the specified file. This command allows only one filename.

4.3.9 LOADMACROS

Syntax

LOADMACROS *filename_1, filename_2, ...*

Loads all macros coded in the specified files. This command allows multiple filenames. Separate filenames with a space, a comma, and another space.

4.3.10 FLASHIT

Syntax

FLASHIT *filename offset*

Program the specified file to flash memory with the specified offset.

4.3.11 UPLOADBIN

Syntax

UPLOADBIN *filename start_addr end_addr* [append]

Upload a raw binary file to the specified *filename* over the range bounded by *start_addr* and *end_addr*. Use the argument **append** to upload the data without overwriting any previously existing data in the specified file.

The command is a byte-oriented access request; that is, the order of bytes in the file is the same as the order of bytes in memory.

4.3.12 SETBLOCK

Syntax

SETBLOCK *number*

Set the size of the download.

number is the number of kilobytes used on download. The default is 2.

4.3.13 BLOCKSIZE

Same as **SETBLOCK**.

4.3.14 BLOCK

Same as **SETBLOCK**.

4.4 Breakpoint Commands

4.4.1 BREAKENABLE

Enable all breakpoints.

4.4.2 BREAKDISABLE

Disable all breakpoints.

4.4.3 BREAKDELETE

Delete all breakpoints.

4.4.4 BREAKIN

Syntax

BREAKIN *symbol* or *address*

Set a breakpoint at the specified symbol or address.

4.5 Complex Breakpoint Commands

The nine commands in this section all use the same syntax, which take the following four arguments: *specifier*, *action*, *data*, and [*firmware_command*].

specifier - This can be any of the following:

- *symbol_name*
- *function_name*
- *filename:line_number*
- *address*

- *(address)*
- *start_address . . end_address*

action - This can be any of the following:

- TE (Trace Enable)
- TD (Trace Disable)
- TC (Trace Around Here)
- BRK (Break Here)
- TRIG (Trigger Trace Here)
- EXTRIG

data - This is a numeric value. Leading zeroes indicate the size of the data comparison. It can also take the format *xx/xxxx/xxxxxx/xxxxxxxx*.

[*firmware_command*] -- This syntax is allowed, but the functionality is not supported in Workbench. That is, using a firmware command as an argument here will not cause an error, but Workbench will not execute the command.

NOTE: Implementation of these commands varies between processor families. Depending on which processor you are using, you may not have access to all types of internal hardware breakpoints.

4.5.1 SETSB

Syntax

SETSB *specifier action data*

Set a software breakpoint at the specified location.

4.5.2 SETHBC

Syntax

SETHBC *specifier action data*

Set a hardware code breakpoint at the specified location.

4.5.3 SETHBD

Syntax

SETHBD *specifier action data*

Set a hardware data breakpoint at the specified location.

4.5.4 SETHBDR

Syntax

SETHBDR *specifier action data*

Set a hardware data breakpoint on any read to the specified location.

4.5.5 SETHBDW

Syntax

SETHBDW *specifier action data*

Set a hardware data breakpoint on any write to the specified location.

4.5.6 SETIHBC

Syntax

SETIHBC *specifier action data*

Set an internal hardware code breakpoint at the specified location.

4.5.7 SETIHBD

Syntax

SETIHBD *specifier action data*

Set an internal hardware data breakpoint at the specified location.

4.5.8 SETIHBDR

Syntax

SETIHBDR *specifier action data*

Set an internal hardware data breakpoint on any read to the specified location.

4.5.9 SETIHBDW

Syntax

SETIHBDR *specifier action data*

Set an internal hardware data breakpoint on any write to the specified location.

4.6 Run/Step Commands

4.6.1 G

Send a **GO** command to the emulator to start the target running.

4.6.2 H

Send a **HALT** command to the emulator to stop the target CPU and force the target into background mode. Also updates all open Workbench views.

4.6.3 ISTEP

Step one assembly-language instruction.

Syntax

ISTEP *count*

ISTEP will always step at least one instruction, so *count* is the number of steps beyond one you want to step. For example, the command **ISTEP 5** will step six times.

4.6.4 ISTEPOV

Step over one assembly-language instruction.

Syntax

ISTEPOV *count*

count is the number of times Workbench will step over.

4.6.5 PLAY

Play back commands from the required file.

Syntax

PLAY *filename*

4.6.6 RUNTO

Direct the emulator to step over branches. A temporary breakpoint is set at the next instruction and a **GO** command is issued. If a branch condition exists, all code in the branch will be executed before the breakpoint is taken.

Syntax

RUNTO *symbol* or *address*

Example

```
RUNTO main.c#65
```

4.6.7 SETPC

Set the Program Counter to a specified symbol or address.

Syntax

SETPC *symbol* or *address*

4.6.8 **STEP**

Step a source line into a function.

Syntax

STEP *count*

count is the number of times Workbench will step.

4.6.9 **STEPOV**

Step a source line over a function.

Syntax

STEPOV *count*

count is the number of times Workbench will step over.

4.6.10 **STEPOUT**

Step to the first line outside of the current function.

4.6.11 **WAIT**

Display an hourglass for the indicated number of seconds. Takes one argument, which is the number of seconds to wait before continuing; the limit is 10.

Syntax

WAIT *number_of_seconds*

NOTE: For previous users of visionCLICK: in visionCLICK, the **WAIT** command could be set to any number of seconds. Workbench manages its backend differently, so the **WAIT** command is limited to ten seconds.

4.7 Memory Commands

4.7.1 DUMPMEM

Syntax

DUMPMEM *filename addr number_of_elements element_size*

Dump memory at specified address *addr* to a specified file.

4.7.2 APPENDMEM

Syntax

APPENDMEM *filename addr number_of_elements element_size*

Append memory at specified address *addr* to a specified file, without overwriting any data that already exists in that file.

5
Wind River ICE Network Operation Commands

5.1 Wind River ICE SX Network Command Reference

NOTE: This chapter applies only to the Wind River ICE SX emulator.

5.1.1 APPLOAD

Wind River ICE SX supports dynamic firmware uploads. The **APPLOAD** command can be used to dynamically activate any set of firmware that is stored in the Wind River ICE SX unit. The **APPLOAD** command returns a port number.

Syntax

`APPLOAD` *processor*:*designator*:*Filename*

processor = This is the processor that is resident on your target, such as MPC8260.

designator = This is the reference designator that is assigned to the device that you are loading the firmware for. It is the designator that identifies that device in the JTAG scan chain, and it is the parameter that you specify in your board file.

Filename = This is the name of the board file to be loaded.

designator and *Filename* are optional in this command. If there is only a single device on the JTAG scan chain, you only need to specify a processor.

Example

The following example describes loading a file called **Motorola_2_PPC8260.brd** for a MPC8260 processor, with reference designator U0. Then the **DISPLAY** command is used to display the active firmware.

```
>NET>appload MPC8260:U0:Motorola_2_PPC8260.brd

Loading Application from FFS...

*****************************************************************************
Wind River ICE Target Driver
Copyright (c) 2004, Wind River Systems, Inc. All rights reserved
*****************************************************************************
Firmware Type PPC82XX  Version 1.0q  Created On: Jul 23 2002 17:45:20
Starting TCP BKM tools server  ESTD0 [ 1234] ........ PASSED
Starting TCP BKM tools server  TCPD0 [ 1235] ........ PASSED
Starting TCP BKM tools server LOADD0 [ 1236] ........ PASSED
Starting TCP TGTVIO    server TVIOD0 [ 1237] ........ PASSED
Starting with Saved Parameters....................... PASSED

*****************************************************************************
WIND River ICE UJD module
Copyright (c) 2002, Wind River Systems, Inc. All rights reserved
*****************************************************************************
Firmware Type UJD Module  Version 1.0m  Created On: Jul 23 2002 14:47:02
Initializing the UJD module......................... PASSED
Attaching to the UJD Module......................... PASSED
!PORT! - [ 1234] Attached to the Application Task.... PASSED
>NET>display
module_name:parms                port
ppc82xx.elf:U0:Motorola_2_PPC8260.brd 1234
servers.elf                      N/A
>NET>
```

See the **DISPLAY** command for more information.

5.1.2 ARP

Syntax

```
arp [-a] [-d] [-s]  [host]
```

Synopsis

Performs operations on the network ARP table

Description:

The **ARP** command allows for various operations on the network Address Resolution Protocol table (ARP table). The ARP table translates IP addresses into

Ethernet addresses and vice versa. The simplest form of the **ARP** command is **ARP** *host* where *host* is the IP address of another node. The command will display the associated Ethernet address for *host* if it is in the ARP table. The command **ARP -a** displays the entire contents of the ARP table.

It is possible to remove a host from the ARP table with the command **ARP -d** *host* where *host* is the IP address of the node you wish to delete. This is sometimes useful for testing or if a host changes its Ethernet address.

Hosts can be manually added to the ARP table with the command **ARP -s** *host ether_addr.* In this case, *host* is the IP address you wish to associate with *ether_addr*. *ether_addr* is a 6 byte number delimited with colon characters (for example, 00:0A:00:99:01:ff).

5.1.3 BKM

Syntax

BKM [**-n**]

Synopsis

Opens a channel to the **>BKM>** board prompt

Description

The **BKM** command allows users to open a channel to the BDM board and interact with the target via the **BKM** commands. Upon entering **BKM** Mode, the **>NET>** prompt will disappear and be replaced by a **>BKM>** prompt. All commands and responses will be from the BDM card in the Wind River ICE SX. To exit from the **>BKM>** prompt, type **CTRL+D**. This will return a **>NET>** prompt. The **-n** option suppresses sending an initial new line character. This is useful in some instances since a new line will restart an old **BKM** command.

Before a **BKM** command will pass, you need to perform an **APPLOAD**. See the **APPLOAD** command.

Example:

```
NET> bkm

Entering BKM Mode.
Opening channel ... Complete. ^D to exit BKM mode.

>BKM>
>BKM> in
```

```
***********************************************************************
Wind River ICE Initialization Sequence.
Copyright (c) Wind River Systems, Inc., 1999-2004. All rights reserved.
***********************************************************************
WIND River ICE UNIT#= none
    Support Expires....... 01/01/1995
    Warranty Number....... demo
    Target Processor...... 68341
    Serial Baud Rate...... 38400
    Host Debugger......... PSOS
VisionControl        Serial#= none    Firmware=c2.8a / 6.0x
Type CF For A Menu of Configuration Options
Initializing Background Debug Mode........Successful
>BKM> ^D
NET>
```

5.1.4 BOOTLOG

Syntax

BOOTLOG

Description

The **BOOTLOG** command displays the tests that were run during the last Wind River ICE SX reset. The information that is displayed includes information about the network configuration for Wind River ICE SX, as well as all of the hardware and firmware tests that were run and whether or not they passed.

Example

The following is the display that appears when the **BOOTLOG** command is entered at the **>NET>** prompt.

```
****************************************************************************
Wind River ICE SX Ethernet Platform
Copyright (c) 2004, Wind River Systems, Inc. All Rights Reserved
****************************************************************************
Firmware Type Wind River ICE SX BSP Version 1.2f Created On: Jun 10 2004
14:06:39
Configuring TCP/IP Network Suite:
IP Address......DHCP
            DHCP Sending DISCOVER 0 - 1 - +
            DHCP OFFER received
            DHCP Sending REQUEST 0 - 1 - +
            DHCP ACK received
DHCP Server at 172.16.12.114 returned IP address 172.16.17.35
DHCP IP address 172.16.17.35 lease time - 7 days, 0 hours, 0 min

Netmask.........DHCP = 0xFFFF0000
```

```
Default Gateway DHCP = 172.16.1.1
Routing.........Disabled
MAC Address.....00:A0:1E:00:32:E7
Pseudo device initialization................PASSED
TFTP device initialized.....................PASSED
Wind River ICE SX device initialized...........PASSED
Starting TCP/IP on Port A 10BaseT...........PASSED
Starting TCP/IP on Port B 100BaseT..........PASSED
Starting DHCP daemon server.................PASSED
Initializing FFS Driver.....................PASSED
FFS disk initialized 2640 Kbytes free........PASSED
FFS disk 0 percent fragmented...............PASSED
Disk volume 45.0.0 initialization...........PASSED
Starting TCP TGTCONS server TGTCONS [ 1232]..PASSED
Starting WRS shell server
Wind River ICE SX System Shell - Type HELP for list of commands
>NET>
```

5.1.5 CAT

Syntax

CAT *filename*

filename = This is the name of the file in Wind River ICE SX that you wish to display.

Description

The **CAT** command displays the contents of a file in the Wind River ICE SX Flash File System in ASCII format.

Example

The following example uses the **CAT** command to display the contents of the **8260_2.xml** file in ASCII format.

```
>NET>cat 8260_2.xml
<DEVICE_TABLE>
    <TABLE_MODE>SLOW</TABLE_MODE>
    <TABLE_CLOCK>6MHZ</TABLE_CLOCK>
    <!--Enable Multiple devices on a chain-->
    <TABLE_MULTI>ENABLE</TABLE_MULTI>
    <DEVICE>
        <NAME>8260_1</NAME>
        <DESCRIPTION>8260 Processor</DESCRIPTION>
        <TYPE>MICROPROCESSOR</TYPE>
        <TARGET>8260</TARGET>
        <DESIGNATOR>U0</DESIGNATOR>
        <IR_LEN>8</IR_LEN>
    </DEVICE>
    <DEVICE>
        <NAME>8260_2</NAME>
        <DESCRIPTION>8260 Processor</DESCRIPTION>
        <TYPE>MICROPROCESSOR</TYPE>
        <TARGET>8260</TARGET>
        <DESIGNATOR>U1</DESIGNATOR>
        <IR_LEN>8</IR_LEN>
    </DEVICE>
</DEVICE_TABLE
>NET>
```

5.1.6 COMTAP

NOTE: You must make a connection with your software before executing a **COMTAP** session.

Syntax

COMTAP *process_name* **[-o | -i] [-b]**

where *process_name* is the name of the server to tap (for example, bkm or tcpd)

[**-o** | **-i**] will trace only output or only input (both by default)

[**-b**] will monitor binary protocols (ASCII by default)

Synopsis

Traces host/Wind River ICE SX interactions.

Description:

COMTAP is a flexible tool for tracing host/Wind River ICE SX interactions.

Examples:

```
NET>comtap bkm -o        /* Will echo BKM data to a telnet session */
NET>comtap bkm -o -b     /* Will display dump of BKM data to telnet session*/
NET>comtap udpd          /* Will monitor all UDP debugger transactions */
NET>comtap loadd -o      /* Will monitot all LOADER ASCII data net->host */
```

Exceptions:

- Monitoring tcpd or udpd is always done in Binary Mode.
- Loading **.BDX** files will only display initial ASCII sequence unless the **-b** switch is specified.

- Since all data is echoed in BDM Mode, the **-o** switch should be used unless you are interested in double echo of input.
- BKM is a legal process name (even if it does not show up in a pstat) as long as a **BKM** session is active.

COMTAP can only be used from Telnet sessions. It will simply return on the RS232 console. To exit Comtap Mode, press the **ENTER** key in the Telnet session displaying the monitored data. Note that there may be some data queued up, so it may take a moment before output stops.

To see a list of running processes that are available for a **COMTAP** session, type the command **PSTAT** at the **>NET>** prompt.

5.1.7 DATE

Syntax

`DATE` [*yyyymmddhhmm*[**`.ss`**]]

Synopsis

Displays and/or sets the current system calendar date and time of day.

Description

The Wind River ICE SX system maintains a calendar date and time while it is running. This is used primarily for system log purposes to time correlate data. Entering the date command with no arguments yields the current system date and time of day. At system startup, the date is set to January 1, 1990 at midnight.

You may change the system date, although it is not necessary to do so unless you wish to use the system date/day to time the duration of tests, etc.

Example:

```
NET> date
0:15:20 Jan 1 1990
NET> date 199403050900.00
9:0:0 Mar 5 1994
NET> date
9:0:1 Mar 5 1994
NET>
```

5.1.8 DEFRAG

Syntax

```
DEFRAG
```

Description

The **DEFRAG** command optimizes space in the Flash File System on Wind River ICE SX, freeing up blocks of space that are not being used to their full potential. Defragmenting the Flash File System on Wind River ICE SX is similar to defragmenting a hard drive on your PC.

Example

The following example illustrates the output that appears when the **DEFRAG** command is being used.

```
>NET>defrag
Defragmenting the FFS will free up invalid blocks
!DO NOT reset or power-cycle the ICE during this operation
>NET>
```

5.1.9 DIR

Syntax

```
DIR
```

Description

The **DIR** command displays all of the files that are currently stored in Wind River ICE SX.

Example

```
>NET>dir
 Volume in FFS is 45.0.0
        2048 BITMAP.SYS
       14336 FLIST.SYS
     1850316 81v3r16p.bin
       18575 vp_dll.cfg
      108681 cachsp2k.elf
      211408 xyimage.hex
      211408 flxppc6xx.hex
      216561 servers.elf
     1668785 msc81xx.elf
         453 Motorola_8101ADS.brd
      343860 MSC81XX_U17.dat
      347 BD_WRS8260.brd
     7104 tdfppc.bin
   126497 cache6xx.elf
   303720 ppcjtflh.bin
   304984 ppcjtflhr.bin
  2544929 ppc82xx.elf
   343860 MPC82XX_U1.dat
      347 BD_WRS82xx.brd
   854560 flashbsp.bin
      628 8260_2.xml
   343860 MPC82XX_U0.dat
    22  file(s) 9477267  bytes
     0   dir(s)  17564525 bytes free
>NET>
```

5.1.10 DISPLAY

Syntax

`display`

Description

The **DISPLAY** command displays information about the application firmware that you have running on your Wind River ICE SX unit. Entering the **DISPLAY** command without any parameters will list all of the application firmware that is currently running on your Wind River ICE SX unit.

Example

This example uses the **DISPLAY** command with no parameters specified.

```
>NET>display
module_name:parms                port
ppc82xx.elf:U0:8260_2.xml        1234
servers.elf                      N/A
>NET>
```

5.1.11 ETHSETUP

Syntax

`ETHSETUP`

Synopsis

Modifies the Ethernet non-volatile setup parameters.

Description

The **ETHSETUP** command allows users to modify the non-volatile copy of the Ethernet setup parameters. Unlike the Configuration Switch method, **ETHSETUP** allows users to return to the **>NET>** prompt. Any new parameters will take effect after the next hardware reset (reset performed by tripping the switch on the rear of the emulator) or **RESET** command.

Example

```
>NET>ethsetup
                          Ethernet Setup Mode
Select from the operations below
     1. Display Basic IP parameters  2. Modify Basic IP parameters
     3. Display Routing parameters   4. Modify Routing parameters
     5. Display Server parameters    6. Modify Server parameters
     7. View ethernet address        8. Save parameters
     9. Exit setup mode             10. Port A/B select
    11. Advanced Options
Make a selection:
```

5.1.12 HELP

Syntax

HELP *topic*

Synopsis

Displays general or topic-specific online help

Description

The **HELP** command displays help for a given topic (when a *topic* is specified) or general help concerning available commands when entered with no arguments.

Example:

```
>NET>help
appload   arp       bkm       bootlog   brd       cat       cd
coldstart comtap    date      defrag    dir       display   download
du        echo      ethsetup  firmup    help      ifconfig  load
netinfo   netstat   ping      pstat     query     reset     route
runtime   sync      syslog    unload    version   ftp       telnet
tftp      tget      tput      update
>NET>help appload
Command        Description                      Syntax
=======    ====================    ==================================
appload    Load an application with APPLOAD </t/s(hostip or hostname)>
```

```
            CPU translation         <target name>
>NET>
```

5.1.13 IFCONFIG

Syntax

IFCONFIG *interface_number*
[**af** [**address** [*dest_addr*]] [**up**] [**down**]
[**netmask mask**] [**broadcast** *broad_addr*]]
[**arp** | **-arp**]
ifconfig -a

Synopsis

Reconfigures a network interface.

Description

The **IFCONFIG** command allows viewing and modifying of the network interfaces in Wind River ICE SX. There are two interfaces available: The Ethernet interface and Loopback interface. The Loopback interface should never need modification.

When combined with just the **-a** switch, the **IFCONFIG** command will display the state of the network interfaces, as shown below:

```
>NET>ifconfig -a
1: flags=120201<BROADCAST,UP>
    inet 172.16.17.35 netmask ffff0000 broadcast 172.16.255.255
6: flags=212<NOARP,UP>
    inet 127.0.0.1 netmask ff000000
>NET>
```

When used in combination with other flags, the **IFCONFIG** command allows users to temporarily change the network interface settings. The new settings will be lost on power-off or reset unless they are programmed using the **ETHSETUP** command. The example below shows how to change Wind River ICE SX's IP address:

```
NET> ifconfig 3 192.9.201.10 up
NET> ifconfig -a
1: flags=120201<BROADCAST,UP>
    inet 192.9.200.103 netmask ffffff00 broadcast 192.9.200.255
3: flags=202<NOARP,UP>
    inet 192.9.201.10 netmask ffffff00
NET>
```

5.1.14 NETINFO

Syntax

NETINFO

Description

The **NETINFO** command returns information about the Wind River ICE SX network configuration, including the IP address, netmask, and broadcast address.

Example

```
>NET>netinfo
TCP/IP on PORT B
1: flags=120201<BROADCAST,UP>
    inet 123.45.67.89 netmask ffff0000 broadcast 123.45.255.255
>NET>
```

This example shows the output of a **NETINFO** command. It returns the protocol the ICE is using (TCP/IP); the port in use (B); the number of broadcast flags in use, and whether they are up or down (**flags**); the IP address of the ICE unit (123.45.67.89); the ICE's netmask (FFFF0000); and the ICE's broadcast address (123.45.255.255).

5.1.15 NETSTAT

Syntax

NETSTAT [**-a**, **-i**, **-r**, **-s**]

-a - Displays status of all network connections

-i - Displays status of network interfaces

-r - Displays network routing table

-s - Displays per protocol information

Description

The **NETSTAT** command displays information about the TCP/IP protocol stack in Wind River ICE SX. It operates in a fashion similar to the **NETSTAT** command in a UNIX environment. The example below shows the output from various forms of the **NETSTAT** command.

Example

```
>NET>netstat
```

```
Proto     Local Address          Foreign Address         (state)
tcp       172.16.17.35.1234      172.16.18.142.1957       ESTABLISHED
>NET>netstat -a
Proto     Local Address          Foreign Address         (state)
udp       0.0.0.0.520
udp       0.0.0.0.111
udp       0.0.0.0.2049
udp       0.0.0.0.771
udp       0.0.0.0.1024
udp       0.0.0.0.68
udp       0.0.0.0.69
tcp       0.0.0.0.1237           0.0.0.0.0                LISTEN
tcp       0.0.0.0.111            0.0.0.0.0                LISTEN
tcp       0.0.0.0.1233           0.0.0.0.0                LISTEN
tcp       0.0.0.0.1236           0.0.0.0.0                LISTEN
tcp       0.0.0.0.23             0.0.0.0.0                LISTEN
tcp       0.0.0.0.1232           0.0.0.0.0                LISTEN
tcp       0.0.0.0.1024           0.0.0.0.0                LISTEN
tcp       172.16.17.35.1234      172.16.18.142.1957       ESTABLISHED
tcp       0.0.0.0.21             0.0.0.0.0                LISTEN
tcp       0.0.0.0.1235           0.0.0.0.0                LISTEN
>NET>netstat -i
I/F  Mtu  Address          Ipkts     Ierrs     Opkts     Oerrs     Queue
1    1500 172.16.17.35     32232     0         1993      0         100
6    1536 127.0.0.1        151       0         151       0         0
>NET>netstat -r
Destination         Gateway             Flags     Interface
default             172.16.1.1          UG        1
127.0.0.1           127.0.0.1           U         6
172.16.0.0          172.16.17.35        U         1
>NET>netstat -s
udp:
    11123 datagrams delivered to users
    0 datagrams received for unknown ports
    281 datagrams received with other errors
    997 datagrams sent
tcp:
    1104 segments sent
    0 segments retransmitted
    0 segments sent with RST flag
    1139 segments received
    0 segments received in error
    0 failed TCP connection attempts
    1 TCP connections reset
ip:
    13581 received from interfaces
    0 drops due to format errors
    1038 drops due to invalid addresses
    0 drops due to unknown protocol
    0 discarded with no problems
    2101 supplied by IP user protocols
    0 dropped due to no routes
    0 IP datagrams forwarded
>NET>
```

5.1.16 PING

Syntax

PING [**-s**] *host_ip* [*repeat_count*]

Synopsis

Sends **ICMP ECHO_REQUEST** messages to the specified *host_ip*.

Description

The **PING** command provides a method to test that a connection can be established over the network to a specific host. It is very useful in diagnosing the ability to send packets to and from a host from the Wind River ICE SX unit.

By specifying the **-s** switch, the **PING** command will repeat the operation 10 times or *repeat_count* times. When reporting results, the **PING** command will print the elapsed time to send/receive for each **ICMP ECHO_REQUEST / ECHO_REPLY** pair. The time is rounded to the nearest millisecond.

5.1.17 PSTAT

Syntax

PSTAT

Synopsis

Lists the state of processes on the Wind River ICE SX.

Description

The Wind River ICE SX is a multi-tasking system capable of executing many processes at once. The **PSTAT** command allows users to view the state of most of the system processes. This is useful when trying to determine what host process is associated with a server on the Wind River ICE SX.

There are several types of processes that run on the Wind River ICE SX. They are outlined in the following table:

Table 5-1 **Wind River ICE Processes**

Type	Description
PrmPro	Permanent Process-Cannot be killed by the user

Table 5-1 **Wind River ICE Processes**

Type	Description
PrmServ	Permanent Server -- Can be killed and restarted by the user
DynServ	Dynamic Server -- Can be killed by the user; will restart as needed

Example:

```
>NET>pstat
NAME          TSK_ID    TYPE       PRIORITY  STATE
--------------------------------------------------------------------------
 dhcp          00000017  DynProc    00000025  ConnWait  0.0.0.0 (0)
 TGTCONS      *00000026  PrmServ    00000190  ConnWait  0.0.0.0 (1232)
 SHELLD       *00000027  PrmProc    00000065  ConnWait  0.0.0.0 (1233)
 PPC82XX       00000045             00000180  running
 ESTD0        *00000047  PrmServ    00000200  Connected 172.16.18.142 (1957)
 TCPD0        *00000048  PrmServ    00000200  ConnWait  0.0.0.0 (1235)
 LOADD0       *00000049  PrmServ    00000210  ConnWait  0.0.0.0 (1236)
 TVIOD0       *00000050  PrmServ    00000190  ConnWait  0.0.0.0 (1237)
 UJD           00000066  PrmServ    00000210  running
             * = Tapable with comtap
>NET>
```

5.1.18 QUERY

Syntax

QUERY

Description

The **QUERY** command displays a list of all of the firmware versions that are currently stored in the Wind River ICE SX Flash File System. The display is a table that includes the file names, the type of file, the version of the firmware, and the date and time that it was added to Wind River ICE SX.

Example

```
>NET>query

FFSNAME           NAME          TYPE          VER      DATE/TIME
--------------------------------------------------------------------------
cache82xx.elf     CACHE82XX     CACHE APP     2.1a     Oct 22 2004 16:08:16
servers.elf       UJD Module    SERVER        2.3a     Oct 21 2004 13:50:34
cache6xx.elf      CACHE6XX      CACHE APP     2.1a     Oct 21 2004 13:51:16
```

```
ppc82xx.elf        PPC82XX       TARGET APP    2.4a     Oct 22 2004 16:08:41
ppc6x.elf          PPC6XX        TARGET APP    2.3a     Oct 21 2004 13:51:53

>NET>
```

5.1.19 RESET

Syntax

RESET

Synopsis

Issues a cold start reset of the emulator.

Description

The **RESET** command power cycles the Wind River ICE SX unit. This command is provided as a convenience and is typically issued after using the **ETHSETUP** command to modify the basic operating parameters of the Wind River ICE SX unit.

5.1.20 ROUTE

Syntax

ROUTE [**ADD,DELETE**] [*hostip* | *netip*] *destination* [*gateway* [*metric*]]

Synopsis

Adds or deletes network routes on the Wind River ICE SX.

Description

The **ROUTE** command allows users to temporarily add and delete network routes on Wind River ICE SX. To permanently add/delete routes, use **ETHSETUP** or consult your software documentation if dynamic routing is enabled. The **ROUTE** command is provided primarily for the purpose of testing routing entries or adding temporary routes.

Example:

```
NET> netstat -r
Destination         Gateway             Flags     Interface
127.0.0.1           127.0.0.1           U         3
192.9.200.0         192.9.200.103       U         1
NET> route add 192.9.201.0 192.9.200.100 1
```

```
assuming route via gateway
add net 192.9.201.0: gateway 192.9.200.100 (192.9.200.100)
NET> route add 192.9.202.0 192.9.200.101 1
assuming route via gateway
add net 192.9.202.0: gateway 192.9.200.101 (192.9.200.101)
NET> route delete 192.9.202.0 192.9.200.101
delete net 192.9.202.0: gateway 192.9.200.101 (192.9.200.101)
NET> netstat -r
Destination         Gateway             Flags     Interface
127.0.0.1           127.0.0.1           U         3
192.9.200.0         192.9.200.103       U         1
192.9.201.0         192.9.200.100       UG        1
NET>
```

5.1.21 RUNTIME

Syntax

RUNTIME

Description

The **RUNTIME** command displays the length of time in days, hours, minutes, and seconds since the last Wind River ICE SX initialization.

Example

```
>NET>runtime

Time since last reset/power cycle
730 Days, 9 Hours, 27 Minutes, 3 Seconds
>NET>
```

5.1.22 SYSLOG

Syntax

SYSLOG

Synopsis

Displays a log of the last 25 events on the Wind River ICE SX unit.

Description

Wind River ICE SX maintains a log of the last 25 significant events that have occurred since the last time it was reset. These events are stored in a FIFO that can be displayed by using the **SYSLOG** command. The **SYSLOG** command is useful for looking at the usage of particular facilities on Wind River ICE SX.

In addition to usage information, any abnormal conditions are also saved in the system log. For example, if a server loses synchronization with a client and restarts itself, a message will be placed in the system log.

System log information is shown with the most recent event first, followed by older events. The time stamps are relative to system start time or the last time/date set command. The format of the time is *dd:hh:mm:ss*.

Example:

```
>NET>syslog
[00:00:00:06]  LOADAPPS  : bootapps.lst not found in FFS
[00:00:00:06]  TGTCONS   : Server awaiting connection.
[00:00:00:06]  TGTCONS   : Starting TCP TGTCONS   server TGTCONS [ 1232]
[00:00:00:06]  FFS       : Task Started
[00:00:00:00]  DHCP      : Rebinding Time - 6 days, 3 hours, 0 min
[00:00:00:00]  DHCP      : Renewal Time - 3 days, 12 hours, 0 min
[00:00:00:00]  DHCP      : BOUND state
[00:00:00:00]  DHCP      : daemon started
>NET>
```

5.1.23 TELNET

Syntax

`TELNET` *IP_Address*

Synopsis

Allows connection to other telnet capable hosts from the RS232 console.

Description

The **TELNET** command allows users to telnet from the RS232 console on Wind River ICE SX to any other host on the network that Wind River ICE SX can talk to. The telnet client in Wind River ICE SX is a close implementation of the BSD telnet command found in most UNIX systems.

5.1.24 UNLOAD

Syntax

`UNLOAD` *port_number* or *NameOfProcess* *port_number* = This is the port number that was assigned to your activated firmware during the **APPLOAD** command. You can view the port number using the **DISPLAY** command.

NameOfProcess = You can specify the name of the application firmware that you wish to deactivate instead of its associated port number. View the name of the application using the **DISPLAY** command.

Instead of specifying a port number or a process name, using the command **UNLOAD *** will unload all active firmware sets.

Description

Wind River ICE SX supports dynamic firmware uploads and downloads. The **UNLOAD** command can be used to dynamically deactivate any set of firmware that is currently active on the Wind River ICE SX unit. Currently active processes can be viewed using the **DISPLAY** command.

Example

The first example uses the display command to view the active sets of firmware. Then a set of firmware is deleted using the **UNLOAD** command.

```
>NET>display

module_name:parms                port
ppc82xx.elf:U0:Motorola_2_PPC8260.brd 1234
servers.elf                      N/A
>NET>unload /1234

Requesting 'ppc82xx' to delete Itself................ PASSED
>NET>
```

The second example uses the **UNLOAD *** command to delete two sets of firmware that are active simultaneously.

```
>NET>unload *

Requesting 'ppc82xx' to delete Itself................ PASSED
Requesting 'servers' to delete Itself................ PASSED
>NET>
```

5.1.25 VERSION

Syntax

```
VERSION
```

Description

The **VERSION** command displays the build date and revision number of the Wind River ICE SX firmware.

Example:

```
>NET>version

******************************************************************
Wind River ICE Ethernet Controller
Copyright © Wind River Systems, Inc. 1999-2004.  All rights reserved
******************************************************************
Firmware Type BSP  Version 2.0a  Created On: Oct 22 2004 16:07:17


OS     version = V2.5.0
REPC   version = V2.5.0
NA     version = V4.0.5
>NET>
```