

*bd*GDB

JTAG debug interface for GNU Debugger

MIPS32



User Manual

Manual Version 1.18 for BDI2000

1 Introduction	4
1.1 BDI2000.....	4
1.2 BDI Configuration	5
2 Installation	6
2.1 Connecting the BDI2000 to Target	6
2.1.1 Changing Target Processor Type	8
2.2 Connecting the BDI2000 to Power Supply	9
2.2.1 External Power Supply.....	9
2.2.2 Power Supply from Target System	10
2.3 Status LED «MODE».....	11
2.4 Connecting the BDI2000 to Host	12
2.4.1 Serial line communication	12
2.4.2 Ethernet communication	13
2.5 Installation of the Configuration Software	14
2.5.1 Configuration with a Linux / Unix host.....	15
2.5.2 Configuration with a Windows host.....	17
2.5.3 Recover procedure.....	18
2.6 Testing the BDI2000 to host connection.....	19
2.7 TFTP server for Windows NT	19
3 Using bdiGDB	20
3.1 Principle of operation	20
3.2 Configuration File.....	21
3.2.1 Part [INIT].....	22
3.2.2 Part [TARGET]	24
3.2.3 Part [HOST].....	29
3.2.4 Part [FLASH]	31
3.2.5 Part [REGS]	35
3.3 Debugging with GDB	37
3.3.1 Target setup	37
3.3.2 Connecting to the target.....	37
3.3.3 Breakpoint Handling.....	38
3.3.4 GDB monitor command.....	38
3.3.5 Target serial I/O via BDI.....	39
3.3.6 Embedded Linux MMU Support	40
3.4 Telnet Interface.....	42
3.5 Multi-Core Support.....	45
3.5.1 MIPS 34K MT ASE Support	46
4 Specifications	48
5 Environmental notice.....	49
6 Declaration of Conformity (CE).....	49
7 Warranty and Support Terms.....	50
7.1 Hardware	50
7.2 Software	50
7.3 Warranty and Disclaimer	50
7.4 Limitation of Liability	50

7 Appendices

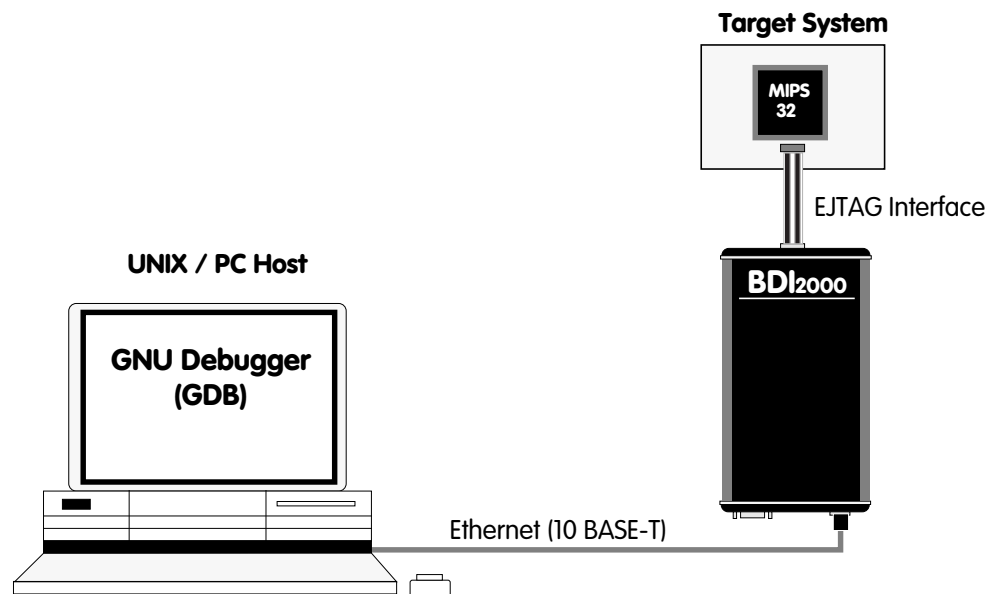
A Troubleshooting	51
B Maintenance	52
C Trademarks	54

1 Introduction

bdiGDB enhances the GNU debugger (GDB), with EJTAG debugging for MIPS32 based targets. With the builtin Ethernet interface you get a very fast download speed of up to 100 kBytes/sec. 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 BDI2000 interface is connected between the host and the target:



1.1 BDI2000

The BDI2000 is the main part of the bdiGDB system. This small box implements the interface between the EJTAG pins of the target CPU and a 10Base-T ethernet connector. The firmware and the programmable logic of the BDI2000 can be updated by the user with a simple setup tool. The BDI2000 supports 1.8 – 5.0 Volts target systems (3.0 – 5.0 Volts target systems with Rev. A/B).

1.2 BDI Configuration

As an initial setup, the IP address of the BDI2000, the IP address of the host with the configuration file and the name of the configuration file is stored within the flash of the BDI2000.

Every time the BDI2000 is powered on, it reads the configuration file via TFTP.

Following an example of a typical configuration file:

```
; bdiGDB configuration file for IDT79S334A board
; -----
;
[INIT]
; Setup Internal Bus
WM32 0xFFFFE200 0xAA82AAAA ;CPU Port Width Register, Flash 32bit
WM32 0xFFFFE204 0x3FFFFFFF ;CPU BTA Register
WM32 0xB8000000 0x3FFFFFFF ;BTA Register
WM32 0xB8000004 0x00000007 ;Address Latch Timing Register
;
WCP0 12 0x10010000 ;Setup Status Register, clear BEV
WCP0 13 0x00000000 ;Clear Cause Register
WCP0 16 0x00000003 ;Set kseg0 coherency
WM32 0xB8000730 0x00000000 ;Disable Watchdog Timer
;
; Init memory controller
WM32 0xB8000080 0x1FC00000 ;Memory Base Address Bank 0, Flash
WM32 0xB8000084 0xFFC00000 ;Memory Base Mask Bank 0, Flash
WM32 0xB8000088 0x04000000 ;Memory Base Address Bank 1, SRAM
WM32 0xB800008C 0xFFF00000 ;Memory Base Mask Bank 1, SRAM
WM32 0xB8000200 0x00002884 ;Memory Control Bank 0, Flash 32bit
WM32 0xB8000204 0x00002863 ;Memory Control Bank 1, SRAM
....
;
[TARGET]
JTAGCLOCK 1 ;use 8 MHz JTAG clock
CPUTYPE RC32300 ;the used target CPU type
ENDIAN LITTLE ;target is little endian
WORKSPACE 0xA0000080 ;workspace in target RAM for fast download
BREAKMODE SOFT ;SOFT or HARD, HARD uses hardware breakpoints
VECTOR CATCH ;catch unhandled exceptions
;
[HOST]
IP 151.120.25.115
FILE E:\cygnus\root\usr\demo\mips\vm linux
FORMAT ELF
LOAD MANUAL ;load code MANUAL or AUTO after reset
;
[FLASH]
WORKSPACE 0xa0000000 ;workspace in target RAM for fast programming algorithm
CHIPTYPE AM29F ;Flash type (AM29F | AM29BX8 | AM29BX16 | I28BX8 | I28BX16)
CHIPSIZE 0x80000 ;The size of one flash chip in bytes (e.g. AM29F040 = 0x80000)
BUSWIDTH 32 ;The width of the flash memory bus in bits (8 | 16 | 32)
FILE E:\cygnus\root\usr\demo\mips\loop_le.sss
ERASE 0xBFC00000 ;erase sector 0
;
[REGS]
DMM1 0xFF300000 ;DSU base address
DMM2 0xB8000000 ;Memory mapped registers
FILE E:\cygnus\root\usr\demo\mips\reg32334.def
```

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

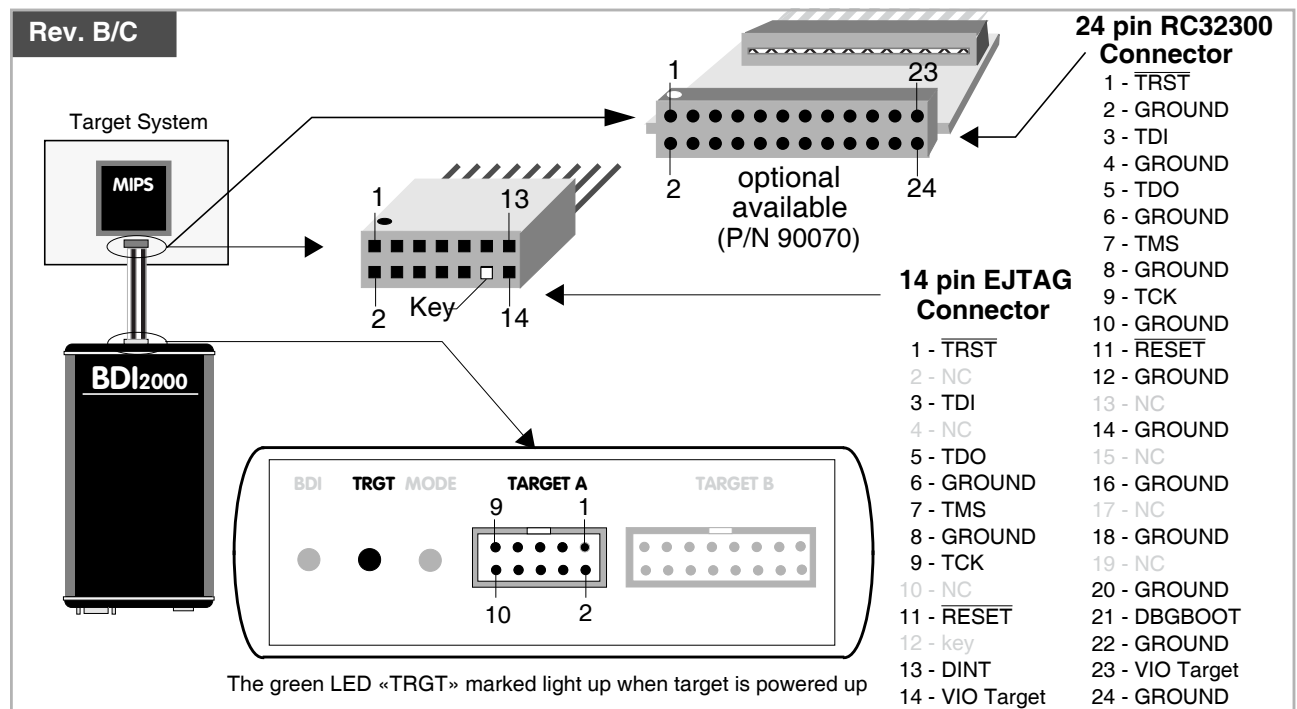
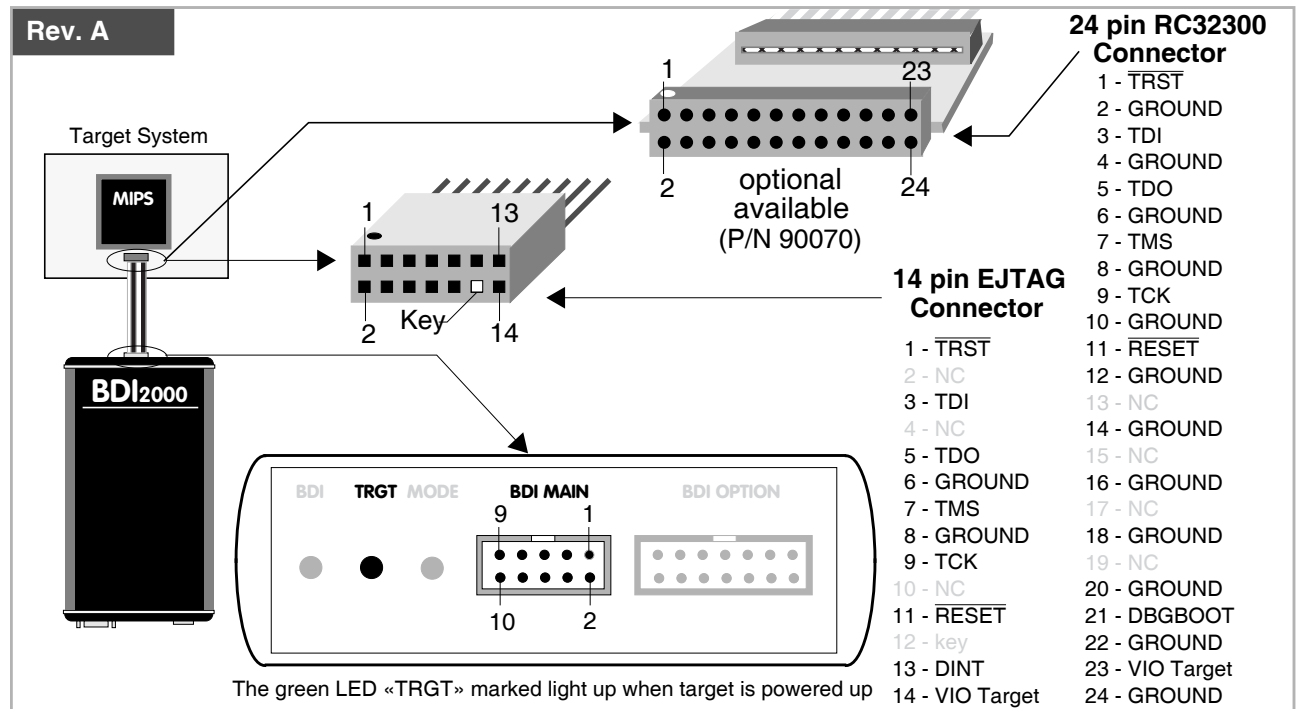
2 Installation

2.1 Connecting the BDI2000 to Target

The cables to the target system are designed for the IDT RC32300 Development Boards (optional available: Part# 90070) and for EJTAG 2.5 compatible boards (enclosed). In case where the target system has the same connector layout, the cable (14 pin or 24 pin) 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").



For BDI MAIN / TARGET A connector signals see table on next page.

BDI MAIN / TARGET A Connector Signals

Pin	Name	Description
1	DINT	EJTAG Debug Interrupt EJTAG 2.5: This output of the BDI2000 connects to the target DINT line. RC32300: This output of the BDI2000 connects to the target DebugBoot line.
2	$\overline{\text{TRST}}$	EJTAG Test Reset This output of the BDI2000 resets the JTAG TAP controller on the target.
3+5	GND	System Ground
4	TCK	EJTAG Test Clock This output of the BDI2000 connects to the target TCK line.
6	TMS	EJTAG Test Mode Select This output of the BDI2000 connects to the target TMS line.
7	$\overline{\text{RESET}}$	This open collector output of the BDI2000 is used to reset the target system.
8	TDI	EJTAG Test Data In This output of the BDI2000 connects to the target TDI line.
9	VIO Target	1.8 – 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. 3.0 – 5.0V with Rev. A/B : This input to the BDI2000 is used to detect if the target is powered up. If there is a current limiting resistor between this pin and the target Vdd, it should be 100 Ohm or less.
10	TDO	EJTAG Test Data Out This input to the BDI2000 connects to the target TDO line.

2.1.1 Changing Target Processor Type

Before you can use the BDI2000 with an other target processor type (e.g. ARM <--> MIPS), a new setup has to be done (see chapter 2.5). During this process the target cable must be disconnected from the target system. The BDI2000 needs to be supplied with 5 Volts via the BDI OPTION connector (Rev. A) or via the POWER connector (Rev. B/C). For more information see chapter 2.2.1 «External Power Supply»).



To avoid data line conflicts, the BDI2000 must be disconnected from the target system while programming the logic for an other target CPU.

2.2 Connecting the BDI2000 to Power Supply

2.2.1 External Power Supply

The BDI2000 needs to be supplied with 5 Volts (max. 1A) via the BDI OPTION connector (Rev. A) or via POWER connector (Rev. B/C). The available power supply from Abatron (option) or the enclosed power cable can be directly connected. In order to ensure reliable operation of the BDI2000, keep the power supply cable as short as possible.



For error-free operation, the power supply to the BDI2000 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.**

Rev. A

The green LED «BDI» marked light up when 5V power is connected to the BDI2000

BDI OPTION Connector

- 1 - NOT USED
- 2 - GROUND
- 3 - NOT USED
- 4 - GROUND
- 5 - NOT USED
- 6 - GROUND
- 7 - NOT USED
- 8 - GROUND
- 9 - NOT USED
- 10 - GROUND
- 11 - NOT USED
- 12 - Vcc (+5V)
- 13 - Vcc Target (+5V)
- 14 - Vcc (+5V)

Rev. B/C

The green LED «BDI» marked light up when 5V power is connected to the BDI2000

POWER Connector

- 1 - Vcc (+5V)
- 2 - VccTGT
- 3 - GROUND
- 4 - NOT USED

Please switch on the system in the following sequence:

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

2.2.2 Power Supply from Target System

The BDI2000 needs to be supplied with 5 Volts (max. 1A) via BDI MAIN target connector (Rev. A) or via TARGET A connector (Rev. B/C). This mode can only be used when the target system runs with 5V and the pin «Vcc Target» is able to deliver a current up to 1A@5V. For pin description and layout see chapter 2.1 «Connecting the BDI2000 to Target». Insert the enclosed Jumper as shown in figure below. **Please ensure that the jumper is inserted correctly.**



For error-free operation, the power supply to the BDI2000 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.**

Rev. A

The green LEDs «BDI» and «TRGT» marked light up when target is powered up and the jumper is inserted correctly

BDI OPTION Connector

- 1 - NOT USED
- 2 - GROUND
- 3 - NOT USED
- 4 - GROUND
- 5 - NOT USED
- 6 - GROUND
- 7 - NOT USED
- 8 - GROUND
- 9 - NOT USED
- 10 - GROUND
- 11 - NOT USED
- 12 - Vcc (+5V)
- 13 - Vcc Target (+5V)
- 14 - Vcc BDI2000 (+5V)

Rev. B/C

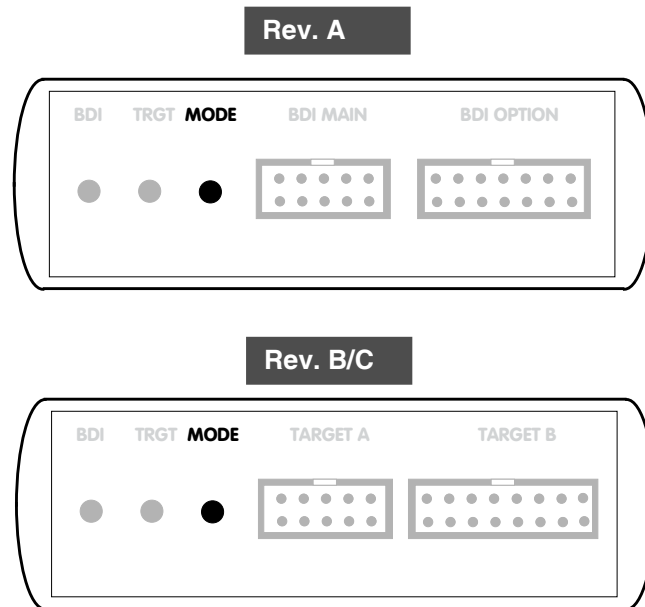
The green LEDs «BDI» and «TRGT» marked light up when target is powered up and the jumper is inserted correctly

POWER Connector

- 1 - Vcc BDI2000 (+5V)
- 2 - Vcc Target (+5V)
- 3 - GROUND
- 4 - NOT USED

2.3 Status LED «MODE»

The built in LED indicates the following BDI states:



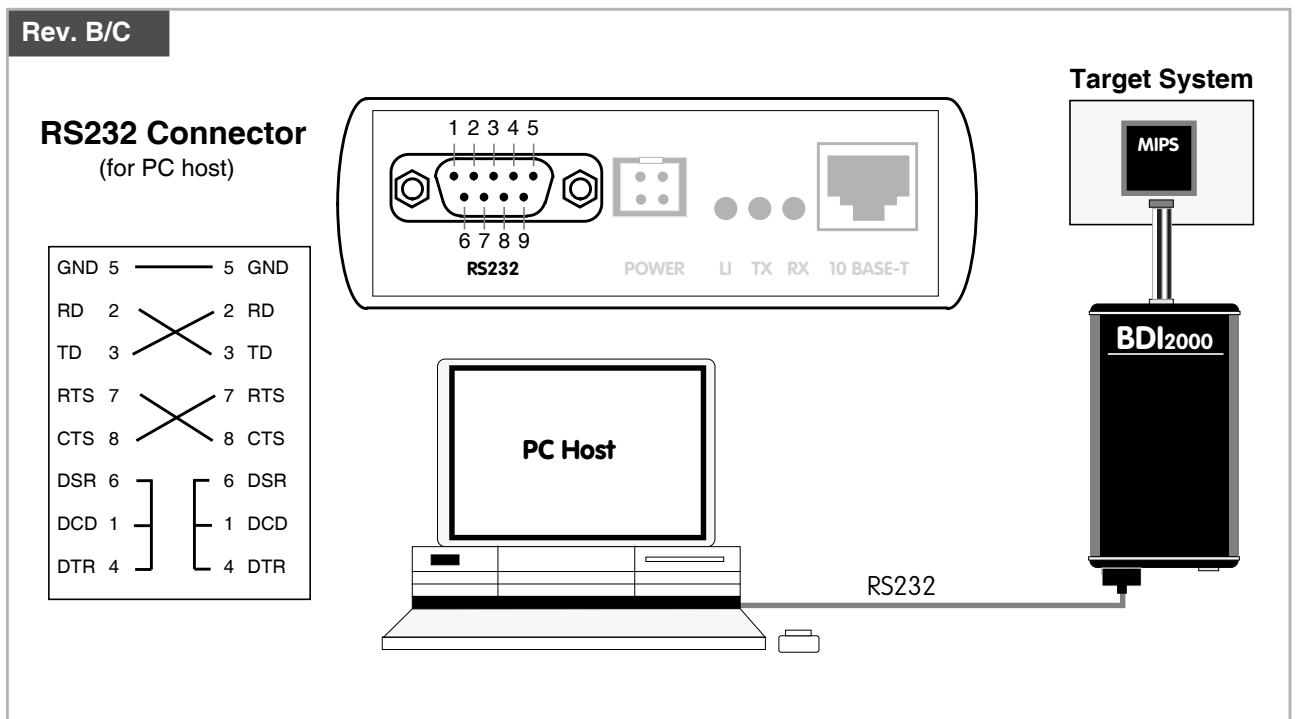
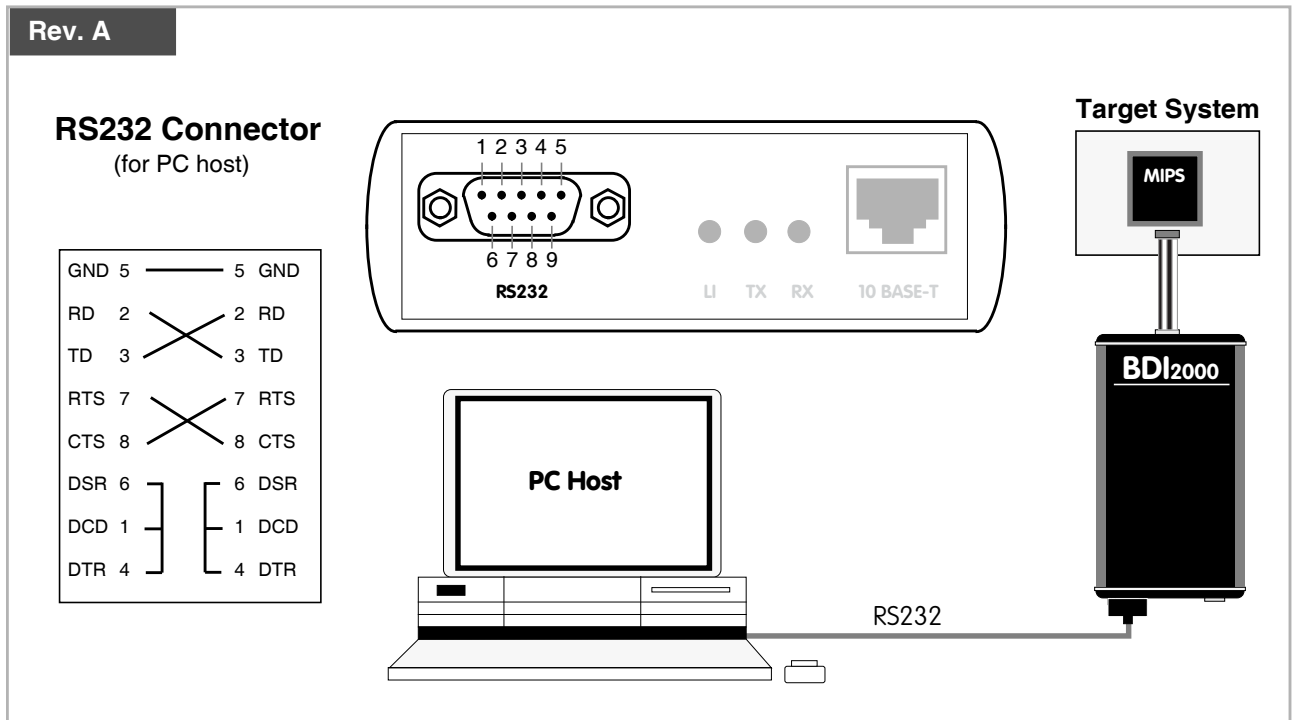
MODE LED	BDI STATES
OFF	The BDI is ready for use, the firmware is already loaded.
ON	The power supply for the BDI2000 is < 4.75VDC.
BLINK	The BDI «loader mode» is active (an invalid firmware is loaded or loading firmware is active).

2.4 Connecting the BDI2000 to Host

2.4.1 Serial line communication

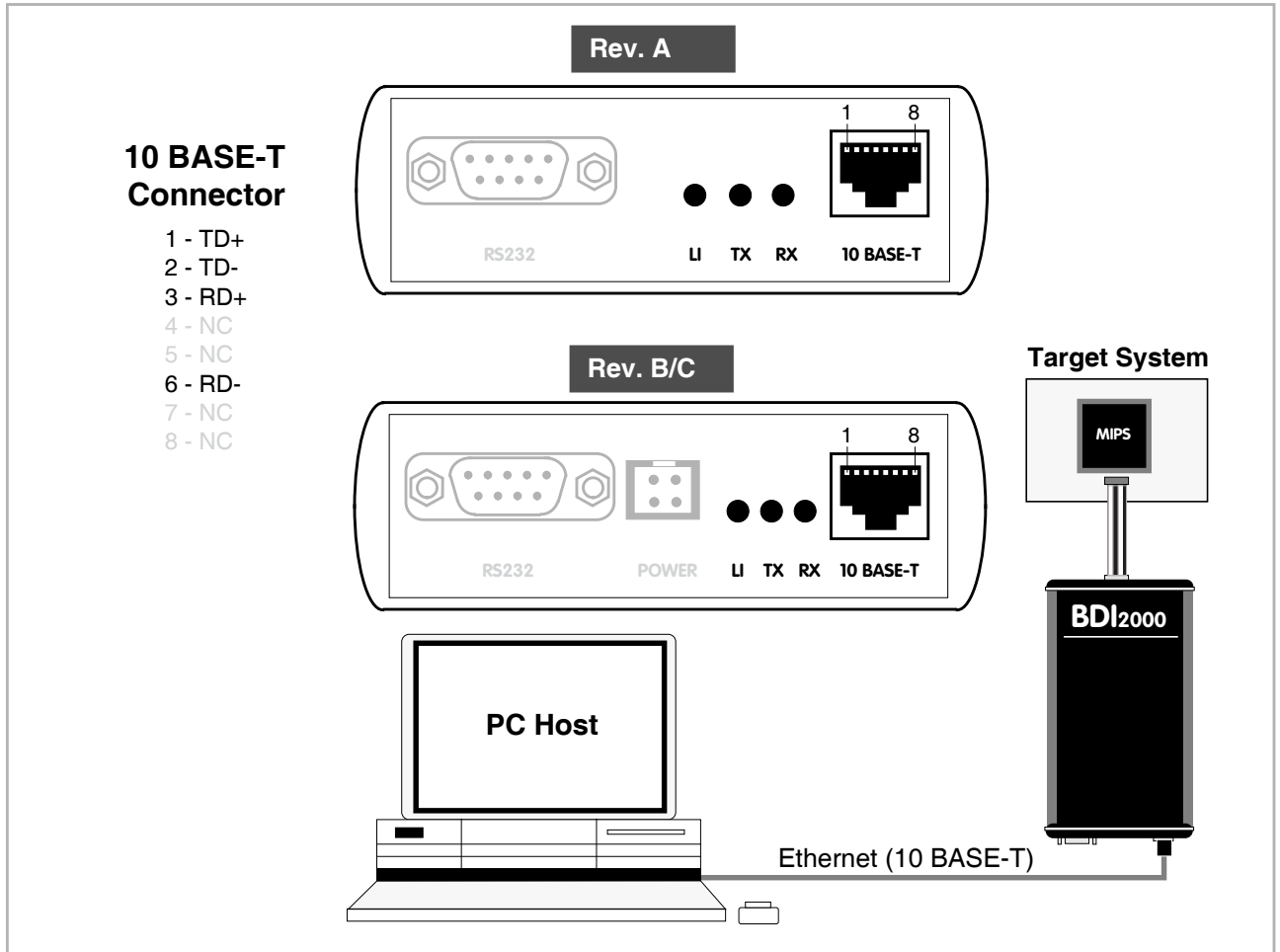
Serial line communication is only used for the initial configuration of the bdiGDB system.

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 BDI2000 has a built-in 10 BASE-T Ethernet interface (see figure below). Connect an UTP (Unshielded Twisted Pair) cable to the BDI2000. For thin Ethernet coaxial networks you can connect a commercially available media converter (BNC-->10 BASE-T) between your network and the BDI2000. Contact your network administrator if you have questions about the network.



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

LED	Name	Description
LI	Link	When this LED light is ON, data link is successful between the UTP port of the BDI2000 and the hub to which it is connected.
TX	Transmit	When this LED light BLINKS, data is being transmitted through the UTP port of the BDI2000
RX	Receive	When this LED light BLINKS, data is being received through the UTP port of the BDI2000

2.5 Installation of the Configuration Software

On the enclosed diskette you will find the BDI configuration software and the firmware required for the BDI2000. For Windows users there is also a TFTP server included.

The following files are on the diskette.

b20r4kgd.exe	Configuration program
b20r4kgd.xxx	Firmware for the BDI2000
r4kjed20.xxx	JEDEC file for the BDI2000 (Rev. A/B) logic device (CPLD)
r4kjed21.xxx	JEDEC file for the BDI2000 (Rev. C) logic device (CPLD)
ftpsrv.exe	TFTP server for Windows (WIN32 console application)
*.cfg	Configuration files
*.def	Register definition files
loop_le.sss	S-record file with a short little endian endless loop mapped to 0xBFC00000
loop_be.sss	S-record file with a short big endian endless loop mapped to 0xBFC00000
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 to load/update the BDI firmware/logic
Note: A new BDI has no firmware/logic loaded.
- Use the setup tool to transmit 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 simply 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# 93123457 ==>> 00-0C-01-93-12-34

2.5.1 Configuration with a Linux / Unix host

The firmware / logic update and the initial configuration of the BDI2000 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 BDI2000 must be disconnected from the target system while programming the logic for an other target CPU (see Chapter 2.1.1).

Following the steps to bring-up a new BDI2000:

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 bdisetup.o bdisetup.c
cc -O2 -c -o bdisetup.o bdisetup.c
cc -s bdisetup.o bdisetup.o bdisetup.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.

```
[root@LINUX_1 bdisetup]# ./bdisetup -v -p/dev/ttyS0 -b57
BDI Type : BDI2000 Rev.C (SN: 92152150)
Loader   : V1.05
Firmware : unknown
Logic    : 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   : ????????????????????
```

3. Load/Update the BDI firmware/logic:

With "bdisetup -u" the firmware is loaded and the CPLD within the BDI2000 is programmed. This configures the BDI for the target you are using. Based on the parameters -a and -t, the tool selects the correct firmware / logic files. If the firmware / logic files are in the same directory as the setup tool, there is no need to enter a -d parameter.

```
[root@LINUX_1 bdisetup]# ./bdisetup -u -p/dev/ttyS0 -b57 -aGDB -tMIPS
Connecting to BDI loader
Erasing CPLD
Programming firmware with ./b20r4kgd.100
Programming CPLD with ./r4kjed21.100
```

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 BDI2000. Ask your network administrator for assigning an IP address to this BDI2000. Every BDI2000 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).

```
[root@LINUX_1 bdisetup]# ./bdisetup -c -p/dev/ttyS0 -b57 \  
> -i151.120.25.101 \  
> -h151.120.25.118 \  
> -fs334a.cnf  
Connecting to BDI loader  
Writing network configuration  
Writing init list and mode  
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 flashing. 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.

```
[root@LINUX_1 bdisetup]# ./bdisetup -v -p/dev/ttyS0 -b57 -s  
BDI Type : BDI2000 Rev.C (SN: 92152150)  
Loader : V1.05  
Firmware : V1.00 bdiGDB for MIPS32  
Logic : V1.00 MIPS32  
MAC : 00-0c-01-92-15-21  
IP Addr : 151.120.25.101  
Subnet : 255.255.255.255  
Gateway : 255.255.255.255  
Host IP : 151.120.25.118  
Config : s334a.cnf
```

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

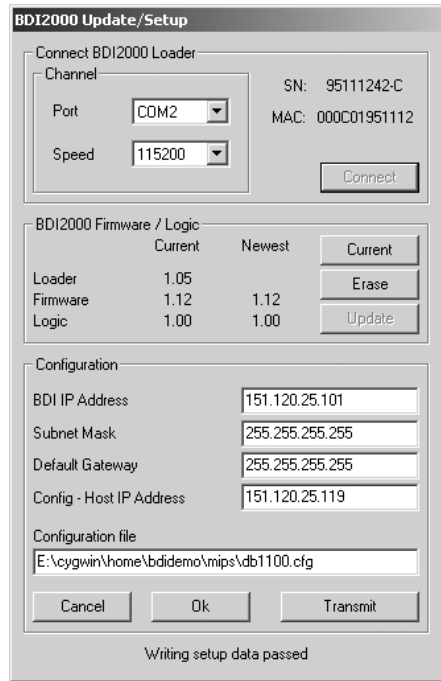
```
[root@LINUX_1 bdisetup]# telnet 151.120.25.101
```


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 BDI2000 must be disconnected from the target system while programming the logic for an other target CPU (see Chapter 2.1.1).



dialog box «BDI2000 Update/Setup»

Before you can use the BDI2000 together with the GNU debugger, you must store the initial configuration parameters in the BDI2000 flash memory. The following options allow you to do this:

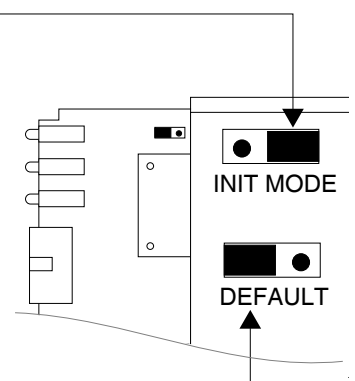
- Channel Select the communication port where the BDI2000 is connected during this setup session.
- Baudrate Select the baudrate used to communicate with the BDI2000 loader during this setup session.
- Connect Click on this button to establish a connection with the BDI2000 loader. Once connected, the BDI2000 remains in loader mode until it is restarted or this dialog box is closed.
- Current Press this button to read back the current loaded BDI2000 software and logic versions. The current loader, firmware and logic version will be displayed.
- Update This button is only active if there is a newer firmware or logic version present in the execution directory of the bdiGDB setup software. Press this button to write the new firmware and/or logic into the BDI2000 flash memory / programmable logic.

BDI IP Address	Enter the IP address for the BDI2000. 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 BDI2000. Every BDI2000 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 file is read by the setup tool or via TFTP.
Transmit	Click on this button to store the configuration in the BDI2000 flash memory.

2.5.3 Recover procedure

In rare instances you may not be able to load the firmware in spite of a correctly connected BDI (error of the previous firmware in the flash memory). **Before carrying out the following procedure, check the possibilities in Appendix «Troubleshooting».** In case you do not have any success with the tips there, do the following:

- Switch OFF the power supply for the BDI and open the unit as described in Appendix «Maintenance»
- Place the jumper in the «**INIT MODE**» position
- Connect the power cable or target cable if the BDI is powered from target system
- Switch ON the power supply for the BDI again and wait until the LED «MODE» blinks fast
- Turn the power supply OFF again
- Return the jumper to the «**DEFAULT**» position
- Reassemble the unit as described in Appendix «Maintenance»



2.6 Testing the BDI2000 to host connection

After the initial setup is done, you can test the communication between the host and the BDI2000. There is no need for a target configuration file and no TFTP server is needed on the host.

- If not already done, connect the bdiGDB system to the network.
- Power-up the BDI2000.
- Start a Telnet client on the host and connect to the BDI2000 (the IP address you entered during initial configuration).
- If everything is okay, a sign on message like «BDI Debugger for ARM» should be displayed in the Telnet window.

2.7 TFTP server for Windows NT

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.

<code>tftpsrv p</code>	Starts the TFTP server and enables protocol output
<code>tftpsrv p w</code>	Starts the TFTP server, enables protocol output and write accesses are allowed.
<code>tftpsrv dC:\tftp\</code>	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"

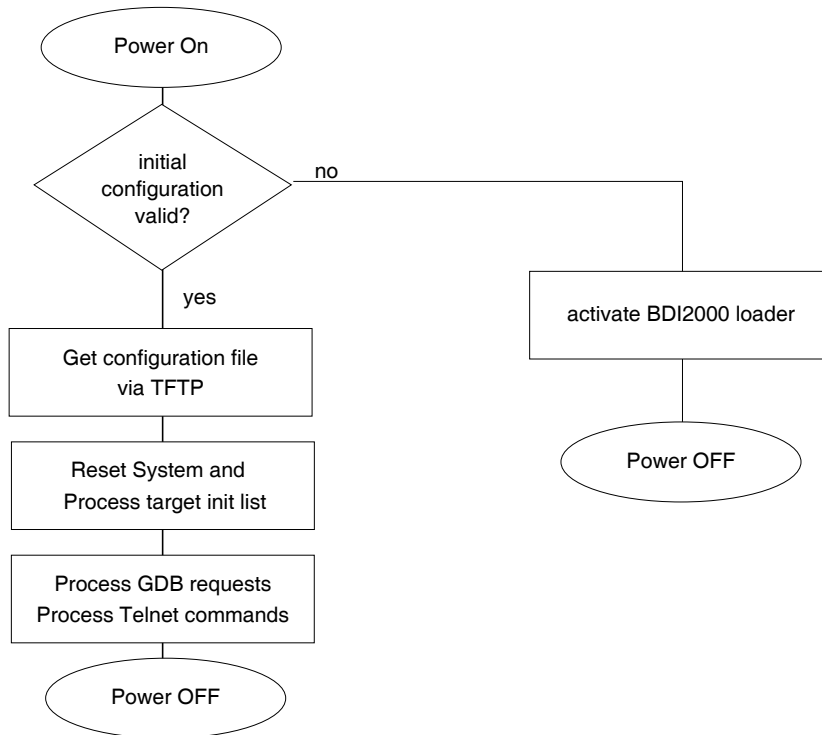
You may enter the TFTP server into the Startup group so the server is started every time you login.

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 BDI2000 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 for IDR RC32300 processors:

The debug boot function on IDT RC323000 processors does not work. Therefore the EJTAG debug interface can not always get control over the processor if there is no valid code in the boot ROM. If there is an empty boot flash, the BDI may need multiple reset sequences until it gets control over the processor. It is recommended to program at least a small endless loop into the boot flash. On the distribution diskette you will find the appropriate S-record files with this small loop code. One for little endian and one for big endian systems.

Also the hardware breakpoint logic inside the RC32300 does not always work as expected. It is highly recommended to use only BREAKMODE SOFT and STEPMODE SWBP. In cases where it is absolutely necessary to use hardware breakpoints (debugging ROM code) use the HWBP's very defensive. Do not set breakpoints following load/store instructions or following a branch with a load/store instruction in the branch delay slot. This is especially important if the code is cached.

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.

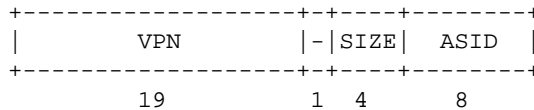
WGPR register value	Write value to the selected general purpose register. register the register number 0 .. 31 value the value to write into the register Example: WGPR 0 5
WREG name value	Write value to the selected register/memory by name name the case sensitive register name from the reg def file value the value to write to the register/memory Example: WREG pc 0xbfc00000
WCP0 register value	Write value to the selected Coprocessor 0 register. register the register number 0 .. 31, add 0x0n00 for Select n value the value to write into the register Example: WCP0 13 0x00000000 ;Clear Cause Register
RCP0 register	Read the selected Coprocessor 0 register. register the register number 0 .. 31, add 0x0n00 for Select n Example: RCP0 16 ; Read Config0
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
RM8 address value	Read a byte (8bit) from the selected memory place. address the memory address Example: RM8 0x00000000
RM16 address value	Read a half word (16bit) from the selected memory place. address the memory address Example: RM16 0x00000000
RM32 address value	Read a word (32bit) from the selected memory place. address the memory address Example: RM32 0x00000000

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.

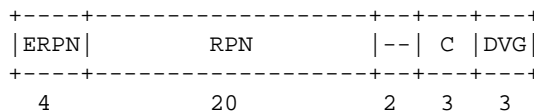
The vpn parameter defines the effective page number, size and ASID:



The 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:



The field ERPN (extended real page number) is used for physical address bits 35:32. 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.

```

[INIT]
; Setup TLB
WTLB 0x00000500 0x01FC0017 ;Boot ROM 2 x 1MB, uncached DVG
    
```

3.2.2 Part [TARGET]

The part [TARGET] defines some target specific values.

CPUTYPE type [MIPS16] This value gives the BDI information about the connected CPU. The optional parameter MIPS16 forces the BDI to use 16-bit software breakpoints in any case. If this parameter is not present, the "length" parameter of the GDB Z0 packet selects between 32-bit and 16-bit breakpoints.

type RC32300, AU1000, M4K, M4KE, M24K, M34K, M74K
M1004K, M1074K, EJTAG20, iAptiv
Example: CPUTYPE M4KE

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 = 16.6 MHz 2 = 5.5 MHz
1 = 8.3 MHz 3 = 4.1 MHz
Example: JTAGCLOCK 1 ; JTAG clock is 8.3 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 code. 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.
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.
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	<p>This parameter defines a delay in milliseconds the BDI waits after the target has been powered-up until JTAG communications starts.</p> <p>delay the power-up start delay in milliseconds (default 2 sec.)</p> <p>Example: POWERUP 5000 ;start delay after power-up</p>
WAKEUP time	<p>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.</p> <p>time the delay time in milliseconds</p> <p>Example: WAKEUP 3000 ; insert 3sec wake-up time</p>
STARTUP mode [runtime]	<p>This parameter selects the target startup mode:</p> <p>HALT This default mode tries to forces the target to debug mode immediately out of reset.</p> <p>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.</p> <p>RUN After reset, the target executes code until stopped by the Telnet "halt" command.</p> <p>Example: STARTUP STOP 3000 ; let the CPU run for 3 seconds</p>
BREAKMODE mode	<p>This parameter defines how breakpoints are implemented. The current mode can also be changed via the Telnet interface</p> <p>SOFT This is the normal mode. Breakpoints are implemented by replacing code with a SDBBR instruction.</p> <p>HARD In this mode, the EJTAG breakpoint hardware is used.</p> <p>Example: BREAKMODE HARD</p>
STEPMODE mode	<p>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 (e.g. RC32300) others support only one hardware instruction breakpoint.</p> <p>JTAG This is the default mode. The step feature of the EJTAG debug interface is used for single stepping.</p> <p>HWBP In this mode, one or two hardware breakpoints are used to implement single stepping.</p> <p>SWBP In this mode, one or two software breakpoints are used to implement single stepping.</p> <p>Example: STEPMODE HWBP</p>
VECTOR CATCH	<p>When this line is present, the BDI catches all unhandled exceptions. Catching exceptions is only possible if the vector table at 0x80000000 is writable.</p> <p>Example: VECTOR CATCH ; catch unhandled exception</p>

- 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
- MMU XLAT** The BDI supports Linux kernel debugging when MMU is on. If this line is present, the BDI assumes that all addresses received from GDB and Telnet are virtual addresses. If necessary the BDI creates appropriate TLB entries before accessing memory based on information found in the kernel or user page table.
- Translation can be probed with the Telnet command PHYS.
 For more information see also chapter "Embedded Linux MMU Support".
- Example: MMU XLAT ;enable virtual addresses translation
- PTBASE addr [64BIT]** This parameter defines the memory address where the BDI looks for the two page table pointers. If the additional "64BIT" option is present, the BDI assume 64-bit PTE's. For more information see also chapter "Embedded Linux MMU Support".
- addr Address of the memory used to store the two page table pointers.
 Example: PTBASE 0x800002f0
- 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.
 FPR The real floating point registers
 CP0 Some CP0 registers.
 Example: REGLIST STD FPR ; standard and FP registers

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 [bypass]

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).
- bypass An optional hexadecimal bypass pattern. Only necessary if one of the additional JTAG devices needs a bypass instruction that is not all one's (ffffff).

Example: SCANPRED 1 8 ; one device with an IR length of 8
 SCANPRED 1 5 12 ; use 10010 as bypass instruction

SCANSUCC count irlen [bypass]

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).
- bypass An optional hexadecimal bypass pattern. Only necessary if one of the additional JTAG devices needs a bypass instruction that is not all one's (ffffff).

Example: SCANSUCC 2 12 ; two device with an IR length of 8+4
 SCANSUCC 1 8 3c ; use 00111100 as bypass instr.

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 default name of the file that is loaded into RAM using the Telnet 'load' command. 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. 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
DEBUGPORT port [RECONNECT]	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). port the TCP port number (default = 2001) Example: DEBUGPORT 2001

- PROMPT string This entry defines a new Telnet prompt. The current prompt can also be changed via the Telnet interface.
 Example: PROMPT M4K>
- 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 echos 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

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	<p>This parameter defines the type of flash used. It is used to select the correct programming algorithm.</p> <p>format AM29F, AM29BX8, AM29BX16, I28BX8, I28BX16, AT49, AT49X8, AT49X16, STRATAX8, STRATAX16, MIRROR, MIRRORX8, MIRRORX16, S29M64X8, S29M32X16, S29GLSX16, S29VSRX16, M58X32, AM29DX16, AM29DX32</p> <p>Example: CHIPTYPE AM29F</p>
CHIPSIZE size	<p>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.</p> <p>size the size of one flash chip in bytes</p> <p>Example: CHIPSIZE 0x80000</p>
BUSWIDTH width	<p>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.</p> <p>with the width of the flash memory bus in bits (8 16 32)</p> <p>Example: BUSWIDTH 32</p>
FILE filename	<p>The default name of the file that is programmed into flash using the Telnet 'prog' command. 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.</p> <p>filename the filename including the full path or \$ for relative path.</p> <p>Example: FILE F:\gnu\arm\bootrom.hex FILE \$bootrom.hex</p>
FORMAT format [offset]	<p>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.</p> <p>format SREC, BIN, AOUT or ELF</p> <p>Example: FORMAT SREC FORMAT ELF 0x10000</p>
WORKSPACE address	<p>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.</p> <p>address the address of the RAM area</p> <p>Example: WORKSPACE 0x00000000</p>

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 0xbf40000 ;erase sector 4 of flash
 ERASE 0xbf60000 ;erase sector 6 of flash
 ERASE 0xbf10000 UNLOCK 100 ;unlock, wait 100ms
 ERASE 0xbf00000 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  0xFFFF00000  0x0060      unlock block 0
WM16  0xFFFF00000  0x00D0
WM16  0xFFFF10000  0x0060      unlock block 1
WM16  0xFFFF10000  0x00D0
WM16  0xFFFF10000  0x00D0
WM16  0xFFFF00000  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 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) 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\reg32334.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	

Note:

The following register names are predefined: pc, lo, hi, sr, accu, accu0, accu1, accu2, accu3

Example for a register definition (RC32334):

Entry in the configuration file:

```
[REGS]
DMM1 0xFF300000 ;DSU base address
DMM2 0xB8000000 ;Memory mapped registers
FILE E:\cygnus\root\usr\demo\mips\reg32334.def
```

The register definition file:

```
;name          type  addr          size
;-----
;
;
; CP0 Registers
;
index          CP0    0
random         CP0    1
elo0           CP0    2
elo1           CP0    3
context        CP0    4
pmask          CP0    5
wired          CP0    6
bad            CP0    8
ehi            CP0    10
;
count          CP0    9
compare        CP0    11
status         CP0    12
cause          CP0    13
...
;
; DSU Registers
;
dcr            DMM1   0x0000
ibs            DMM1   0x0004
dbs            DMM1   0x0008
pbs            DMM1   0x000c
....
;
; Internal Registers
;
; BUI Control Registers
bta            DMM2   0x0000
alt            DMM2   0x0004
arb            DMM2   0x0008
bec            DMM2   0x0010
bea            DMM2   0x0014
sysid         DMM2   0x0018
;
; Base Address and Mask Registers
mba0           DMM2   0x0080
mbm0           DMM2   0x0084
mba1           DMM2   0x0088
mbm1           DMM2   0x008c
....
```

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 bdi2000:2001
```

bdi2000 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 bdi2000:2001
```

3.3.3 Breakpoint Handling

GDB versions before V5.0:

GDB inserts breakpoints by replacing code via simple memory read / write commands. There is no command like "Set Breakpoint" defined in the GDB remote protocol. When breakpoint mode HARD is selected, the BDI checks the memory write commands for such hidden "Set Breakpoint" actions. If such a write is detected, the write is not performed and the BDI sets an appropriate hardware breakpoint. The BDI assumes that this is a "Set Breakpoint" action when memory write length is 4 bytes and the pattern to write is a BREAK opcode.

GDB version V5.x:

GDB version 5.x uses the 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 - 0xFF3FFFFFF 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 BDx gives control back to the BDI.

3.3.4 GDB monitor command

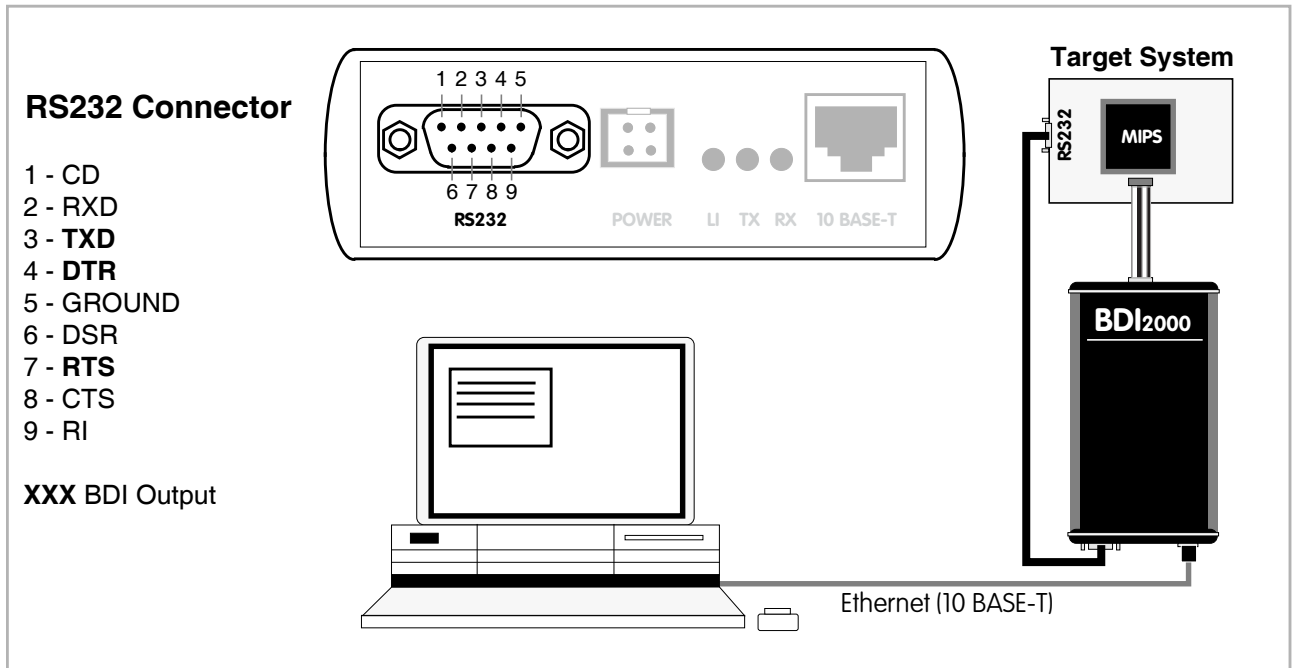
The BDI supports the GDB V5.x "monitor" command. Telnet commands are executed and the Telnet output is returned to GDB. This way you can for example switch the BDI breakpoint mode from within your GDB session.

```
(gdb) target remote bdi2000:2001
Remote debugging using bdi2000:2001
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 Target serial I/O via BDI

A RS232 port of the target can be connected to the RS232 port of the BDI2000. 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 BDI2000 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 BDI asserts RTS and DTR when a TCP connection is established.

```
[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.

3.3.6 Embedded Linux MMU Support

The bdiGDB system supports debugging of Linux kernel code that is allocated in mapped kernel space (kseg2). The MMU configuration parameter enables this mode of operation. Before the BDI accesses mapped memory space it creates an appropriate TLB entry based on information found in the kernel/user page tables. A temporary TLB entry is only created if there is not already a matching one present.

In order to search the page tables, the BDI needs to know the start addresses of it. The configuration parameter PTBASE defines the address in unmapped kernel space where the BDI looks for the addresses of the page tables. The first entry should point to the kernel page table (swapper_pg_dir), the second one can point to a pointer (current_pgd) that itself points to the current user page table. The second (user) page table is only searched if its address is not zero and there was no match in the first one.

The pointer structure is as follows:

```
PTBASE (unmapped address) ->
    PTE kernel pointer (unmapped address)
    PTE pointer pointer(unmapped address) ->
        PTE user pointer (unmapped address)
```

In order to let the kernel update the pointers needed by the BDI, you may add the following short code sequences to "head.S" at the end of "kernel_entry" (see also patch example on next page):

```
/* Setup the PTE pointers for the Abatron bdiGDB.
*/
li    t0, 0x800002f0 /* must match the bdiGDB config file */
la    t1, swapper_pg_dir
sw    t1, (t0)
addiu t0, 4
la    t1, current_pgd or pgd_current
sw    t1, (t0)
```

just before:

```
jal    init_arch
nop
END(kernel_entry)
```

In the configuration file define:

```
[TARGET]
....
MMU          XLAT          ;MMU support enabled
PTBASE       0x800002f0    ;here are the page table pointers
```

Note:

You are free to change the address of the array with the two pointers. Select an address in unmapped kernel space (kseg0) that is not actively used by any kernel code or data.

You may also manually setup the pointers via GDB or Telnet if you cannot change kernel code. Break for example at "start_kernel" and write the appropriate values to PTBASE[0] and PTBASE[1].

Example of a kernel patch that adds BDI support:

```
diff -ru linux.org/arch/mips/Kconfig.debug linux/arch/mips/Kconfig.debug
--- linux.org/arch/mips/Kconfig.debug2005-11-14 19:58:12.000000000 -0500
+++ linux/arch/mips/Kconfig.debug2006-06-26 14:44:30.000000000 -0400
@@ -38,6 +38,12 @@
     better 32 MB RAM to avoid excessive linking time. This is only
     useful for kernel hackers. If unsure, say N.

+config BDI_SWITCH
+    bool "Abatron bdiGDB kernel module debugging support"
+    depends on DEBUG_KERNEL
+    help
+        Enables the Abatron bdiGDB debugger to debug kernel modules
+
+config GDB_CONSOLE
+    bool "Console output to GDB"
+    depends on KGDB
diff -ru linux.org/arch/mips/kernel/head.S linux/arch/mips/kernel/head.S
--- linux.org/arch/mips/kernel/head.S2005-11-14 19:58:17.000000000 -0500
+++ linux/arch/mips/kernel/head.S2006-06-26 13:07:44.000000000 -0400
@@ -153,6 +153,16 @@
     set_saved_spsp, t0, t1
     PTR_SUBUsp, 4 * SZREG# init stack pointer

+#ifdef CONFIG_BDI_SWITCH
+    /* Setup the PTE pointers for the Abatron bdiGDB. */
+    la t0, bdi_ptbase
+    la t1, swapper_pg_dir
+    sw t1, (t0)
+    addiu t0, 4
+    la t1, pgd_current
+    sw t1, (t0)
+#endif
+
+    j          start_kernel
+    END(kernel_entry)

@@ -195,3 +205,7 @@
     page    invalid_pmd_table, _PMD_ORDER
 #endif
     page    invalid_pte_table, _PTE_ORDER
+
+#ifdef CONFIG_BDI_SWITCH
+    .comm bdi_ptbase, SZREG*2, SZREG /* BDI PTBASE should point to this */
+#endif
```

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:

```
DB1100>reset
- TARGET: processing user reset request
- Core#0: ID code is 0x2020228F
- Core#0: IMP reg is 0x20404000
- TARGET: resetting target passed
- TARGET: processing target startup ....
- TARGET: processing target startup passed
DB1100>info
  Core number      : 0
  Core state       : Debug Mode
  Debug entry cause : JTAG break request
  Current PC       : 0xbfc00000
  Current SR       : 0x00400004
  Current LR  (r31) : 0xff210000
  Current SP  (r29) : 0x00000000
DB1100>md 0xbfc00000
bfc00000 : 10000155 00000000 00000000 00000000  U.....
bfc00010 : 00000000 00000000 00000000 00000000  .....
bfc00020 : 00000000 00000000 00000000 00000000  .....
bfc00030 : 00000000 00000000 00000000 00000000  .....
.....
```

Note:

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:

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

"RD     [<name>]                    display general purpose or user defined register",
"RDUMP  [<file>]                    dump all user defined register to a file",
"RDCP0  <number>                    display CP0 register",
"RDFP                                       display floating point registers",
"DSP                                         display DSP ASE registers",
"RM     {<nbr>|<name>} <value>      modify general purpose or user defined register",
"RMCP0  <number> <value>            modify CP0 register",
"RMFP   <number> [<hi>_<lo>]        modify floating point 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 exceptions 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>                     start multiple cores (<cores> = core bit map)",
"TI     [<pc>]                       trace on instuction (single step)",
"HALT   [<cores>]                   force core(s) to debug mode (<cores> = core bit map)",
"BI     <addr> [<mask>]              set instruction breakpoint",
"CI     [<id>]                       clear instruction breakpoint(s)",
"BD     [R|W] <addr> [<mask>]       set data breakpoint",
"CD     [<id>]                       clear data breakpoint(s)",

"SELECT <core>                       change the current core",
"TCSELECT <thread>                  change the current MT ASE thread",
"INFO                                       display information about the current core",
"STATE                                       display information about all cores",
"TCINFO                                       display information about the MT ASE threads",

"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>]]]]",
"UPDATE                   reload the configuration without a reboot",
"HELP                     display command list",
"BOOT [loader]           reboot the BDI and reload the configuration",
"JTAG                     switch to JTAG command mode",
"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
9fc44940
BDI>exec 0x24061234      (addiu r6,r0,0x1234)
BDI>rgpr 6
00001234
BDI>exec 0x24c60005      (addiu r6,r6,4)
BDI>rgpr 6
00001239
BDI>sync
```

3.5 Multi-Core Support

The bdiGDB system supports concurrent debugging of up to 8 MIPS32 cores connected to the same JTAG scan chain. For every core you can start its own GDB session. The default port numbers used to attach the remote targets are 2001 ... 2008. In the Telnet you switch between the cores with the command "select <0..3>". 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.

The following example defines two cores on the scan chain.

```
[TARGET]
;common configurations
JTAGCLOCK      8000000      ;use 8 MHz JTAG clock
POWERUP        5000        ;power-up delay
WAKEUP         2000        ;delay after releasing reset

;configuration for core #0
#0 CPUTYPE     M4KE
#0 SCANPRED    0 0
#0 SCANSUCC    1 5          ;bypass second core
#0 ENDIAN      BIG          ;target is big endian
#0 BREAKMODE   HWBP        ;use hardware breakpoints

;configuration for core #1
#1 CPUTYPE     M4KE
#1 SCANPRED    1 5          ;bypass first core
#1 SCANSUCC    0 0
#1 ENDIAN      BIG          ;target is big endian
#1 BREAKMODE   SOFT        ;use software breakpoints
```

Multi-Core related Telnet commands:

```
"SELECT <core>          change the current core",
"CONT <cores>          start multiple cores (<cores> = core bit map)",
"HALT [<cores>]       force core(s) to debug mode (<cores> = core bit map)",
"STATE                 display information about all cores",
```

3.5.1 MIPS 34K MT ASE Support

The BDI has some basic support for the MIPS 34K Multithreading ASE (MT ASE). Via Telnet you can get information about the current state of the different VPEs and TCs.

The configuration for a MIPS 34K with two VPEs may look as follows:

```
[TARGET]
JTAGCLOCK    0                ;use 16 MHz JTAG clock
POWERUP      2000             ;power-up delay
WAKEUP       100              ;delay after releasing reset

; VPE0
#0 CPUTYPE    M34K             ;the used target CPU type
#0 ENDIAN     BIG              ;target is big endian
#0 STARTUP    RESET           ;halt VPE at the reset vector
#0 BREAKMODE  SOFT            ;SOFT or HARD, HARD uses hardware breakpoints
#0 STEPMODE   JTAG            ;JTAG, HWBP or SWBP
#0 SCANPRED   0 0             ;no device before
#0 SCANSUCC   1 5             ;one device after

; VPE1
#1 CPUTYPE    M34K             ;the used target CPU type
#1 ENDIAN     BIG              ;target is big endian
#1 STARTUP    RUN              ;don't halt VPE1, is not active out of reset
#1 BREAKMODE  SOFT            ;SOFT or HARD, HARD uses hardware breakpoints
#1 STEPMODE   JTAG            ;JTAG, HWBP or SWBP
#1 SCANPRED   1 5             ;one device before
#1 SCANSUCC   0 0             ;no device after
```

After halting the processor you can look at the current state:

```
VPE0>reset run
- TARGET: processing user reset request
- Core#0: ID code is 0x003400CD
- Core#0: IMP reg is 0x61414000
- Core#1: ID code is 0x003410CD
- Core#1: IMP reg is 0x61414000
- TARGET: resetting target passed
- TARGET: processing target startup ....
- TARGET: processing target startup passed
```

```
VPE0>halt
Core number      : 0
Core state       : Debug Mode
Debug entry cause : JTAG break request
Thread (Debug) VPE: 0 (0)
Thread (Debug) TC : 0 (0)
Thread PC        : 0x80101cac
Thread SR        : 0x11004301
Thread LR (r31)  : 0x80103a40
Thread SP (r29)  : 0x80321f90
```

```
VPE0>tcinfo
TC VPE Act Hlt SST PC (Restart)
*0 0 1 0 - 0x80101cac
1 1 1 0 0 0x83e10060
2 1 1 0 0 0x83e10080
3 1 1 0 0 0x83e10080
4 1 1 0 0 0x83e10060
```

It is possible to switch to an other thread via it SST bit:

```
VPE0>tcinfo
TC VPE Act Hlt SST PC (Restart)
*0 0 1 0 - 0x80101cac
 1 1 1 0 0 0x83e10060
 2 1 1 0 0 0x83e10080
 3 1 1 0 0 0x83e10080
 4 1 1 0 0 0x83e10060

VPE0>tcsel 3
Core number      : 0
Core state       : Debug Mode
Debug entry cause : JTAG break request
Thread (Debug) VPE: 1 (0)
Thread (Debug) TC : 3 (0)
Thread PC        : 0x83e10080
Thread SR        : 0x01000000
Thread LR (r31)  : 0x83e10148
Thread SP (r29)  : 0x9d004fac
```

```
VPE0>rm debug 0x100
```

```
VPE0>tcinfo
TC VPE Act Hlt SST PC (Restart)
*0 0 1 0 - 0x80101cac
 1 1 1 0 0 0x83e10060
 2 1 1 0 0 0x83e10080
 3 1 1 0 1 0x83e10080
 4 1 1 0 0 0x83e10060
```

```
VPE0>go
- TARGET: core #1 has entered debug mode
```

```
VPE0>select 1
Core number      : 1
Core state       : Debug Mode
Debug entry cause : single step
Thread (Debug) VPE: 1 (1)
Thread (Debug) TC : 3 (3)
Thread PC        : 0x83e10084
Thread SR        : 0x01000000
Thread LR (r31)  : 0x83e10148
Thread SP (r29)  : 0x9d004fac
```

```
VPE1>tcinfo
TC VPE Act Hlt SST PC (Restart)
 0 0 1 0 0 0x80000200
 1 1 1 0 0 0x83e10064
 2 1 1 0 0 0x83e10084
*3 1 1 0 - 0x83e10084
 4 1 1 0 0 0x83e10064
```

4 Specifications


Operating Voltage Limiting	5 VDC \pm 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 BASE-T
Serial Transfer Rate between BDI and Target	up to 16 Mbit/s
Supported target voltage	1.8 – 5.0 V (3.0 – 5.0 V with Rev. A/B)
Operating Temperature	+ 5 °C ... +60 °C
Storage Temperature	-20 °C ... +65 °C
Relative Humidity (noncondensing)	<90 %rF
Size	190 x 110 x 35 mm
Weight (without cables)	420 g
Host Cable length (RS232)	2.5 m

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)


DECLARATION OF CONFORMITY

This declaration is valid for following product:

Type of device: BDM/JTAG Interface
Product name: BDI2000

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:


EN 50081-2
EN 50082-2


This declaration of conformity is based on the test report no. QNL-E853-05-8-a of QUINEL, Zug, accredited according to EN 45001.

Manufacturer:

ABATRON AG
Stöckenstrasse 4
CH-6221 Rickenbach

Authority:


Max Vock
Marketing Director


Ruedi Dummermuth
Technical Director

Rickenbach, May 30, 1998

7 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 target system (see chapter 2).
- The power supply of the target system is switched off or not in operating range (4.75 VDC ... 5.25 VDC) --> MODE LED is OFF or RED
- The built in fuse is damaged --> MODE LED is OFF
- The BDI is not correctly connected with the Host (see chapter 2).
- A wrong communication port (Com 1...Com 4) is selected.

Problem

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

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 (BDI2000 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 BDI2000 is not connected or not correctly connected to the network (LAN cable or media converter)
- An incorrect IP address was entered (BDI2000 configuration)

B Maintenance

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

If the BDI is connected correctly and it is still not responding, then the built in fuse might be damaged (in cases where the device was used with wrong supply voltage or wrong polarity). To exchange the fuse or to perform special initialization, please proceed according to the following steps:



Observe precautions for handling (Electrostatic sensitive device)
Unplug the cables before opening the cover.
Use exact fuse replacement (Microfuse MSF 1.6 AF).

1

1.1 Unplug the cables

2

2.1 Remove the two plastic caps that cover the screws on target front side (e.g. with a small knife)

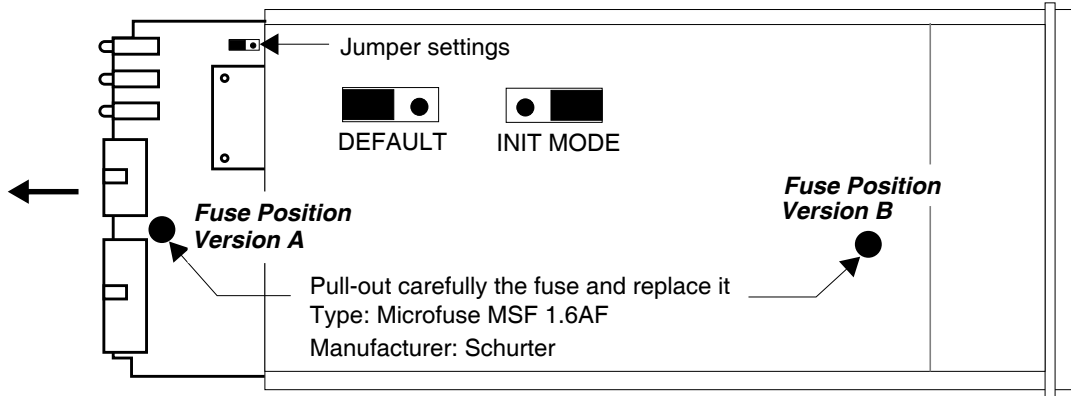
2.2 Remove the two screws that hold the front panel

3

3.1 While holding the casing, remove the front panel and the red elastic sealing

4

4.1 While holding the casing, slide carefully the print in position as shown in figure below

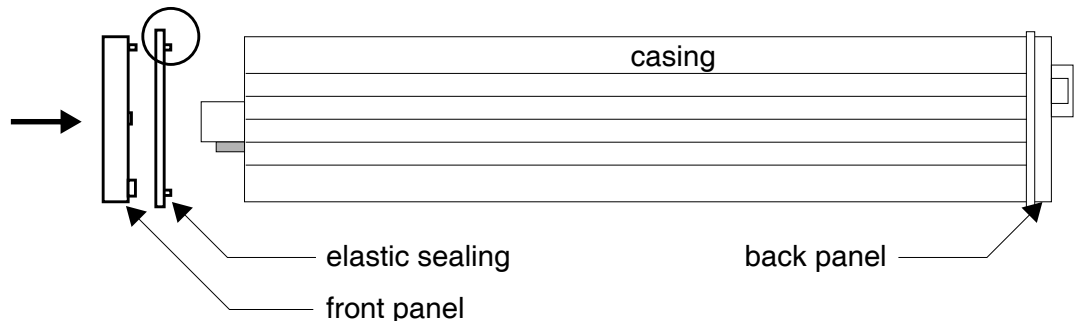


5

Reinstallation

5.1 Slide back carefully the print. Check that the LEDs align with the holes in the back panel.

5.2 Push carefully the front panel and the red elastic sealing on the casing. Check that the LEDs align with the holes in the front panel and that the position of the sealing is as shown in the figure below.



5.3 Mount the screws (do not overtighten it)

5.4 Mount the two plastic caps that cover the screws

5.5 Plug the cables



**Observe precautions for handling (Electrostatic sensitive device)
Unplug the cables before opening the cover.
Use exact fuse replacement (Microfuse MSF 1.6 AF).**

C Trademarks

All trademarks are property of their respective holders.