bdiGDB

JTAG debug interface for GNU Debugger

MIPS64

User Manual

Manual Version 1.13 for BDI3000

abatron

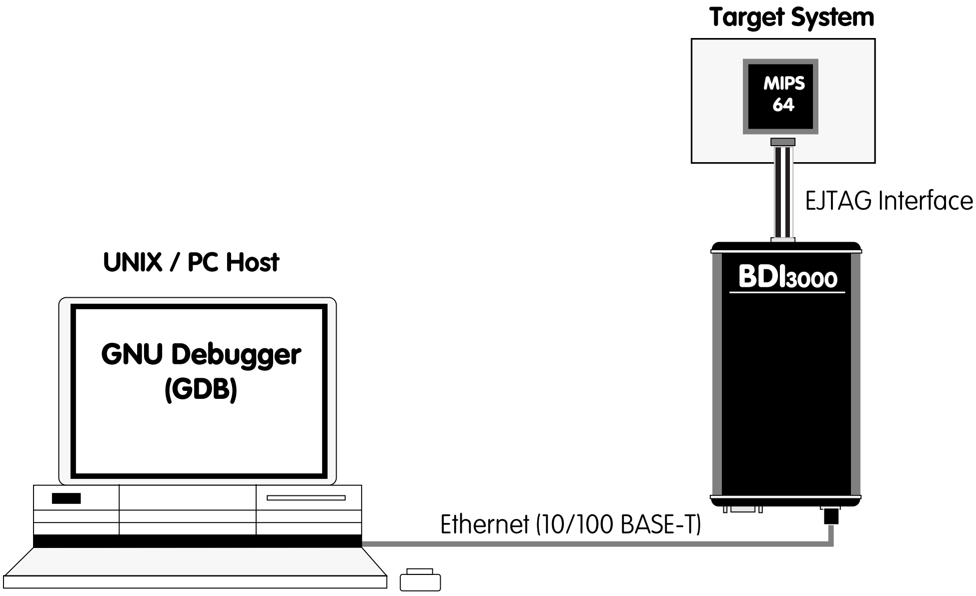
7.4 Appendices

1 Introduction

bdiGDB enhances the GNU debugger (GDB), with EJTAG debugging for MIPS64 based targets. With the built-in Ethernet interface you get a very fast code download speed. No target communication channel (e.g. serial line) is wasted for debugging purposes. Even better, you can use fast Ethernet debugging with target systems without network capability. The host to BDI communication uses the standard GDB remote protocol.

An additional Telnet interface is available for special debug tasks (e.g. force a hardware reset, program flash memory).

The following figure shows how the BDI3000 interface is connected between the host and the target:



1.1 BDI3000

The BDI3000 is the main part of the bdiGDB system. This small box implements the interface between the JTAG pins of the target CPU and a 10/100Base-T Ethernet connector. The firmware of the BDI3000 can be updated by the user with a simple Linux/Windows configuration program or interactively via Telnet/TFTP. The BDI3000 supports 1.2 – 5.0 Volts target systems.

.

1.2 BDI Configuration

As an initial setup, the IP address of the BDI3000, the IP address of the host with the configuration file and the name of the configuration file is stored within the flash of the BDI3000.
Every time the BDI3000 is powered on, it reads the configuration file via TFTP.
Following an example of a typical configuration file:

```
; bdiGDB configuration file for MIPS Malta 5Kc board
; -------------------------------------------------
;
; This configuration uses the YAMON monitor setup the board
;
[INIT]
;
; Setup TLB
WTLB   0x00000600   0x01E00017   ;Monitor Flash 2 x 4MB, uncached DVG
;


[TARGET]
JTAGCLOCK     1             ;use 16 MHz JTAG clock
CPUTYPE       M5KC          ;the used target CPU type
ENDIAN        BIG           ;target is big endian
STARTUP       STOP 4000     ;STOP mode is used to let the monitor init the system
WORKSPACE     0xA0000080    ;workspace in target RAM for fast download
BREAKMODE     SOFT          ;SOFT or HARD, HARD uses hardware breakpoints
STEPMODE      JTAG          ;JTAG, HWBP or SWBP
;VECTOR       CATCH         ;catch unhandled exceptions


[HOST]
IP            151.120.25.119
FILE          E:\cygwin\home\bdidemo\mips64\fibo.x
FORMAT        ELF
;FILE         E:\temp\malta_mon.bin
;FORMAT       BIN 0xA0400000
LOAD          MANUAL        ;load code MANUAL or AUTO after reset


[FLASH]
WORKSPACE     0xa0000000    ;workspace in target RAM for fast programming algorithm
CHIPTYPE      I28BX16       ;Flash type
CHIPSIZE      0x200000      ;The size of one flash chip in bytes
BUSWIDTH      32            ;The width of the flash memory bus in bits (8 | 16 | 32)
;FILE         E:\cygwin\home\bdidemo\mips64\malta_mon.cfg
;FORMAT       BIN 0xbc000000
;ERASE        0xbc000000    ;erase sector 0


[REGS]
DMM1          0xFF300000    ;DSU base address
DMM2          0xBF000000    ;Memory mapped registers
FILE          E:\cygwin\home\bdidemo\mips64\reg5kc.def
```

Based on the information in the configuration file, the target is automatically initialized after every reset.

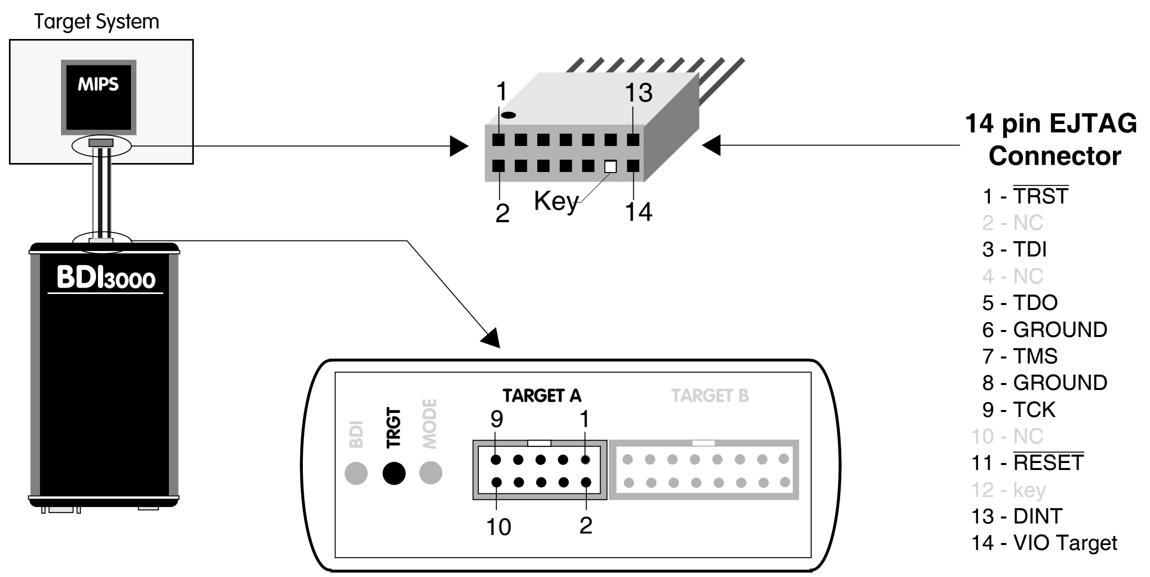
2 Installation

2.1 Connecting the BDI3000 to Target

The cables to the target system are designed for EJTAG 2.5 compatible boards. In case where the target system has the same connector layout, the cable can be directly connected.

⚠

In order to ensure reliable operation of the BDI (EMC, runtimes, etc.) the target cable length must not exceed 25 cm (10").



The green LED «TRGT» marked light up when target is powered up

For TARGET A connector signals see table on next page.

Warning:
Before you can use the BDI3000 with an other target processor type (e.g. MIPS <--> ARM), a new setup has to be done (see chapter 2.5). During this process the target cable must be disconnected from the target system.

⚠

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming a new firmware for an other target CPU.

TARGET A Connector Signals

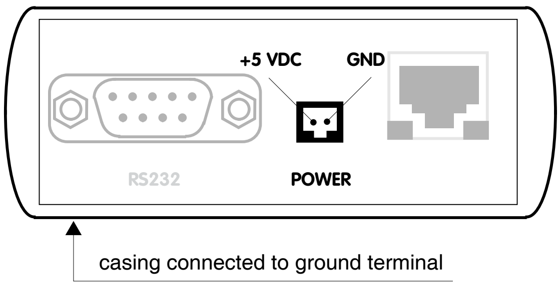
Pin	Name	Description
1	DINT	**EJTAG Debug Interrupt** This output of the BDI3000 connects to the target DINT line.
2	$\overline{\text{TRST}}$	**EJTAG Test Reset** This output of the BDI3000 resets the JTAG TAP controller on the target.
3+5	GND	**System Ground**
4	TCK	**EJTAG Test Clock** This output of the BDI3000 connects to the target TCK line.
6	TMS	**EJTAG Test Mode Select** This output of the BDI3000 connects to the target TMS line.
7	$\overline{\text{RESET}}$	This open collector output of the BDI3000 is used to reset the target system.
8	TDI	**EJTAG Test Data In** This output of the BDI3000 connects to the target TDI line.
9	VIO Target	**1.2 – 5.0V:** This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vdd I/O on the target board.
10	TDO	**EJTAG Test Data Out** This input to the BDI3000 connects to the target TDO line.

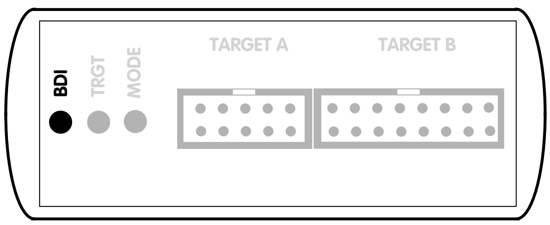
2.2 Connecting the BDI3000 to Power Supply

The BDI3000 needs to be supplied with the enclosed power supply from Abatron (5VDC).

Before use, check if the mains voltage is in accordance with the input voltage printed on power supply. Make sure that, while operating, the power supply is not covered up and not situated near a heater or in direct sun light. Dry location use only.

For error-free operation, the power supply to the BDI3000 must be between 4.75V and 5.25V DC. **The maximal tolerable supply voltage is 5.25 VDC. Any higher voltage or a wrong polarity might destroy the electronics.**





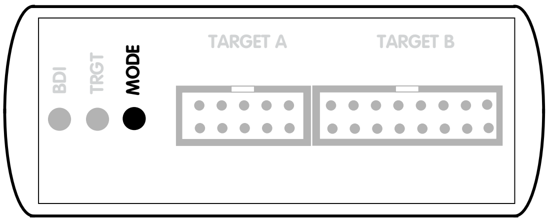
The green LED «BDI» marked light up when 5V power is connected to the BDI3000

Please switch on the system in the following sequence:

- 1 –> external power supply
- 2 –> target system

2.3 Status LED «MODE»

The built in LED indicates the following BDI states:



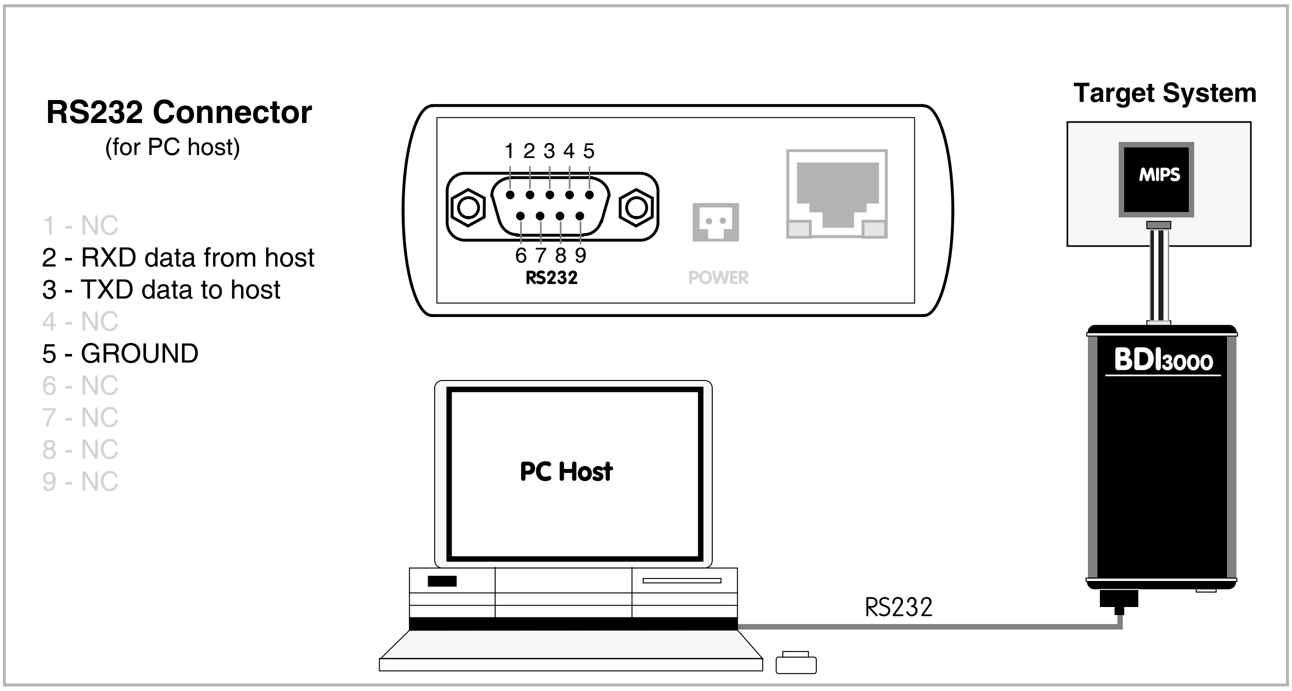
MODE LED	BDI STATES
OFF	The BDI is ready for use, the firmware is already loaded.
ON	The output voltage from the power supply is too low.
BLINK	The BDI «loader mode» is active (an invalid firmware is loaded or loading firmware is active).

2.4 Connecting the BDI3000 to Host

2.4.1 Serial line communication

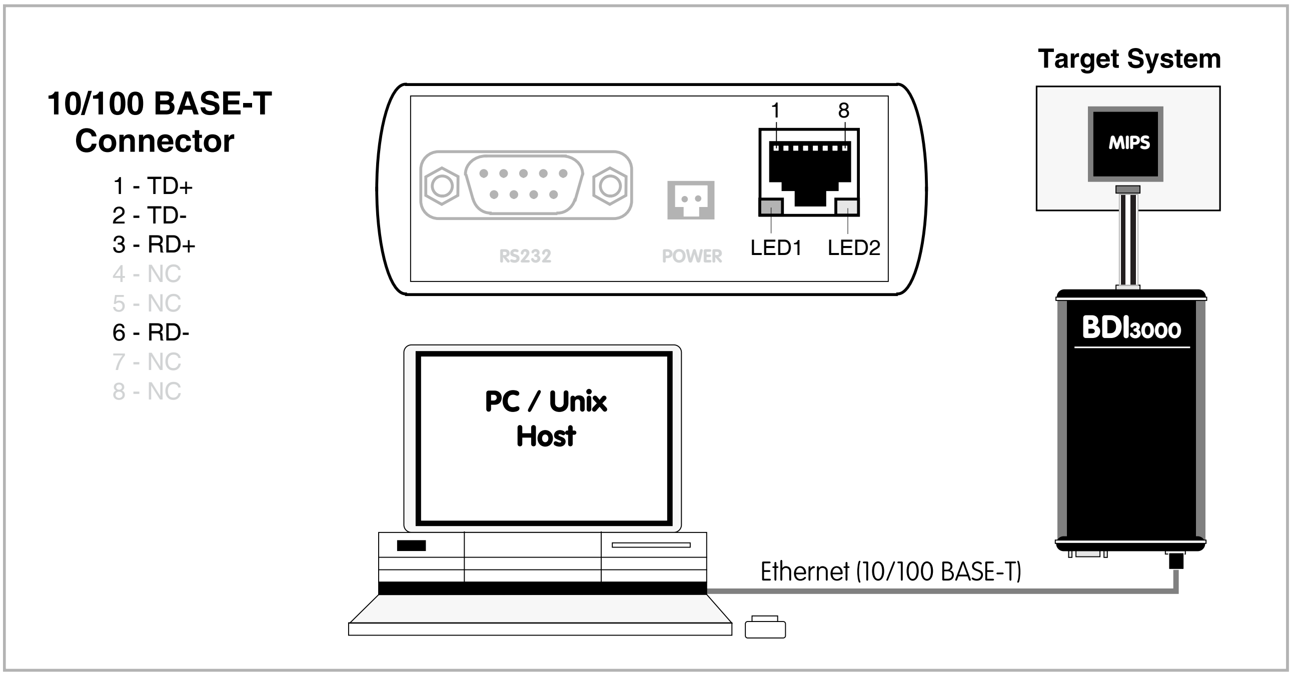
Serial line communication is only used for the initial configuration of the bdiGDB system.

The host is connected to the BDI through the serial interface (COM1...COM4). The communication cable (included) between BDI and Host is a serial cable. There is the same connector pinout for the BDI and for the Host side (Refer to Figure below).



2.4.2 Ethernet communication

The BDI3000 has a built-in 10/100 BASE-T Ethernet interface (see figure below). Connect an UTP (Unshielded Twisted Pair) cable to the BD3000. Contact your network administrator if you have questions about the network.



The following explains the meanings of the built-in LED lights:

LED	Function	Description
LED 1 (green)	Link / Activity	When this LED light is ON, data link is successful between the UTP port of the BDI3000 and the hub to which it is connected. The LED blinks when the BDI3000 is receiving or transmitting data.
LED 2 (amber)	Speed	When this LED light is ON, 100Mb/s mode is selected (default). When this LED light is OFF, 10Mb/s mode is selected

2.5 Installation of the Configuration Software

On the enclosed diskette you will find the BDI configuration software and the firmware required for the BDI3000. For Windows users there is also a TFTP server included.

The following files are on the diskette.

b30r5kgd.exe	Windows Configuration program
b30r5kgd.xxx	Firmware for the BDI3000
tftpsrv.exe	TFTP server for Windows (WIN32 console application)
*.cfg	Configuration files
*.def	Register definition files
bdisetup.zip	ZIP Archive with the Setup Tool sources for Linux / UNIX hosts.

Overview of an installation / configuration process:

- Create a new directory on your hard disk
- Copy the entire contents of the enclosed diskette into this directory
- Linux only: extract the setup tool sources and build the setup tool
- Use the setup tool or Telnet (default IP) to load/update the BDI firmware
  Note: A new BDI has no firmware loaded.
- Use the setup tool or Telnet (default IP) to load the initial configuration parameters
  - IP address of the BDI.
  - IP address of the host with the configuration file.
  - Name of the configuration file. This file is accessed via TFTP.
  - Optional network parameters (subnet mask, default gateway).

Activating BOOTP:
The BDI can get the network configuration and the name of the configuration file also via BOOTP. For this simple enter 0.0.0.0 as the BDI's IP address (see following chapters). If present, the subnet mask and the default gateway (router) is taken from the BOOTP vendor-specific field as defined in RFC 1533.

With the Linux setup tool, simply use the default parameters for the -c option:

```
[root@LINUX_1 bdisetup]# ./bdisetup -c -p/dev/ttyS0 -b57
```

The MAC address is derived from the serial number as follows:
MAC: 00-0C-01-xx-xx-xx , replace the xx-xx-xx with the 6 left digits of the serial number
Example: SN# 33123407 ==>> 00-0C-01-33-12-34

Default IP: 192.168.53.72
Before the BDI is configured the first time, it has a default IP of 192.168.53.72 that allows an initial configuration via Ethernet (Telnet or Setup Tools). If your host is not able to connect to this default IP, then the initial configuration has to be done via the serial connection.

2.5.1 Configuration with a Linux / Unix host

The firmware update and the initial configuration of the BDI3000 is done with a command line utility. In the ZIP Archive bdisetup.zip are all sources to build this utility. More information about this utility can be found at the top in the bdisetup.c source file. There is also a make file included.
Starting the tool without any parameter displays information about the syntax and parameters.

⚠

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming the firmware for an other target CPU family.

Following the steps to bring-up a new BDI3000:

1. Build the setup tool:
The setup tool is delivered only as source files. This allows to build the tool on any Linux / Unix host. To build the tool, simply start the make utility.

```
[root@LINUX_1 bdisetup]# make
cc -O2   -c -o bdisetup.o bdisetup.c
cc -O2   -c -o bdicnf.o bdicnf.c
cc -O2   -c -o bdidll.o bdidll.c
cc -s bdisetup.o bdicnf.o bdidll.o -o bdisetup
```

2. Check the serial connection to the BDI:
With "bdisetup -v" you may check the serial connection to the BDI. The BDI will respond with information about the current loaded firmware and network configuration.
Note: Login as root, otherwise you probably have no access to the serial port.

```
$ ./bdisetup -v -p/dev/ttyS0 -b115
BDI Type : BDI3000 (SN: 30000154)
Loader   : V1.00
Firmware : unknown
MAC      : ff-ff-ff-ff-ff-ff
IP Addr  : 255.255.255.255
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 255.255.255.255
Config   : ÿÿÿÿÿÿÿ........
```

3. Load/Update the BDI firmware:
With "bdisetup -u" the firmware is programmed into the BDI3000 flash memory. This configures the BDI for the target you are using. Based on the parameters -a and -t, the tool selects the correct firmware file. If the firmware file is in the same directory as the setup tool, there is no need to enter a -d parameter.

```
$ ./bdisetup -u -p/dev/ttyS0 -b115 -aGDB -tMIPS64
Connecting to BDI loader
Programming firmware with ./b30r5kgd.100
Erasing firmware flash ....
Erasing firmware flash passed
Programming firmware flash ....
Programming firmware flash passed
```

4. Transmit the initial configuration parameters:
With "bdisetup -c" the configuration parameters are written to the flash memory within the BDI.
The following parameters are used to configure the BDI:

BDI IP Address	The IP address for the BDI3000. Ask your network administrator for assigning an IP address to this BDI3000. Every BDI3000 in your network needs a different IP address.
Subnet Mask	The subnet mask of the network where the BDI is connected to. A subnet mask of 255.255.255.255 disables the gateway feature. Ask your network administrator for the correct subnet mask. If the BDI and the host are in the same subnet, it is not necessary to enter a subnet mask.
Default Gateway	Enter the IP address of the default gateway. Ask your network administrator for the correct gateway IP address. If the gateway feature is disabled, you may enter 255.255.255.255 or any other value.
Config - Host IP Address	Enter the IP address of the host with the configuration file. The configuration file is automatically read by the BDI after every start-up via TFTP. If the host IP is 255.255.255.255 then the setup tool stores the configuration read from the file into the BDI internal flash memory. In this case no TFTP server is necessary.
Configuration file	Enter the full path and name of the configuration file. This file is read by the setup tool or via TFTP. Keep in mind that TFTP has it’s own root directory (usual /tftpboot).

```
$ ./bdisetup -c -p/dev/ttyS0 -b115 \
> -i151.120.25.102 \
> -h151.120.25.112 \
> -fe:/bdi3000/mytarget.cfg
Connecting to BDI loader
Writing network configuration
Configuration passed
```

5. Check configuration and exit loader mode:
The BDI is in loader mode when there is no valid firmware loaded or you connect to it with the setup tool. While in loader mode, the Mode LED is blinking. The BDI will not respond to network requests while in loader mode. To exit loader mode, the "bdisetup -v -s" can be used. You may also power-off the BDI, wait some time (1min.) and power-on it again to exit loader mode.

```
$ ./bdisetup -v -p/dev/ttyS0 -b115 -s
BDI Type : BDI3000 (SN: 30000154)
Loader   : V1.00
Firmware : V1.00 bdiGDB for MIPS64
MAC      : 00-0c-01-30-00-01
IP Addr  : 151.120.25.102
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 151.120.25.112
Config   : /bdi3000/mytarget.cfg
```

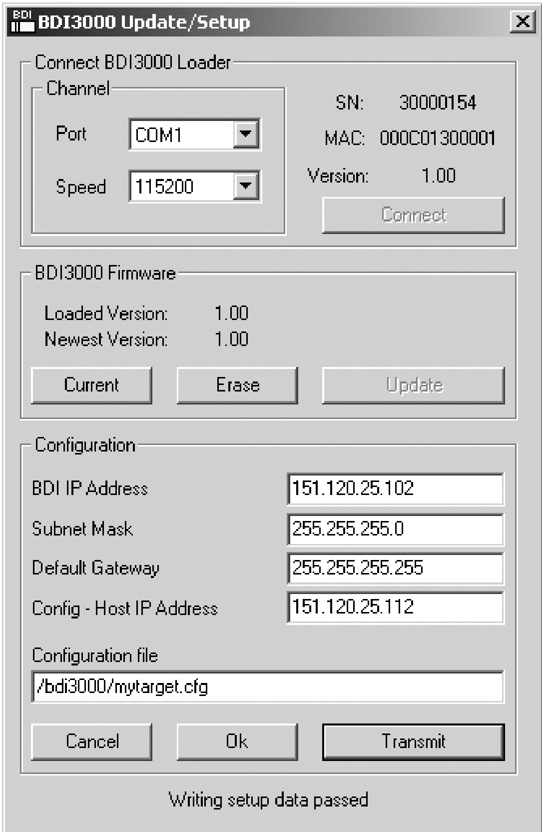
The Mode LED should go off, and you can try to connect to the BDI via Telnet.

```
$ telnet 151.120.25.102
```

2.5.2 Configuration with a Windows host

First make sure that the BDI is properly connected (see Chapter 2.1 to 2.4).

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming the firmware for an other target CPU family.



dialog box «BDI3000 Update/Setup»

Before you can use the BDI3000 together with the GNU debugger, you must store the initial configuration parameters in the BDI3000 flash memory. The following options allow you to do this:

Port	Select the communication port where the BDI3000 is connected during this setup session. If you select Network, make sure the Loader is already active (Mode LED blinking). If there is already a firmware loaded and running, use the Telnet command "boot loader" to activate Loader Mode.
Speed	Select the baudrate used to communicate with the BDI3000 loader during this setup session.
Connect	Click on this button to establish a connection with the BDI3000 loader. Once connected, the BDI3000 remains in loader mode until it is restarted or this dialog box is closed.
Current	Press this button to read back the current loaded BDI3000 firmware version. The current firmware version will be displayed.

Erase	Press this button to erase the current loaded firmware.
Update	This button is only active if there is a newer firmware version present in the execution directory of the bdiGDB setup software. Press this button to write the new firmware into the BDI3000 flash memory.
BDI IP Address	Enter the IP address for the BDI3000. Use the following format: xxx.xxx.xxx.xxx e.g.151.120.25.101 Ask your network administrator for assigning an IP address to this BDI3000. Every BDI3000 in your network needs a different IP address.
Subnet Mask	Enter the subnet mask of the network where the BDI is connected to. Use the following format: xxx.xxx.xxx.xxxe.g.255.255.255.0 A subnet mask of 255.255.255.255 disables the gateway feature. Ask your network administrator for the correct subnet mask.
Default Gateway	Enter the IP address of the default gateway. Ask your network administrator for the correct gateway IP address. If the gateway feature is disabled, you may enter 255.255.255.255 or any other value.
Config - Host IP Address	Enter the IP address of the host with the configuration file. The configuration file is automatically read by the BDI after every start-up via TFTP. If the host IP is 255.255.255.255 then the setup tool stores the configuration read from the file into the BDI internal flash memory. In this case no TFTP server is necessary.
Configuration file	Enter the full path and name of the configuration file. This name is transmitted to the TFTP server when reading the configuration file.
Transmit	Click on this button to store the configuration in the BDI3000 flash memory.

Note:
Using this setup tool via the Network channel is only possible if the BDI3000 is already in Loader mode (Mode LED blinking). To force Loader mode, enter "boot loader" at the Telnet. The setup tool tries first to establish a connection to the Loader via the IP address present in the "BDI IP Address" entry field. If there is no connection established after a time-out, it tries to connect to the default IP (192.168.53.72).

2.5.3 Configuration via Telnet / TFTP

The firmware update and the initial configuration of the BDI3000 can also be done interactively via a Telnet connection and a running TFTP server on the host with the firmware file. In cases where it is not possible to connect to the default IP, the initial setup has to be done via a serial connection.

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming the firmware for an other target CPU family.

Following the steps to bring-up a new BDI3000 or updating the firmware.
Connect to the BDI Loader via Telnet.
If a firmware is already running enter "boot loader" and reconnect via Telnet.

```
$ telnet 192.168.53.72
or
$ telnet <your BDI IP address>
```

Update the network parameters so it matches your needs:

```
LDR>network
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 192.168.53.72
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 255.255.255.255
    Config File :

LDR>netip 151.120.25.102
LDR>nethost 151.120.25.112
LDR>netfile /bdi3000/mytarget.cfg


LDR>network
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg

LDR>network save
saving network configuration ... passed
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg
```

In case the subnet has changed, reboot before trying to load the firmware

```
LDR>boot loader
```

Connect again via Telnet and program the firmware into the BDI flash:

```
$ telnet 151.120.25.102

LDR>info
    BDI Firmware: not loaded
    BDI CPLD ID : 01285043
    BDI CPLD UES: ffffffff
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg

LDR>fwload e:/temp/b30r5kgd.100
erasing firmware flash ... passed
programming firmware flash ... passed

LDR>info
    BDI Firmware: 36 / 1.00
    BDI CPLD ID : 01285043
    BDI CPLD UES: ffffffff
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg
LDR>
```

To boot now into the firmware use:

```
LDR>boot
```

The Mode LED should go off, and you can try to connect to the BDI again via Telnet.

```
telnet 151.120.25.102
```

2.6 Testing the BDI3000 to host connection

After the initial setup is done, you can test the communication between the host and the BDI3000. There is no need for a target configuration file and no TFTP server is needed on the host.

- If not already done, connect the BDI3000 system to the network.
- Power-up the BDI3000.
- Start a Telnet client on the host and connect to the BDI3000 (the IP address you entered during initial configuration).
- If everything is okay, a sign on message like «BDI Debugger for Embedded PowerPC» and a list of the available commands should be displayed in the Telnet window.

2.7 TFTP server for Windows

The bdiGDB system uses TFTP to access the configuration file and to load the application program. Because there is no TFTP server bundled with Windows, Abatron provides a TFTP server application **tftpsrv.exe**. This WIN32 console application runs as normal user application (not as a system service).

Command line syntax: tftpsrv [p] [w] [dRootDirectory]

Without any parameter, the server starts in read-only mode. This means, only read access request from the client are granted. This is the normal working mode. The bdiGDB system needs only read access to the configuration and program files.

The parameter [p] enables protocol output to the console window. Try it.
The parameter [w] enables write accesses to the host file system.
The parameter [d] allows to define a root directory.

`tftpsrv p`	Starts the TFTP server and enables protocol output
`tftpsrv p w`	Starts the TFTP server, enables protocol output and write accesses are allowed.
`tftpsrv dC:\tftp\`	Starts the TFTP server and allows only access to files in C:\tftp and its subdirectories. As file name, use relative names. For example "bdi\mpc750.cfg" accesses "C:\tftp\bdi\mpc750.cfg"

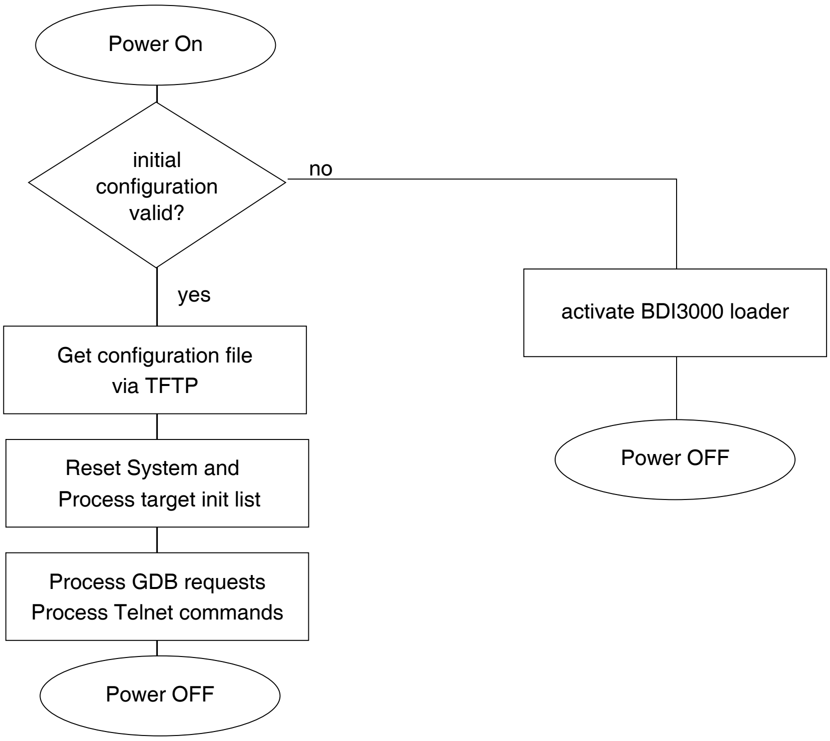
You may enter the TFTP server into the Startup group so the server is started every time you login.

3 Using bdiGDB

3.1 Principle of operation

The firmware within the BDI handles the GDB request and accesses the target memory or registers via the JTAG interface. There is no need for any debug software on the target system. After loading the code via TFTP debugging can begin at the very first assembler statement.

Whenever the BDI system is powered-up the following sequence starts:



3.2 Configuration File

The configuration file is automatically read by the BDI3000 after every power on.
The syntax of this file is as follows:

```
; comment
[part name]
identifier parameter1 parameter2 ..... parameterN  ; comment
identifier parameter1 parameter2 ..... parameterN
.....
[part name]
identifier parameter1 parameter2 ..... parameterN
identifier parameter1 parameter2 ..... parameterN
.....
             etc.
```

Numeric parameters can be entered as decimal (e.g. 700) or as hexadecimal (0x80000).

Note about how to enter 64bit values:

The syntax for 64 bit parameters is :	[<high word>_]<low word>
Hex values may also be entered as:	0xnnnnnnnnnnnnnnnn

The "high word" (optional) and "low word" can be entered as decimal or hexadecimal. They are handled as two separate values concatenated with an underscore.

Examples:

```
0x0123456789abcdef        =>>    0x0123456789abcdef
0x01234567_0x89abcdef     =>>    0x0123456789abcdef
1_0                       =>>    0x0000000100000000
256                       =>>    0x0000000000000100
3_0x1234                  =>>    0x0000000300001234
0x80000000_0              =>>    0x8000000000000000
```


3.2.1 Part [INIT]

The part [INIT] defines a list of commands which should be executed every time the target comes out of reset. The commands are used to get the target ready for loading the program file.

WGPR register value
: Write value to the selected general purpose register.
  - register — the register number 0 .. 31
  - value — the value to write into the register
  - Example: WGPR 0 5

WREG name value
: Write value to the selected register/memory by name
  - name — the case sensitive register name from the reg def file
  - value — the value to write to the register/memory
  - Example: WREG pc 0x80001000

WCP0 register value
: Write value to the selected Coprocessor 0 register.

WCP1 register value
: Write value to the selected Coprocessor 1 register.

WCP2 register value
: Write value to the selected Coprocessor 2 register.
  - register — the register number 0 .. 31, add 0x0n00 for Select n, add 0x1000 for a 64bit register
  - value — the value to write into the register
  - Example: WCP0 13 0x00000000 ;Clear Cause Register

WCTR register value
: Write value to the selected XLP Control register.
  - register — the register address
  - value — the value to write into the register
  - Example: WCTR 0x300 0x01;LSU Configuration

WM8 address value
: Write a byte (8bit) to the selected memory place.
  - address — the memory address
  - value — the value to write to the target memory
  - Example: WM8 0xFFFFFA21 0x04 ; SYPCR: watchdog disable ...

WM16 address value
: Write a half word (16bit) to the selected memory place.
  - address — the memory address
  - value — the value to write to the target memory
  - Example: WM16 0x02200200 0x0002 ; TBSCR

WM32 address value
: Write a word (32bit) to the selected memory place.
  - address — the memory address
  - value — the value to write to the target memory
  - Example: WM32 0x02200000 0x01632440 ; SIUMCR

WM64 address value
: Write a double word (64bit) to the selected memory place.
  - address — the memory address
  - value — the value to write to the target memory
  - Example: WM64 0x1000 0x01234567_0x89abcdef

DELAY value	Delay for the selected time. value — the delay time in milliseconds (1...30000) Example: DELAY 500 ; delay for 0.5 seconds
IVIC [ways sets]	This entry invalidates the instruction cache. way — the number of ways in the IC sets — the number of sets in the IC Example: IVIC 2 256 ; Invalidate IC, 2 way, 256 sets
IVDC [ways sets]	This entry invalidates the data cache. way — the number of ways in the DC sets — the number of sets in the DC Example: IVDC 2 64 ; Invalidate DC, 2 way, 64 sets
WTLB vpn rpn	Adds an entry to the TLB array. For parameter description see below. vpn — the virtual page number, size and ASID rpn — the real page number, coherency and DVG bits Example: WTLB 0x00000500 0x01FC0017 ;Boot ROM 2 x 1MB

Adding entries to the TLB:
Sometimes it is necessary to setup the TLB before memory can be accessed. This is because on a MIPS the MMU is always enabled. The init list entry WTLB allows an initial setup of the TLB array. The first WTLB entry clears also the whole TLB array.

The vpn parameter defines the memory region, virtual page number, size and ASID:

```
+--+---------------------+-------------------------+-+----+--------+
|R |        fill         |            VPN          |-|SIZE|  ASID  |
+--+---------------------+-------------------------+-+----+--------+
 2           22                      27             2  4      8
```

The SIZE (size) field decodes as follows:

```
0 = (1KB)     1 = 4KB       2 = 16KB      3 = 64KB      4 = 256KB
5 = 1MB       6 = 4MB       7 = 16MB      8 = 64MB      9 = 256MB
```

The rpn parameter defines the real page number, coherency and DVG bits:

```
+-------------+-----------------+--+---+---+
|    EPFN     |       PFN       |--| C |DVG|
+-------------+-----------------+--+---+---+
     n                20          2   3   3
```

The field EPFN (extended page frame number) is used for physical address bits nn:32.
The field positions are selected so the physical address becomes readable.

The following example clears the TLB and adds one entry to access ROM via address 0x00000000 and adds an other entry to access the ROM also via address 0xc000000210000000 (xkseg).

```
WTLB    0x00000600              0x01E00017  ;Monitor Flash 2 x 4MB, uncached DVG
WTLB    0xc0000002_0x10000600   0x01E00017  ;Monitor Flash 2 x 4MB, uncached DVG
```

3.2.2 Part [TARGET]

The part [TARGET] defines some target specific values.

CPUTYPE type [32BIT]

This value gives the BDI information about the connected CPU. The optional 32BIT parameter forces the BDI to transfer only 32-bit register values to GDB. This allows to connect with a GDB built for MIPS32.

type	M5KC, M5KM, M5KP, M5KF, CNMIPS, CNMIPS2, CNMIPSII, CNMIPS3, XLP
Example:	CPUTYPE M5KC CPUTYPE M5KF 32BIT

ENDIAN format

This entry defines the endiannes of the memory system.

format	The endiannes of the target memory: BIG (default), LITTLE
Example:	ENDIAN LITTLE

JTAGCLOCK value

With this value you select the JTAG clock frequency.

value	The JTAG clock frequency in Hertz or an index value from the following table: 0 = 32 MHz 3 = 8 MHz 1 = 16 MHz 4 = 5 MHz 2 = 11 MHz 5 = 4 MHz
Example:	JTAGCLOCK 1 ; JTAG clock is 16 MHz

JTAGDELAY wait

This entry defines a wait time in Run-Test/Idle state before a value is read or after a value was written via JTAG. Useful when accessing slow memory with a fast JTAG clock. Allows to optimize download performance.

wait	number of 8 TCK's in Run-Test/Idle state
Example:	JTAGDELAY 4 ; Wait for 32 TCK's

BDIMODE mode [param]

This parameter selects the BDI debugging mode. The following modes are supported:

LOADONLY	Loads and starts the application core. No debugging via JTAG port.
AGENT	The debug agent runs within the BDI. There is no need for any debug software on the target. This mode accepts a second parameter. If RUN is entered as a second parameter, the loaded application will be started immediately, otherwise only the PC is set and BDI waits for GDB requests.
Example:	BDIMODE AGENT RUN

RESET type [time]

This parameter selects the type of reset the BDI applies to the target during power-up or when "reset" is entered via Telnet. Default is HARD.

NONE	No reset is applied (default).
JTAG	Reset is forced via the EJTAG control register.
HARD	Reset is applied via the EJTAG connector reset pin. The "time" parameter defines the time in milliseconds the BDI assert the reset signal.
Example:	RESET JTAG

POWERUP delay

This parameter defines a delay in milliseconds the BDI waits after the target has been powered-up until JTAG communications starts.

delay	the power-up start delay in milliseconds (default 2 sec.)
Example:	POWERUP 5000 ;start delay after power-up

WAKEUP time

This entry in the init list allows to define a delay time (in ms) the BDI inserts between releasing the RESET line and starting communicating with the target. This init list entry may be necessary if RESET is delayed on its way to the processors reset pin.

time	the delay time in milliseconds
Example:	WAKEUP 3000 ; insert 3sec wake-up time

STARTUP mode [runtime]

This parameter selects the target startup mode:

RESET	This default mode forces the target to debug mode immediately out of reset. No code is executed after reset.
STOP	In this mode, the BDI lets the target execute code for "runtime" milliseconds after reset. This mode is useful when monitor code should initialize the target system.
RUN	After reset, the target executes code until stopped by the Telnet "halt" command.
WAIT	This special startup mode allows to force a cnMIPS core immediately to debug mode once it is released from reset.
Example:	STARTUP STOP 3000 ; let the CPU run for 3 seconds

BREAKMODE mode

This parameter defines how breakpoints are implemented. The current mode can also be changed via the Telnet interface

SOFT	This is the normal mode. Breakpoints are implemented by replacing code with a SDBBR instruction.
HARD	In this mode, the EJTAG breakpoint hardware is used.
Example:	BREAKMODE HARD

STEPMODE mode

This parameter defines how single step (instruction step) is implemented. The alternate step modes (HWBP or SWBP) are useful when stepping instructions that causes a TLB miss exception. Not all targets allow to use all step modes. Some of them do not implement the EJTAG step mode others support only one hardware instruction breakpoint.

JTAG	This is the default mode. The step feature of the EJTAG debug interface is used for single stepping.
HWBP	In this mode, one or two hardware breakpoints are used to implement single stepping.
SWBP	In this mode, one or two software breakpoints are used to implement single stepping.
Example:	STEPMODE HWBP

VECTOR CATCH	When this line is present, the BDI catches all unhandled exceptions. Catching exceptions is only possible if the vector table at 0x80000000 is writable. Example: VECTOR CATCH ; catch unhandled exception
WORKSPACE address	If a workspace is defined, the BDI uses a faster download / upload mode. The workspace is used for a short code sequence. There must be at least 64 bytes of RAM available for this purpose. address the address of the RAM area Example: WORKSPACE 0xA0000080
SIO port [baudrate]	When this line is present, a TCP/IP channel is routed to the BDI's RS232 connector. The port parameter defines the TCP port used for this BDI to host communication. You may choose any port except 0 and the default Telnet port (23). On the host, open a Telnet session using this port. Now you should see the UART output in this Telnet session. You can use the normal Telnet connection to the BDI in parallel, they work completely independent. Also input to the UART is implemented. port The TCP/IP port used for the host communication. baudrate The BDI supports 2400 ... 115200 baud Example: SIO 7 9600 ;TCP port for virtual IO
REGLIST list	This parameter defines what registers are sent to GDB. By default only the standard registers are sent (gpr's, sr, lo, hi, bad, cause, pc, dummy fpr's). The following names are use to select a register group: STD The standard registers. CP0 The CP0 registers. Example: REGLIST STD CP0 ; standard and CP0 registres
MCDEBUG value	cnMIPS only: This parameters allows to define the value for the Multicore Debug register. The BDI writes this value to the register just before the core is restarted. This allows for example to halt multiple cores if one core enters debug mode. For a given core when GSDB is set, the BDI automatically also sets the TE bit for hardware breakpoints (IBCx,DBCx). value the value to write to the Multicore Debug Register just before the core exits debug mode. Example: MCDEBUG 0x1107 ;set GSDB,MSKM0, clear MCDx

Daisy chained JTAG devices:
For MIPS targets, the BDI can also handle systems with multiple devices connected to the JTAG scan chain. In order to put the other devices into BYPASS mode and to count for the additional bypass registers, the BDI needs some information about the scan chain layout. Enter the number (count) and total instruction register (irlen) length of the devices present before the MIPS chip (Predecessor). Enter the appropriate information also for the devices following the MIPS chip (Successor):

SCANPRED count irlen — This value gives the BDI information about JTAG devices present before the MIPS chip in the JTAG scan chain.

count	The number of preceding devices
irlen	The sum of the length of all preceding instruction registers (IR).
Example:	SCANPRED 1 8 ; one device with an IR length of 8

SCANSUCC count irlen — This value gives the BDI information about JTAG devices present after the MIPS chip in the JTAG scan chain.

count	The number of succeeding devices
irlen	The sum of the length of all succeeding instruction registers (IR).
Example:	SCANSUCC 2 12 ; two device with an IR length of 8+4

Low level JTAG scan chain configuration:
Sometimes it is necessary to configure the test access port (TAP) of the target before the EJTAG debug interface is visible and accessible in the usual way. The BDI supports this configuration in a very generic way via the SCANINIT configuration option. It accepts a string that defines the JTAG sequence to execute. The following example shows how to use these commands:

```
; Configure Master TAP to make EJTAG TAP visible
SCANINIT    t1:w1000:t0:w1000:     ;toggle TRST
SCANINIT    i5=05:w100000          ;enter MIPS EJTAG mode
;
```

The following low level JTAG commands are supported in the string. Use ":" between commands.

```
  I<n>=<...b2b1b0>   write IR, b0 is first scanned
  D<n>=<...b2b1b0>   write DR, b0 is first scanned
                       n  : the number of bits 1..256
                       bx : a data byte, two hex digits
  W<n>               wait for n (decimal) micro seconds
  T1                 assert TRST
  T0                 release TRST
  R1                 assert RESET
  R0                 release RESET
  CH<n>              clock TCK n (decimal) times with TMS high
  CL<n>              clock TCK n (decimal) times with TMS low
```

The SCANINIT sequence replaces the standard TAP reset sequence used in the BDI firmware. This standard TAP reset sequence asserts TRST for 1 ms and then toggles TCK 5 times with TMS high. After this init sequence the scan chain should look like defined with SCANPRED and SCANSUCC.

3.2.3 Part [HOST]

The part [HOST] defines some host specific values.

IP ipaddress	The IP address of the host. ipaddress the IP address in the form xxx.xxx.xxx.xxx Example: IP 151.120.25.100
FILE filename	The file name of the program file. This name is used to access the application file via TFTP. If the filename starts with a $, this $ is replace with the path of the configuration file name. filename the filename including the full path or $ for relative path. Example: FILE F:\gnu\demo\mips\test.elf FILE $test.elf
FORMAT format [offset]	The format of the image file and an optional load address offset. If the image is already stored in ROM on the target, select ROM as the format. The optional parameter "offset" is added to any load address read from the image file. format SREC, BIN, AOUT, ELF or ROM Example: FORMAT ELF FORMAT ELF 0x10000
LOAD mode	In Agent mode, this parameters defines if the code is loaded automatically after every reset. mode AUTO, MANUAL Example: LOAD MANUAL
START address	The address where to start the program file. If this value is not defined and the core is not in ROM, the address is taken from the code file. If this value is not defined and the core is already in ROM, the PC will not be set before starting the target. This means, the program starts at the normal reset address (0x00000000). address the address where to start the program file Example: START 0x10000
PROMPT string	This entry defines a new Telnet prompt. The current prompt can also be changed via the Telnet interface. Example: PROMPT M5KF>
DUMP filename	The default file name used for the Telnet DUMP command. filename the filename including the full path Example: DUMP dump.bin

TELNET mode

By default the BDI sends echoes for the received characters and supports command history and line editing. If it should not send echoes and let the Telnet client in "line mode", add this entry to the configuration file.

mode	ECHO (default), NOECHO or LINE
Example:	TELNET NOECHO ; use old line mode

DEBUGPORT port [RECONNECT] [NS-MT]

The TCP port GDB uses to access the target. If the RECONNECT parameter is present, an open TCP/IP connection (Telnet/GDB) will be closed if there is a connect request from the same host (same IP address). The option NS-MT enables "Non-stop Multi-Threaded Debugging" in GDB.

port	the TCP port number (default = 2001)
Example:	DEBUGPORT 2001

Non-stop Multi-Threaded Debugging in GDB:

In this mode the cores are mapped to GDB threads and only one GDB session is started to access all cores. Google for "Non-stop Multi-Threaded Debugging in GDB" to find the document that explains this mode from the GDB point of view. The support on the BDI side for this mode is still experimental but you are free to use it. To enable this mode add the NS-MT option to the DEBUGPORT parameter in the BDI configuration [HOST] section. To enable this mode in GDB use:

```
(gdb) set target-async 1
(gdb) set pagination off
(gdb) set non-stop on
(gdb) target remote bdi2000:2001
```

Be aware that cores are actually not threads. Cores have individual caches and also their own hardware breakpoint registers. This can conflict with the way GDB handles threads.

3.2.4 Part [FLASH]

The Telnet interface supports programming and erasing of flash memories. The bdiGDB system has to know which type of flash is used, how the chip(s) are connected to the CPU and which sectors to erase in case the ERASE command is entered without any parameter.

CHIPTYPE type	This parameter defines the type of flash used. It is used to select the correct programming algorithm.	
	type	AM29F, AM29BX8, AM29BX16, I28BX8, I28BX16, AT49, AT49X8, AT49X16, STRATAX8, STRATAX16, MIRROR, MIRRORX8, MIRRORX16, S29M32X16, S29WSRX16, S29GLSX16, S29VSRX16 M58X32, AM29DX16, AM29DX32
	Example:	CHIPTYPE AM29F
CHIPSIZE size	The size of **one** flash chip in bytes (e.g. AM29F010 = 0x20000). This value is used to calculate the starting address of the current flash memory bank.	
	size	the size of one flash chip in bytes
	Example:	CHIPSIZE 0x80000
BUSWIDTH width	Enter the width of the memory bus that leads to the flash chips. Do not enter the width of the flash chip itself. The parameter CHIPTYPE carries the information about the number of data lines connected to one flash chip. For example, enter 16 if you are using two AM29F010 to build a 16bit flash memory bank.	
	with	the width of the flash memory bus in bits (8 | 16 | 32 | 64)
	Example:	BUSWIDTH 32
FILE filename	The name of the file to program into the flash. This name is used to access the file via TFTP. If the filename starts with a $, this $ is replace with the path of the configuration file name. This name may be overridden interactively at the Telnet interface.	
	filename	the filename including the full path or $ for relative path.
	Example:	FILE F:\gnu\mips\bootrom.hex FILE $bootrom.hex
FORMAT format [offset]	The format of the file and an optional address offset. The optional parameter "offset" is added to any load address read from the program file.	
	format	SREC, BIN, AOUT or ELF
	Example:	FORMAT SREC FORMAT ELF 0x10000
WORKSPACE address	If a workspace is defined, the BDI uses a faster programming algorithm that runs out of RAM on the target system. Otherwise, the algorithm is processed within the BDI. The workspace is used for a 1kByte data buffer and to store the algorithm code. There must be at least 2kBytes of RAM available for this purpose.	
	address	the address of the RAM area
	Example:	WORKSPACE 0x00000000

ERASE addr [increment count] [mode [wait]]

The flash memory may be individually erased or unlocked via the Telnet interface. In order to make erasing of multiple flash sectors easier, you can enter an erase list. All entries in the erase list will be processed if you enter ERASE at the Telnet prompt without any parameter. This list is also used if you enter UNLOCK at the Telnet without any parameters. With the "increment" and "count" option you can erase multiple equal sized sectors with one entry in the erase list.

address	Address of the flash sector, block or chip to erase
increment	If present, the address offset to the next flash sector
count	If present, the number of equal sized sectors to erase
mode	BLOCK, CHIP, UNLOCK Without this optional parameter, the BDI executes a sector erase. If supported by the chip, you can also specify a block or chip erase. If UNLOCK is defined, this entry is also part of the unlock list. This unlock list is processed if the Telnet UNLOCK command is entered without any parameters. **Note:** Chip erase does not work for large chips because the BDI time-outs after 3 minutes. Use block erase.
wait	The wait time in ms is only used for the unlock mode. After starting the flash unlock, the BDI waits until it processes the next entry.
Example:	ERASE 0xff040000 ;erase sector 4 of flash ERASE 0xff060000 ;erase sector 6 of flash ERASE 0xff000000 CHIP ;erase whole chip(s) ERASE 0xff010000 UNLOCK 100 ;unlock, wait 100ms ERASE 0xff000000 0x10000 7 ; erase 7 sectors

Example for the AMD DB1100 board:

```
[FLASH]
WORKSPACE     0xA0001000;
CHIPTYPE      MIRRORX16       ;there is a MirrorBit flash in x16 mode
CHIPSIZE      0x800000        ;the chip is Am29LV640MH
BUSWIDTH      32              ;there are two chips building a 32-bit system
FILE          E:\temp\dump512k.bin
FORMAT        BIN 0xBFC80000;
ERASE         0xBFC80000;
ERASE         0xBFCA0000;
ERASE         0xBFCC0000;
ERASE         0xBFCE0000;
```

the above erase list maybe replaces with:

```
ERASE         0xBFC80000 0x20000 4 ;erase 4 sectors
```


Supported standard parallel NOR Flash Memories:

There are different flash algorithm supported. Almost all currently available parallel NOR flash memories can be programmed with one of these algorithm. The flash type selects the appropriate algorithm and gives additional information about the used flash.

On our web site (www.abatron.ch -> Debugger Support -> GNU Support -> Flash Support) there is a PDF document available that shows the supported parallel NOR flash memories.

Some newer Spansion MirrorBit flashes cannot be programmed with the MIRRORX16 algorithm because of the used unlock address offset. Use S29M32X16 for these flashes.

The AMD and AT49 algorithm are almost the same. The only difference is, that the AT49 algorithm does not check for the AMD status bit 5 (Exceeded Timing Limits).

Only the AMD and AT49 algorithm support chip erase. Block erase is only supported with the AT49 algorithm. If the algorithm does not support the selected mode, sector erase is performed. If the chip does not support the selected mode, erasing will fail. The erase command sequence is different only in the 6th write cycle. Depending on the selected mode, the following data is written in this cycle (see also flash data sheets): 0x10 for chip erase, 0x30 for sector erase, 0x50 for block erase.
To speed up programming of Intel Strata Flash and AMD MirrorBit Flash, an additional algorithm is implemented that makes use of the write buffer. The Strata algorithm needs a workspace, otherwise the standard Intel algorithm is used.

Note:
Some Intel flash chips (e.g. 28F800C3, 28F160C3, 28F320C3) power-up with all blocks in locked state. In order to erase/program those flash chips, use the init list to unlock the appropriate blocks:

```
WM16   0xFFF00000   0x0060       unlock block 0
WM16   0xFFF00000   0x00D0
WM16   0xFFF10000   0x0060       unlock block 1
WM16   0xFFF10000   0x00D0
       ....
WM16   0xFFF00000   0xFFFF       select read mode
```

or use the Telnet "unlock" command:

```
UNLOCK [<addr> [<delay>]]
```

addr	This is the address of the sector (block) to unlock
delay	A delay time in milliseconds the BDI waits after sending the unlock command to the flash. For example, clearing all lock-bits of an Intel J3 Strata flash takes up to 0.7 seconds.

If "unlock" is used without any parameter, all sectors in the erase list with the UNLOCK option are processed.

To clear all lock-bits of an Intel J3 Strata flash use for example:

```
BDI> unlock 0xFF000000 1000
```

To erase or unlock multiple, continuous flash sectors (blocks) of the same size, the following Telnet commands can be used:

```
ERASE  <addr> <step> <count>
UNLOCK <addr> <step> <count>
```

addr	This is the address of the first sector to erase or unlock.
step	This value is added to the last used address in order to get to the next sector. In other words, this is the size of one sector in bytes.
count	The number of sectors to erase or unlock.

The following example unlocks all 256 sectors of an Intel Strata flash (28F256K3) that is mapped to 0x00000000. In case there are two flash chips to get a 32bit system, double the "step" parameter.

```
BDI> unlock 0x00000000 0x20000 256
```

3.2.5 Part [REGS]

In order to make it easier to access target registers via the Telnet interface, the BDI can read in a register definition file. In this file, the user defines a name for the register and how the BDI should access it (e.g. as memory mapped, memory mapped with offset, ...). The name of the register definition file and information for different registers type has to be defined in the configuration file.
The register name, type, address/offset/number and size are defined in a separate register definition file. This way, you can create one register definition file for a specific target processor that can be used for all possible positions of the internal memory map. You only have to change one entry in the configuration file.

An entry in the register definition file has the following syntax:

```
name   type   addr   size
```

name	The name of the register (max. 12 characters)
type	The register type GPR General purpose register CP0 Coprocessor 0 register CP1 Coprocessor 1 register CP2 Coprocessor 2 register CTR XLP Control register MM Absolute direct memory mapped register DMM1...DMM4 Relative direct memory mapped register IMM1...IMM4 Indirect memory mapped register
addr	The address, offset or number of the register
size	The size (8, 16, 32, 64) of the register

The following entries are supported in the [REGS] part of the configuration file:

FILE filename	The name of the register definition file. This name is used to access the file via TFTP. The file is loaded once during BDI startup. filename the filename including the full path Example: FILE C:\bdi\regs\reg5kc.def
DMMn base	This defines the base address of direct memory mapped registers. This base address is added to the individual offset of the register. base the base address Example: DMM1 0xB8000000
IMMn addr data	This defines the addresses of the memory mapped address and data registers of indirect memory mapped registers. The address of a IMMn register is first written to "addr" and then the register value is access using "data" as address. addr the address of the Address register data the address of the Data register Example: DMM1 0x04700000

Example for a register definition (MIPS 5Kc):

Entry in the configuration file:

```
[REGS]
DMM1   0xFF300000      ;DSU base address
DMM2   0xBF000000      ;Memory mapped registers
FILE   E:\cygwin\home\bdidemo\mips64\reg5kc.def
```

The register definition file:

```
;name          type   addr          size
;-------------------------------------------
;
;
; CP0 Registers
;
index          CP0     0          32
random         CP0     1          32
elo0           CP0     2          32
elo1           CP0     3          32
context        CP0     4          64
pmask          CP0     5          32
wired          CP0     6          32
bad            CP0     8          64
ehi            CP0     10         64
xcontext       CP0     20         64
;
count          CP0     9          32
compare        CP0     11         32
status         CP0     12         32
cause          CP0     13         32
epc            CP0     14         64
prid           CP0     15         32
config         CP0     0x010      32
config1        CP0     0x110      32
watchlo        CP0     18         64
watchhi        CP0     19         32
debug          CP0     23         32
depc           CP0     24         64
taglo          CP0     0x01c      32
datalo         CP0     0x11c      64
taghi          CP0     0x01d      32
datalo         CP0     0x11d      32
eepc           CP0     30         64
desave         CP0     31         64
               ...
;
; DSU Registers
;
dcr            DMM1    0x0000
ibs            DMM1    0x1000
dbs            DMM1    0x2000
;
iba0           DMM1    0x1100
ibm0           DMM1    0x1108
ibasid0        DMM1    0x1110
               ....
;
```


3.3 Debugging with GDB

Because the target agent runs within BDI, no debug support has to be linked to your application. There is also no need for any BDI specific changes in the application sources. Your application must be fully linked because no dynamic loading is supported.

3.3.1 Target setup

Target initialization may be done at two places. First with the BDI configuration file, second within the application. The setup in the configuration file must at least enable access to the target memory where the application will be loaded. Disable the watchdog and setting the CPU clock rate should also be done with the BDI configuration file. Application specific initializations like setting the timer rate are best located in the application startup sequence.

3.3.2 Connecting to the target

As soon as the target comes out of reset, BDI initializes it and loads your application code. If RUN is selected, the application is immediately started, otherwise only the target PC is set. BDI now waits for GDB request from the debugger running on the host.

After starting the debugger, it must be connected to the remote target. This can be done with the following command at the GDB prompt:

```
(gdb)target remote bdi3000:2001
```

bdi3000	This stands for an IP address. The HOST file must have an appropriate entry. You may also use an IP address in the form xxx.xxx.xxx.xxx
2001	This is the TCP port used to communicate with the BDI

If not already suspended, this stops the execution of application code and the target CPU changes to background debug mode.

Remember, every time the application is suspended, the target CPU is freezed. During this time no hardware interrupts will be processed.

Note: For convenience, the GDB detach command triggers a target reset sequence in the BDI.

```
(gdb)...
(gdb)detach
... Wait until BDI has reseted the target and reloaded the image
(gdb)target remote bdi3000:2001
```

3.3.3 Breakpoint Handling

The BDI supports the GDB Z-packet to set breakpoints (watchpoints). For software breakpoints, the BDI replaces code with a SDBBP instruction. When breakpoint mode HARD is selected, the BDI sets an appropriate hardware breakpoint.

User controlled hardware breakpoints:
The MIPS processor has special watchpoint / breakpoint hardware integrated. Normally the BDI controls this hardware in response to Telnet commands (BI, BDx) or when breakpoint mode HARD is selected. Via the Telnet commands BI and BDx, you cannot access all the features of the breakpoint hardware. Therefore the BDI assumes that the user will control / setup this breakpoint hardware as soon as an address in the range 0xFF300000 - 0xFF3FFFFF is written to. This way the debugger or the user via Telnet has full access to all features of this watchpoint / breakpoint hardware. A hardware breakpoint set via BI or BD gives control back to the BDI.

3.3.4 GDB monitor command

The BDI supports the GDB "monitor" command. Telnet commands are executed and the Telnet output is returned to GDB. This way you can for example switch the BDI breakpoint mode from within your GDB session.

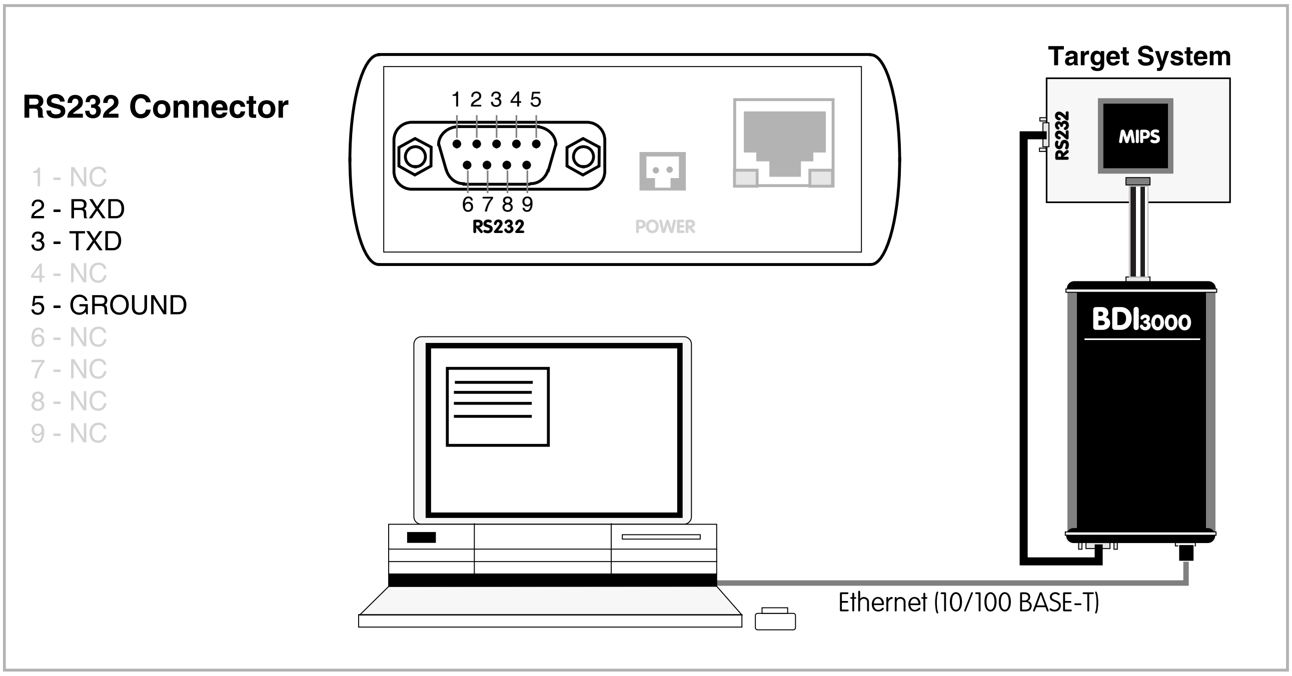
```
(gdb) target remote bdi3000:2001
Remote debugging using bdi3000:2001
0x10b2 in start ()
(gdb) mon break
Breakpoint mode is SOFT
(gdb) mon break hard

(gdb) mon break
Breakpoint mode is HARD
(gdb)
```

3.3.5 GDB remote address size

The BDI supports 32 bit and 64 bit addresses in a GDB remote protocol frame. The 32 bit addresses are sign-extended to build the required 64 bit addresses.

3.3.6 Target serial I/O via BDI

A RS232 port of the target can be connected to the RS232 port of the BDI3000. This way it is possible to access the target's serial I/O via a TCP/IP channel. For example, you can connect a Telnet session to the appropriate BDI3000 port. Connecting GDB to a GDB server (stub) running on the target should also be possible.



The configuration parameter "SIO" is used to enable this serial I/O routing.
The used framing parameters are 8 data, 1 stop and not parity.

```
[TARGET]
....
SIO    7      9600          ;Enable SIO via TCP port 7 at 9600 baud
```

Warning!!!
Once SIO is enabled, connecting with the setup tool to update the firmware will fail. In this case either disable SIO first or disconnect the BDI from the LAN while updating the firmware.

3.4 Telnet Interface

A Telnet server is integrated within the BDI. The Telnet channel is used by the BDI to output error messages and other information. Also some basic debug commands can be executed.

Telnet Debug features:

- Display and modify memory locations
- Display and modify general and special purpose registers
- Single step a code sequence
- Set hardware breakpoints
- Load a code file from any host
- Start / Stop program execution
- Programming and Erasing Flash memory

During debugging with GDB, the Telnet is mainly used to reboot the target (generate a hardware reset and reload the application code). It may be also useful during the first installation of the bdiGDB system or in case of special debug needs.

Multiple commands separated by a semicolon can be entered on one line.

Example of a Telnet session:

```
- TARGET: waiting for target Vcc
- TARGET: waiting for target Vcc
- TARGET: waiting for target Vcc
- TARGET: processing user reset request
- TARGET: reseting target passed
- TARGET: processing target startup ....
- TARGET: processing target startup passed
BDI>info
    Target state      : debug mode
    Debug entry cause : JTAG break request
    Current PC        : 0xffffffff80025bf8
    Current SR        : 0x00002c00
    Current LR  (r31) : 0xffffffff80025aa4
    Current SP  (r29) : 0xffffffff8008ede8
BDI>mdd 0xbfc00000
ffffffffbfc00000 : 1000000500000000 0000000000000000  ................
ffffffffbfc00010 : 0001052000000000 0000000040809000  ... ........@...
ffffffffbfc00020 : 0000000000000000 0000000000000000  ................
ffffffffbfc00030 : 4080980000000000 0000000000000000  @...............
ffffffffbfc00040 : 401a600000000000 0000000000000000  @.`.............
              .....................
```

Notes:
The DUMP command uses TFTP to write a binary image to a host file. Writing via TFTP on a Linux/Unix system is only possible if the file already exists and has public write access. Use "man tftpd" to get more information about the TFTP server on your host.

The Telnet commands:

```
"MDD   [<address>] [<count>]  display double (64bit)",
"MDW   [<address>] [<count>]  display word (32bit)",
"MDH   [<address>] [<count>]  display half word (16bit)",
"MDB   [<address>] [<count>]  display byte (8bit)",
"DUMP  <addr> <size> [<file>] dump target memory to a file",
"MMD   <addr> <value> [<cnt>] modify doubles(s) (64bit)",
"MMW   <addr> <value> [<cnt>] modify word(s) (32bit)",
"MMH   <addr> <value> [<cnt>] modify half word(s) (16bit)",
"MMB   <addr> <value> [<cnt>] modify byte(s) (8bit)",
"MT    <addr> <count>         memory test",
"MC    [<address>] [<count>]  calculates a checksum over a memory range",
"MV                           verifies the last calculated checksum",

"XMDD  [<address>] [<count>]  XLP phys: display double (64bit) ",
"XMDW  [<address>] [<count>]  XLP phys: display word (32bit)",
"XMDH  [<address>] [<count>]  XLP phys: display half word (16bit)",
"XMDB  [<address>] [<count>]  XLP phys: display byte (8bit)",
"XMMD  <addr> <value> [<cnt>] XLP phys: modify doubles(s) (64bit)",
"XMMW  <addr> <value> [<cnt>] XLP phys: modify word(s) (32bit)",
"XMMH  <addr> <value> [<cnt>] XLP phys: modify half word(s) (16bit)",
"XMMB  <addr> <value> [<cnt>] XLP phys: modify byte(s) (8bit)",

"RD    [<name>]               display general purpose or user defined register",
"RDUMP [<file>]               dump all user defined register to a file",
"RDFP                         display FP  registers",
"RDCP0 <number>               display CP0 register",
"RM    {<nbr>|<name>} <value> modify general purpose or user defined register",
"RMCP0 <number>   <value>     modify CP0 register",

"TLB   <from> [<to>]          display TLB entry",
"DTAG  <from> [<to>]          display data cache tag",
"ITAG  <from> [<to>]          display instruction cache tag",
"DFLUSH [<addr> [<size>]]     flush data cache",
"IFLUSH [<addr> [<size>]]     invalidate instruction cache",

"EXEC  <opcode>               execute an instruction",
"RGPR  <regnum>               read from core GPR",
"WGPR  <regnum> <value>       write to  core GPR (don't modify r1 and r30)",
"SYNC                         check for exception and restore debug PC",

"RESET  [HALT | RUN [time]]   reset the target system, change startup mode",
"BREAK  [SOFT | HARD]         display or set current breakpoint mode",
"GO     [<pc>]                set PC and start current core",
"CONT   <cores> [<cores>]     restart multiple cores (<cores> = core bit map)",
"                                 <core#63...core#0> [<core#127...core#64>]",
"TI     [<pc>]                trace on instuction (single step)",
"HALT   [<cores> [<cores>]]   force core(s) to debug mode (<cores> = core bit map)",
"FREEZE [<cores> [<cores>]]   XLP: force core(s) to debug mode via ECR2 break bit",
"BI       <addr> [<mask>]     set instruction breakpoint (<mask> = address mask)",
"BD [R|W] <addr> [<mask>]     set data breakpoint (<mask> = address mask)",
"CI [<id>]                    clear instruction breakpoint(s) of current core",
"CD [<id>]                    clear data breakpoint(s) of current core",
"CLEAR  [<cores> [<cores>]]   clear all breakpoints of all or selected cores",
"SELECT  <core>               change the current core",
"INFO                         display information about the current core",
"STATE                        display information about all cores",
```

```
"LOAD   [<offset>] [<file> [<format>]] load program file to target memory",
"VERIFY [<offset>] [<file> [<format>]] verify a program file to target memory",
"PROG   [<offset>] [<file> [<format>]] program flash memory",
"                                       <format> : SREC or BIN or AOUT or ELF",
"ERASE  [<address> [<mode>]]  erase a flash memory sector, chip or block",
"                   <mode>  : CHIP, BLOCK or SECTOR (default is sector)",
"ERASE  <addr> <step> <count> erase multiple flash sectors",
"UNLOCK [<addr> [<delay>]]    unlock a flash sector",
"UNLOCK <addr> <step> <count> unlock multiple flash sectors",
"FLASH  <type> <size> <bus>   change flash configuration",

"DELAY  <ms>                  delay for a number of milliseconds",
"HOST   <ip>                  change IP address of program file host",
"PROMPT <string>              defines a new prompt string",
"CONFIG                       display or update BDI configuration",
"CONFIG <file> [<hostIP> [<bdiIP> [<gateway> [<mask>]]]]",
"HELP                         display command list",
"BOOT                         reset the BDI and reload the configuration",
"QUIT                         terminate the Telnet session"
```

The following commands allow to execute instructions on the target processor:

```
"EXEC  <opcode>               execute an instruction",
"RGPR  <regnum>               read from core GPR",
"WGPR  <regnum> <value>       write to  core GPR (don't modify r1 and r30)",
"SYNC                         check for exceptions and restore debug PC",
```

At the end of a code sequence or after many (say 1000) stuffed instruction a "sync" command should be executed. This will set the debug PC back to a BDI defined start value.

Following a simple instruction sequence:

```
BDI>rgpr 6
ffffffff9fc44940
BDI>exec 0x24061234        (addiu r6,r0,0x1234)
BDI>rgpr 6
0000000000001234
BDI>exec 0x24c60005        (addiu r6,r6,5)
BDI>rgpr 6
0000000000001239
BDI>sync
```

3.5 Multi-Core Support

The BDI supports up to 96 cores connected to the same JTAG scan chain.Via Telnet you can switch between the cores with the command "select <0..95>". In the configuration file, simply begin the line with the appropriate core number. If there is no #n in front of a line, the BDI assumes core #0.

Up to 32 cores can be prepared for GDB debugging. To select a core for GDB debugging define an approriate debug port number. Only core #0 has a default debug port of 2001 assigned. For every selected core you can start its own individual GDB session.

The following example defines 3 cores on a scan chain. For a complete example, look at the configuration examples.

```
[TARGET]
; common parameters
POWERUP     2000                ;power-up delay 2 seconds
JTAGCLOCK   0                   ;use 16 MHz JTAG clock
;
;=====================================================
; !!!! define the cores numbers without any holes !!!!
;=====================================================
;
; Core#0 parameters (active core after reset)
#0 CPUTYPE     CNMIPS           ;the used target CPU type
#0 ENDIAN      BIG              ;target is big endian
#0 JTAGDELAY   5                ;40 TCK's access delay
#0 STARTUP     STOP 5000        ;STOP mode is used to let the monitor init the system
#0 WORKSPACE   0xA0000080       ;workspace in target RAM for fast download
#0 BREAKMODE   HARD             ;SOFT or HARD
#0 SCANPRED    15  75           ;select last core in scan chain
#0 SCANSUCC    0   0
;
; Core#1 parameters
#1 CPUTYPE     CNMIPS
#1 ENDIAN      BIG
#1 JTAGDELAY   5
#1 STARTUP     WAIT             ;CPU is held in reset
#1 BREAKMODE   HARD
#1 SCANPRED    14  70
#1 SCANSUCC    1   5
;
; Core#2 parameters
#2 CPUTYPE     CNMIPS
#2 ENDIAN      BIG
#2 JTAGDELAY   5
#2 STARTUP     WAIT             ;CPU is held in reset
#2 BREAKMODE   HARD
#2 SCANPRED    13  65
#2 SCANSUCC    2   10


[HOST]
#0  PROMPT      cnMIPS#0>
#1  PROMPT      cnMIPS#1>
#2  PROMPT      cnMIPS#2>
;
#0  DEBUGPORT   2001
#1  DEBUGPORT   2002
#2  DEBUGPORT   2003
```

Multi-Core related Telnet commands:

STATE	Display information about all cores.	
SELECT <core>	Change the current Telnet core	
CONT <cores>	Restart selected cores	
	<cores>	core bit map <core#63...core#0> [<core#127...core#64>]
	Example:	cont 0x0d ; start core #0, #2, #3
HALT [<cores>]	Force core(s) to debug mode. If there is no <cores> parameter, the current selected Telnet core is forced to debug mode.	
	<cores>	core bit map <core#63...core#0> [<core#127...core#64>]
	Example:	halt 0xff ; halt all 8 cores #0...#7 halt 0xffffffffffffffff 0xffff ; halt 80 cores #0...#79
CLEAR [<cores>]	Clear all breakpoints of all or selected cores. If there is no <cores> parameter, then all hardware breakpoints in all cores are cleared. To clear the breakpoints in the currently selected core use "ci" and "cd".	
	<cores>	core bit map <core#63...core#0> [<core#127...core#64>]
	Example:	clear 0x0018 ; Clear the breakpoints in core 3 and 4

4 Specifications

Operating Voltage Limiting	5 VDC ± 0.25 V
Power Supply Current	typ. 500 mA max. 1000 mA
RS232 Interface: Baud Rates	9’600,19’200, 38’400, 57’600,115’200
Data Bits	8
Parity Bits	none
Stop Bits	1
Network Interface	10/100 BASE-T
BDM/JTAG clock	up to 32 MHz
Supported target voltage	1.2 – 5.0 V
Operating Temperature	+ 5 °C ... +60 °C
Storage Temperature	-20 °C ... +65 °C
Relative Humidity (noncondensing)	<90 %rF
Size	160 x 85 x 35 mm
Weight (without cables)	280 g
Host Cable length (RS232)	2.5 m
Electromagnetic Compatibility	CE compliant
Restriction of Hazardous Substances	RoHS 2002/95/EC compliant

Specifications subject to change without notice

5 Environmental notice

Disposal of the equipment must be carried out at a designated disposal site.

6 Declaration of Conformity (CE)

CE

DECLARATION OF CONFORMITY

This declaration is valid for following product:

Type of device: BDM/JTAG Interface
Product name: BDI3000

The signing authorities state, that the above mentioned equipment meets the requirements for emission and immunity according to

EMC Directive 89/336/EEC

The evaluation procedure of conformity was assured according to the following standards:

IEC 61000-6-2: 1999, mod. EN61000-6-2: 2001
IEC 61000-6-3: 1996, mod. EN61000-6-2: 2001

This declaration of conformity is based on the test report no. E1087-05-7a of Quinel, Zug, Swiss Testing Service, accreditation no. STS 037

Manufacturer:

ABATRON AG
Lettenstrasse 9
CH-6343 Rotkreuz

Authority:

Max Vock
Marketing Director

Ruedi Dummermuth
Technical Director

Rotkreuz, 7/18/2007

7 Abatron Warranty and Support Terms

7.1 Hardware

ABATRON Switzerland warrants that the Hardware shall be free from defects in material and workmanship for a period of 3 years following the date of purchase when used under normal conditions. Failure in handling which leads to defects or any self-made repair attempts are not covered under this warranty. In the event of notification within the warranty period of defects in material or workmanship, ABATRON will repair or replace the defective hardware. The customer must contact the distributor or Abatron for a RMA number prior to returning.

7.2 Software

License

Against payment of a license fee the client receives a usage license for this software product, which is not exclusive and cannot be transferred.

Copies

The client is entitled to make copies according to the number of licenses purchased. Copies exceeding this number are allowed for storage purposes as a replacement for defective storage mediums.

Update and Support

The agreement includes free software maintenance (update and support) for one year from date of purchase. After this period the client may purchase software maintenance for an additional year.

7.3 Warranty and Disclaimer

ABATRON AND ITS SUPPLIERS HEREBY DISCLAIMS AND EXCLUDES, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT.

7.4 Limitation of Liability

IN NO EVENT SHALL ABATRON OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING, WITHOUT LIMITATION, ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE HARDWARE AND/OR SOFTWARE, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, BUSINESS, DATA, GOODWILL, OR ANTICIPATED SAVINGS, EVEN IF ADVISED OF THE POSSIBILITY OF THOSE DAMAGES.

The hardware and software product with all its parts, copyrights and any other rights remain in possession of ABATRON. Any dispute, which may arise in connection with the present agreement shall be submitted to Swiss Law in the Court of Zug (Switzerland) to which both parties hereby assign competence.

Appendices

A Troubleshooting

Problem
The firmware can not be loaded.

Possible reasons

- The BDI is not correctly connected with the Host (see chapter 2).
- A wrong communication port is selected (Com 1...Com 4).
- The BDI is not powered up

Problem
No working with the target system (loading firmware is okay).

Possible reasons

- Wrong pin assignment (BDM/JTAG connector) of the target system (see chapter 2).
- Target system initialization is not correctly –> enter an appropriate target initialization list.
- An incorrect IP address was entered (BDI3000 configuration)
- BDM/JTAG signals from the target system are not correctly (short-circuit, break, ...).
- The target system is damaged.

Problem
Network processes do not function (loading the firmware was successful)

Possible reasons

- The BDI3000 is not connected or not correctly connected to the network (LAN cable or media converter)
- An incorrect IP address was entered (BDI3000 configuration)

B Maintenance

The BDI needs no special maintenance. Clean the housing with a mild detergent only. Solvents such as gasoline may damage it.

C Trademarks

All trademarks are property of their respective holders.

