bdiGDB

JTAG debug interface for GNU Debugger

PowerPC MPC85xx/P10xx/P2020

User Manual

Manual Version 1.15 for BDI2000

abatron

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
2.3 Status LED «MODE» 10
2.4 Connecting the BDI2000 to Host 11
2.4.1 Serial line communication 11
2.4.2 Ethernet communication 12
2.5 Initial configuration of the bdiGDB system 13
2.5.1 Configuration with a Linux / Unix host 14
2.5.2 Configuration with a Windows host 16
2.5.3 Recover procedure 17
2.6 Testing the BDI2000 to host connection 18
2.7 TFTP server for Windows 18

3 Using bdiGDB 19
3.1 Principle of operation 19
3.2 Configuration File 21
3.2.1 Part [INIT] 22
3.2.2 Part [TARGET] 25
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 Dual-Core Support 45

4 Specifications 47

5 Environmental notice 48

6 Declaration of Conformity (CE) 48

7 Warranty and Support Terms 49
7.1 Hardware 49
7.2 Software 49
7.3 Warranty and Disclaimer 49
7.4 Limitation of Liability 49

Appendices

A Troubleshooting 50

B Maintenance 51

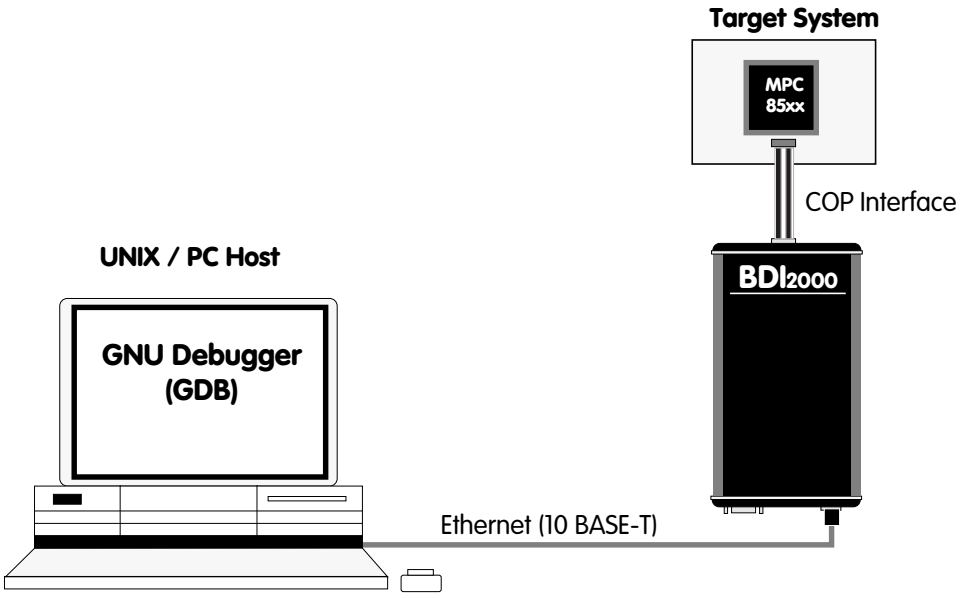
C Trademarks 53

1 Introduction

bdiGDB enhances the GNU debugger (GDB), with JTAG/COP debugging for PowerPC MPC85xx 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 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 JTAG 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 Windows based configuration program. The BDI2000 supports 1.8 – 5.0 Volts target systems (3.0 – 5.0 Volts target systems with Rev. 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 MPC8560ADS
;---------------------------------------
;
[INIT]
; init core register
;
;
; load TLB entries, helper code @ 0xfffff000
WM32    0xfffff000      0x7c0007a4      ;tlbwe
WM32    0xfffff004      0x7c0004ac      ;msync
WM32    0xfffff008      0x48000000      ;loop
;
WSPR    624             0x10030000      ;MAS0: TLB1, Index 3
WSPR    625             0x80000800      ;MAS1: valid, 64 Mbyte
WSPR    626             0x00000008      ;MAS2: 0x00000000, I
WSPR    627             0x0000003f      ;MAS3: 0x00000000, UX,SX,UW,SW,UR,SR
EXEC    0xfffff000
;
;
;
[TARGET]
CPUTYPE     8560        ;the CPU type
JTAGCLOCK   1           ;use 8 MHz JTAG clock
BREAKMODE   SOFT        ;SOFT or HARD, HARD uses PPC hardware breakpoint


[HOST]
IP           151.120.25.119
FILE        E:\cygwin\home\demo\e500\fibo.elf
FORMAT      ELF
LOAD         MANUAL       ;load code MANUAL or AUTO after reset


[FLASH]
CHIPTYPE    STRATAX16
CHIPTYPE    AM29BX16
CHIPSIZE    0x800000    ;The size of one flash chip in bytes
BUSWIDTH    32          ;The width of the flash memory bus in bits (8 | 16 | 32)
WORKSPACE   0x40080000  ;workspace in dual port RAM
FILE        E:\cygwin\home\bdidemo\e500\ads8560.cfg
FORMAT      BIN 0xFF800000
ERASE       0xFF800000  ;erase sector 0

[REGS]
FILE        E:\cygwin\home\bdidemo\e500\reg8560.def
```

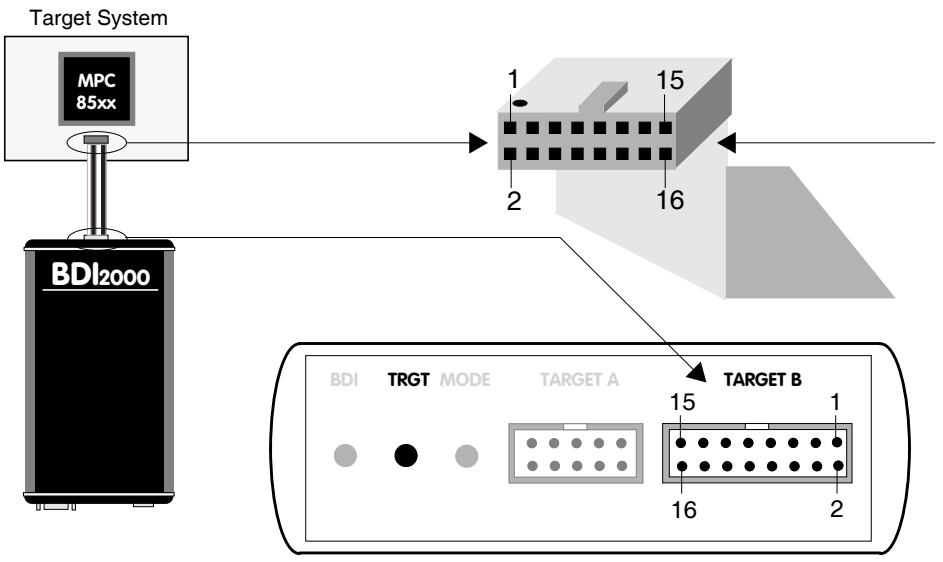
Based on the information in the configuration file, the target is automatically initialized after every reset.

2 Installation

2.1 Connecting the BDI2000 to Target

The cable to the target system is a 16 pin flat ribbon cable. In case where the target system has an appropriate connector, the cable can be directly connected. The pin assignment is in accordance with the PowerPC COP connector specification.

In order to ensure reliable operation of the BDI (EMC, runtimes, etc.) the target cable length must not exceed 20 cm (8").



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

COP/JTAG Connector

1 - TDO
2 - NC (QACK)
3 - TDI
4 - TRST
5 - NC (QREQ)
6 - Vcc Target
7 - TCK
8 - CHKSTP_IN
9 - TMS
10 - NC
11 - SRESET
12 - GROUND
13 - HRESET
14 - NC (key)
15 - CHKSTP_OUT
16 - GROUND

For BDI TARGET B connector signals see table on next page.

BDI TARGET B Connector Signals:

Pin	Name	Describtion
1	TDO	**JTAG Test Data Out** This input to the BDI2000 connects to the target TDO pin.
2	IO2	**General purpose I/O** Currently not used.
3	TDI	**JTAG Test Data In** This output of the BDI2000 connects to the target TDI pin.
4	$\overline{\text{TRST}}$	**JTAG Test Reset** This output of the BDI2000 resets the JTAG TAP controller on the target.
5	IN0	**General purpose Input** Currently not used.
6	Vcc 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 connected to Vdd I/O on the target board. **3.0 – 5.0V with Rev. 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.
7	TCK	**JTAG Test Clock** This output of the BDI2000 connects to the target TCK pin.
8	IO8	**General purpose I/O** This output of the BDI2000 connects to the target CKSTP_IN pin. Currently not used.
9	TMS	**JTAG Test Mode Select** This output of the BDI2000 connects to the target TMS line.
10	IO10	**General purpose I/O** Currently not used.
11	$\overline{\text{SRESET}}$	**Soft-Reset** This open collector output of the BDI2000 connects to the target SRESET pin.
12	GROUND	**System Ground**
13	$\overline{\text{HRESET}}$	**Hard-Reset** This open collector output of the BDI2000 connects to the target HRESET pin.
14	<reseved>	
15	IN1	**General purpose Input** This input to the BDI2000 connects to the target CKSTP_OUT pin. Currently not used.
16	GROUND	**System Ground**

2.1.1 Changing Target Processor Type

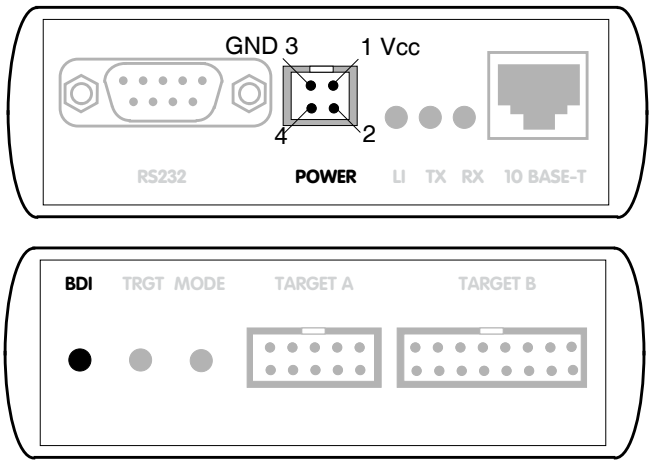
Before you can use the BDI2000 with an other target processor type (e.g. CPU32 <--> PPC), a new setup has to be done (see chapter 2.5). During this process the target cable must be disconnected from the target system. The BDI2000 needs to be supplied with 5 Volts via the BDI OPTION connector (Version A) or via the POWER connector (Version B). 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

The BDI2000 needs to be supplied with 5 Volts (max. 1A) via the POWER connector. 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.**



POWER Connector

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

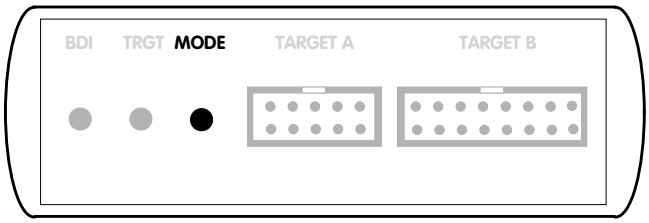
The green LED «BDI» marked light up when 5V power is connected to the BDI2000

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:



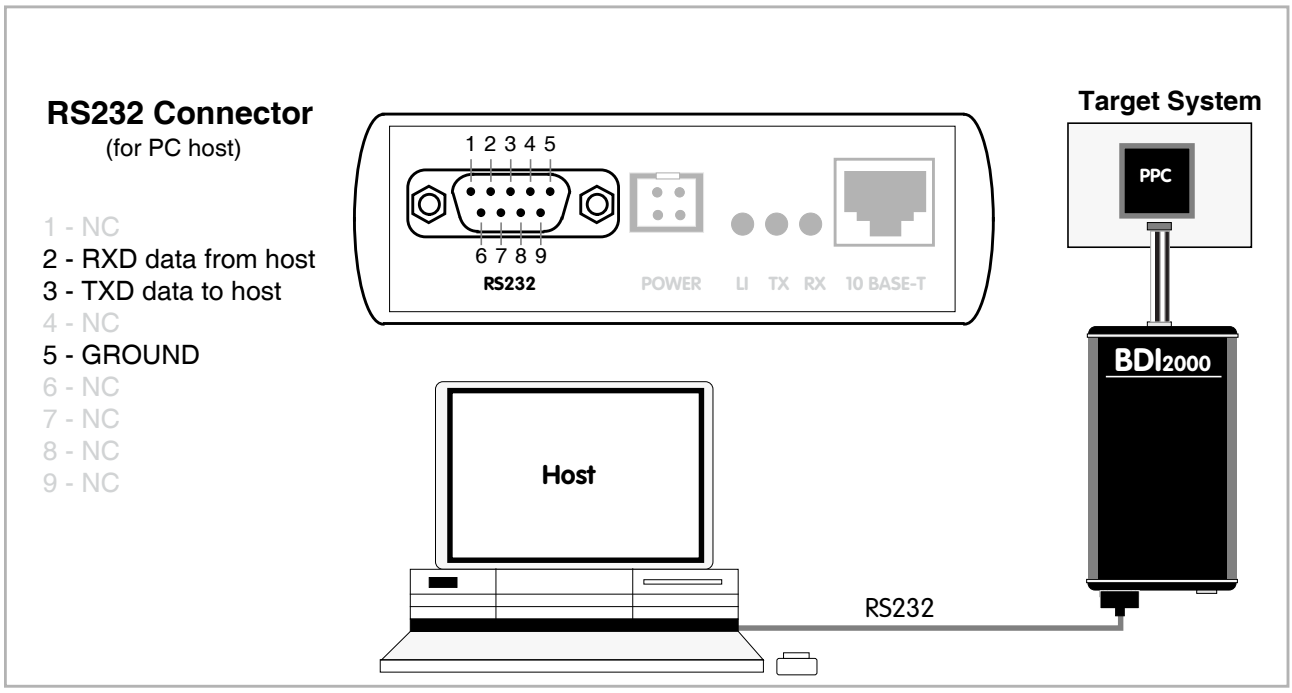
MODE LED	BDI STATES
OFF	The BDI is ready for use, the firmware is already loaded.
ON	The power supply for the BDI2000 is < 4.75VDC.
BLINK	The BDI «loader mode» is active (an invalid firmware is loaded or loading firmware is active).

2.4 Connecting the BDI2000 to Host

2.4.1 Serial line communication

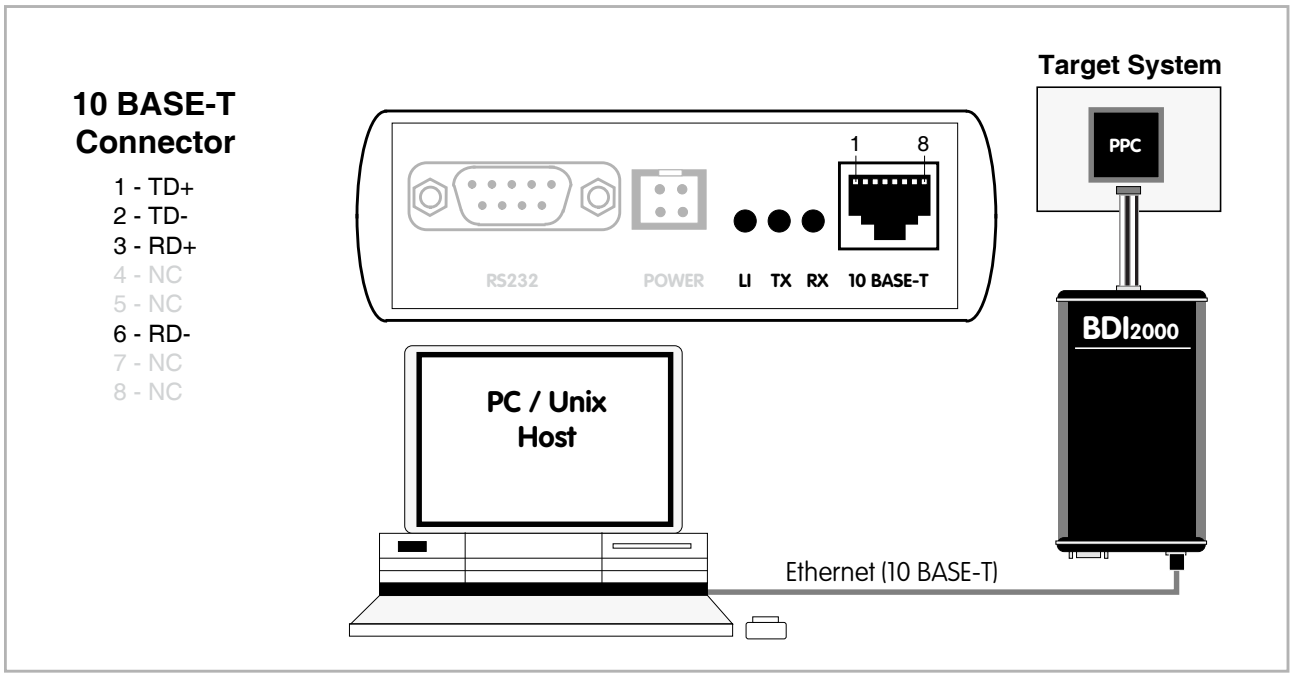
Serial line communication is only used for the initial configuration of the bdiGDB system.

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 (Unshilded Twisted Pair) cable to the BD2000. 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 Initial configuration of the bdiGDB system

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

The following files are on the CD.

b20pq3gd.exe	Windows configuration program
b20pq3gd.xxx	Firmware for the BDI2000
copjed20.xxx	JEDEC file for the BDI2000 (Rev. B) logic device when working with a COP target
copjed21.xxx	JEDEC file for the BDI2000 (Rev. C) logic device when working with a COP target
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 CD 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 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# 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 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.

```
[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      : 00-0c-01-92-15-21
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 -tMPC8500
Connecting to BDI loader
Erasing CPLD
Programming firmware with ./b20pwsgd.100
Programming CPLD with ./copjed21.102
```


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 \
> -fmpc8560.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 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 MPC8500
Logic    : V1.02 PPC6xx/PPC7xx
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   : mpc8560.cfg
```

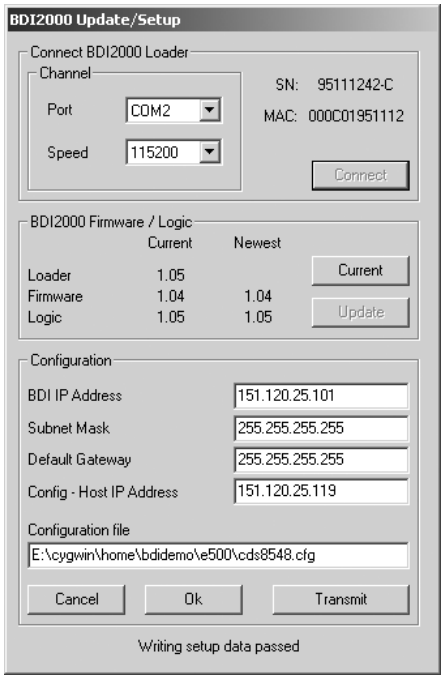
The Mode LED should go off, and you can try to connect to the BDI via Telnet.

```
[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:

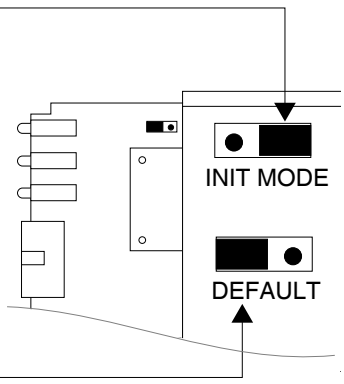
Port	Select the communication port where the BDI2000 is connected during this setup session.
Speed	Select the baudrate used to communicate with the BDI2000 loader during this setup session.
Connect	Click on this button to establish a connection with the BDI2000 loader. Once connected, the BDI2000 remains 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 name is transmitted to the TFTP server when reading the configuration file.
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 BDI2000 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 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"

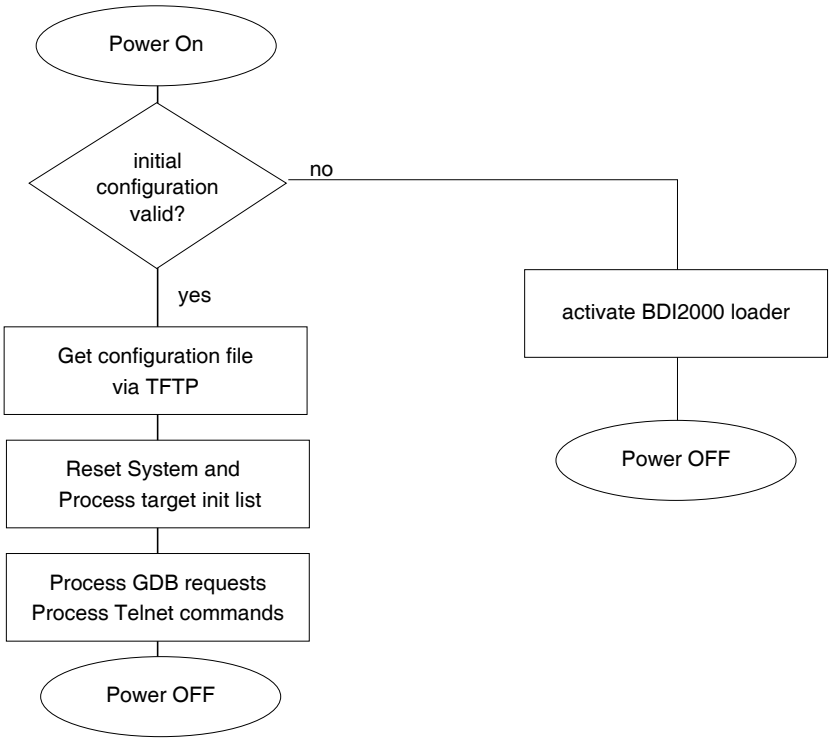
You may enter the TFTP server into the Startup group so the server is started every time you login.

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:



Breakpoints:
There are 3 breakpoint modes supported. One of them (SOFT) is implemented by replacing application code with a TRAP instruction. The other (HARD) uses the built in breakpoint logic. If HARD is used, only 2 breakpoint can be active at the same time. The third mode (LOOP) replaces the code with an endless loop, the processor does not enter debug mode until it is halted via Telnet of GDB. The breakpoint mode LOOP does not depend on valid code at the Debug Interrupt Vector.

The following example selects SOFT as the breakpoint mode:

```
BREAKMODE   SOFT      ;SOFT, HARD or LOOP, HARD uses PPC hardware breakpoints
```

Debug Interrupt (IVOR15):
Debugging via JTAG and flash programming with workspace works only if the Debug Interrupt Vector contains a valid instruction that can be fetched by the e500 core. This because the e500 core does first interrupt processing before it halts. If IVPR + IVOR15 do not point to a valid and fetchable (MMU) instruction the e500 will crash. If necessary (for example for flash programming) setup a valid Debug Interrupt Vector via some init list entries:

```
; Setup flash programming workspace in L2SRAM
WM32   0x40020000      0x68010000      ;L2CTL
WM32   0x40020100      0xf0000000      ;L2SRBAR0
WM32   0x40020000      0xA8010000      ;L2CTL
WSPR   63              0xf0000000      ;IVPR to workspace
WSPR   415             0x0001500       ;IVOR15 : Debug exception
WM32   0xf0001500      0x48000000      ;write valid instruction
```

Because a Debug Interrupt writes to CSRR0 and CSRR1, it is not possible to debug the entry/exit code of a critical interrupt handler with breakpoint mode SOFT or HARD.

Target Reset Sequence (STARTUP LOOP mode):
In order to get control of the core immediately out of reset, the BDI uses a special startup sequence where L2SRAM is mapped to the initial boot page and an endless loop is written to 0xfffffffc. This is done while the core is still kept in reset state. Then the core is released and starts executing this loop at 0xfffffffc until the BDI halts it via the appropriate JTAG command. Therefore after a reset sequence L2SRAM is mapped to 0xfffc0000...0xffffffff. To disable this mapping, enter the appropriate init list entry that disables L2SRAM.

```
WM32   0xFF720000      0x20000000      ;L2CTL : disable L2SRAM
```

Note about e500 JTAG debugging:
JTAG debugging works only correct if MSR[DE] is not cleared and there are no writes to the debug registers DBCRx by code running on the target. Writing to DBCRx may clear breakpoints set by the JTAG debugger.

3.2 Configuration File

The configuration file is automatically read by the BDI 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

WSPR register value — Write value to the selected special purpose register.
- register — the register number
- value — the value to write into the register

Example: WSPR 27 0x00001002 ; SRR1 : ME,RI

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 0x00001000

DELAY value — Delay for the selected time. A delay may be necessary to let the clock PLL lock again after a new clock rate is selected.
- value — the delay time in milliseconds (1...30000)

Example: DELAY 500 ; delay for 0.5 seconds

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. This entry is mainly used to unlock flash blocks. The pattern written is generated by duplicating the value (0x12345678 -> 0x1234567812345678).
- address — the memory address
- value — the value used to generate the pattern

Example: WM64 0xFFF00000 0x00600060 ; unlock block 0

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

RM64 address value — Read a double word (64bit) from the selected memory place.
address the memory address
Example: RM64 0x00000000

SUPM memaddr mdraddr — Starts a sequence of writes to the UPM RAM array (MPC85xx).
memaddr an address in the UPM memory range
mdraddr the address of the MDR register
Example: WM32 0x40005018 0x10000081 ; BR3
WM32 0x40005070 0x10000000 ; MAMR setup
SUPM 0x10000000 0x40005088

WUPM dummy data — Write to the UPM RAM array (*mdraddr = data, *memaddr = 0).
dummy this value is not used here (use 0)
data this value is written to the UPM data register
Example: WUPM 0 0x0FFFEC04

TSZ1 start end — Defines a memory range with 1 byte maximal transfer size.
Normally when the BDI reads or writes a memory block, it tries to access the memory with a burst access. The TSZx entry allows to define a maximal transfer size for up to 8 address ranges.
start the start address of the memory range
end the end address of the memory range
Example: TSZ1 0xFF000000 0xFFFFFFFF ; PCI ROM space

TSZ2 start end — Defines a memory range with 2 byte maximal transfer size.

TSZ4 start end — Defines a memory range with 4 byte maximal transfer size.

TSZ8 start end — Defines a memory range with 8 byte maximal transfer size.

MMAP start end — Because a memory access to an invalid memory space via JTAG can lead to a deadlock, this entry can be used to define up to 32 valid memory ranges. If at least one memory range is defined, the BDI checks against this range(s) and avoids accessing of not mapped memory ranges.
start the start address of a valid memory range
end the end address of this memory range
Example: MMAP 0xFFE00000 0xFFFFFFFF ;Boot ROM

EXEC addr [time]

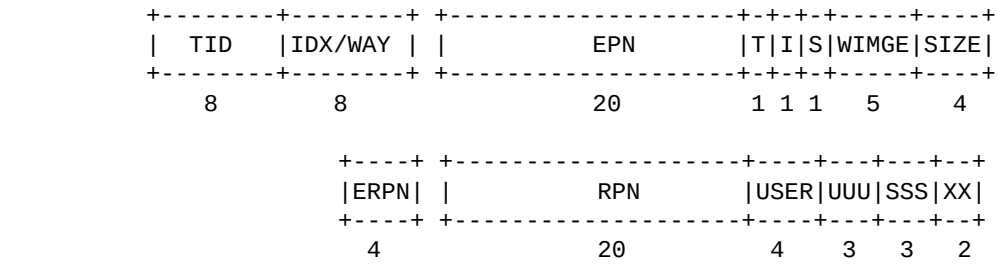
This entry cause the processor to start executing the code at addr. The optional second parameter defines a time in us how long the BDI let the processor run until it is halted. By default the BDI let it run for 500 us. This EXEC function maybe used to create TLB entries via some helper code.

addr	the start address of the code to execute
time	the time the BDI let the processor run (micro seconds).
Example:	EXEC 0xFFFFF000 ; write the TLB entry.

WTLB idx_epn erpn_rpn

Adds an entry to the L2 MMU (TLB1/TLB0) directly without the use of any helper code.. The two 64-bit values of an init list entry are used to define all the relevant parameters. See below. A SIZE of 0 selects TLB0 (L2TLB).

idx_epn	defines TID, IDX/WAY, EPN, flasgs and size
erpn_rpn	defines ERPN, RPN and attributs and flags



Example how to write to the UPM array:

```
WM32   0x4000501C   0xFF000000   ;OR3
WM32   0x40005018   0x10000081   ;BR3
WM32   0x40005070   0x10000000   ;MAMR : setup for array write
SUPM   0x10000000   0x40005088   ;set address of UPM range and MDR
WUPM   0x00000000   0xaba00000   ;write UPM array
WUPM   0x00000000   0xaba00001
WUPM   0x00000000   0xaba00002
WUPM   0x00000000   0xaba00003
WUPM   0x00000000   0xaba00004
       ...
WUPM   0x00000000   0xaba0003A
WUPM   0x00000000   0xaba0003B
WUPM   0x00000000   0xaba0003C
WUPM   0x00000000   0xaba0003D
WUPM   0x00000000   0xaba0003E
WUPM   0x00000000   0xaba0003F
WM32   0x40005070   0x00000000   ;MAMR : setup for normal mode
```


3.2.2 Part [TARGET]

The part [TARGET] defines some target specific values.

CPUTYPE type

This value gives the BDI information about the connected CPU.

type	8540, 8560, 8555, 8541, 8548, 8547, 8545, 8543 8568, 8567, 8533, 8544, 8572, 8536, 8569 P1010, P1011, P1012, P1013, P1014, P1015, P1016 P1017, P1020, P1021, P1022, P1023, P1024, P1025 P2020, P2010
Example:	CPUTYPE 8560

ENDIAN [BIG | LITTLE]

Selects endian mode (default is BIG). This is a general switch and mixed endian is not supported. Care must be taken when accessing CCSR registers. These memory mapped registers are big endian. Have a look at regP1020LE.def. Out of reset boot space is big endian.

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	4 = 500 kHz	7 = 50 kHz
1 = 8.3 MHz	5 = 200 kHz	8 = 20 kHz
2 = 4.1 MHz	6 = 100 kHz	9 = 10 kHz
3 = 1.0 MHz		10 = 5 kHz

Example: CLOCK 1 ; JTAG clock is 8.3 MHz

POWERUP delay

When the BDI detects target power-up, HRESET is forced immediately. This way no code from a boot ROM is executed after power-up. The value entered in this configuration line is the delay time in milliseconds the BDI waits before it begins JTAG communication. This time should be longer than the on-board reset circuit asserts HRESET.

delay — the power-up start delay in milliseconds

Example: POWERUP 5000 ;start delay after power-up

RESET type [time]

Normally the BDI toggles HRESET during the reset sequence. If reset type is NONE, the BDI does not assert HRESET at all. If reset type is KEEP then HRESET is asserted during the whole target power-up cycle to prevent the execution of any maybe not present boot code.This entry can also be used to change the default reset time.

type	NONE HARD (default) KEEP (keep HRESET asserted during target power-up)
time	The time in milliseconds the BDI assert the reset signal.
Example:	RESET NONE ; no reset during startup RESET HARD 1000 ; assert RESET for 1 second

WAKEUP time

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

time — the delay time in milliseconds

Example: WAKEUP 3000 ; insert 3sec wake-up time

STARTUP mode [runtime][mode [SMP]]

This parameter selects the target startup mode. The second mode defines how to handle the second e500 core in a dual-core processor. Not all mode combinations are supported. See chapter about Dual-Core support. If SMP is defined the BDI halts also the other core when one core halts because of a breakpoint.
The following modes are supported:

LOOP	This default mode forces the target to debug mode immediately out of reset. For this, L2SRAM is mapped to the inital boot page with an endless loop at 0xfffffffc.
HALT	Also this mode forces the target to debug mode immediately out of reset but without mapping L2SRAM. This works only if the processor can fetch a valid opcode from the boot address at 0xfffffffc.
STOP	In this mode, the BDI lets the target execute code for "runtime" milliseconds after reset.
RUN	After reset, the target executes code until stopped by the Telnet "halt" command.
WAIT	Force core to debug mode once enabled.
Example:	STARTUP STOP 3000 ; let the CPU run for 3 seconds

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

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 TRAP instruction.
HARD	In this mode, the PPC breakpoint hardware is used. Only 2 breakpoints at a time is supported.
LOOP	In this mode, breakpoints are implemented by replacing code with an endless loop (0x48000000). Maybe useful for special debug tasks. The processor does not automatically enter debug mode, it has to be halted manually via Telnet or GDB.
Example:	BREAKMODE HARD

STEPMODE mode

This parameter defines how single step (instruction step) is implemented. The alternate step mode (HWBP) may be useful when stepping instructions that causes a TLB miss exception.
In case BREAKMODE LOOP is selected, this parameter is ignored and single step is implemented by replacing the code of the next instruction(s) with an endless loop (0x48000000).

JTAG	This is the default mode. Single step is implemented by using the JTAG single step feature.
HWBP	In this mode, a hardware breakpoint on the next instruction is used to implement single stepping.
ICMP	In this mode, single step is implemented via the instruction complete (ICMP) debug event.
Example:	STEPMODE HWBP

MMU XLAT [kb]

In order to support Linux kernel debugging when MMU is on, the BDI translates effective (virtual) to physical addresses. This translation is done based on the current MMU configuration (page tables). If this configuration line is present, the BDI translates the addresses received from GDB before it accesses physical memory. The optional parameter defines the kernel virtual base address (default is 0xC0000000) and is used for default address translation. For more information see also chapter "Embedded Linux MMU Support". Addresses entered at the Telnet are never translated. Translation can be probed with the Telnet command PHYS.
If not zero, the 12 lower bits of "kb" defines the position of the page present bit in a page table entry. By default 0x800 is assumed for the page present bit. The position may depend on the Linux kernel version.
A "kb" value of 0xFFFFFFFF disables the default translation.

kb	The kernel virtual base address (KERNELBASE)
Example:	MMU XLAT ;enable address translation MMU XLAT 0xC0000800 ; page present bit is 0x800

PTBASE addr [64BIT]

This parameter defines the physical memory address where the BDI looks for the virtual/physical address of the array with the two page table pointers. For more information see also chapter "Embedded Linux MMU Support". If this parameter is not defined, the BDI searches TLB0 in order to translate a virtual address (TLB1 is always searched).
If the additional "64BIT" option is present, the BDI assume a 64-bit PTE.

addr	Physical address of the memory used to store the virtual address of the array with the two page table pointers.
Example:	PTBASE 0xf0

ROMLOC value

For some devices it is possible to override the Boot ROM Location. When this option is used, the Boot Sequencer is also disabled.

value	Boot ROM Location value (see processor manual)
Example:	ROMLOC 6 ; override Boot ROM location

REGLIST list

This parameter defines the registers packet that is sent to GDB in response to a register read command. By default STD and FPR are read and transferred. This default is compatible with older GDB versions. The following names are use to select a register group or packet format:

STD	The standard (old) register block. The FPR registers are not read from the target but transferred. You can't disable this register group.
FPR	The floating point registers are read and transferred.
E500	The register packet is sent as expected by GDB for a PowerPC E500 target.
Example:	REGLIST STD ;only standard registers REGLIST E500 ;send E500 register set

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

Daisy chained JTAG devices:

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 PowerPC chip (Predecessor). Enter the appropriate information also for the devices following the PowerPC chip (Successor):

SCANPRED count irlen

This value gives the BDI information about JTAG devices present before the PowerPC 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 PowerPC 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

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\ppc\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, IMAGE* 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 image file. If this value is not defined and the core is already in ROM, the PC will not be set before starting the program file. This means, the program starts at the normal reset address (0xFFF00100).

address the address where to start the program file

Example: START 0x1000

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 MPC8548>

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

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.

format AM29F, AM29BX8, AM29BX16, I28BX8, I28BX16, AT49, AT49X8, AT49X16, STRATAX8, STRATAX16, MIRROR, MIRRORX8, MIRRORX16, S29M32X16, 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 16

FILE filename

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.

filename the filename including the full path or $ for relative path.

Example: FILE F:\gnu\ppc\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.
You get the best programming performance when using a binary format (BIN, AOUT, ELF or IMAGE).

format SREC, BIN, AOUT, ELF or IMAGE

Example: FORMAT BIN 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 ADS8260 flash memory:

```
[FLASH]
CHIPTYPE    I28BX8      ;Flash type
CHIPSIZE    0x200000    ;The size of one flash chip in bytes (e.g. AM29F010 = 0x20000)
BUSWIDTH    32          ;The width of the flash memory bus in bits (8 | 16 | 32 | 64)
WORKSPACE   0x04700000  ;workspace in dual port RAM
FILE        E:\gnu\demo\ads8260\bootrom.hex  ;The file to program
ERASE       0xFF900000  ;erase sector 4 of flash SIMM (LH28F016SCT)
ERASE       0xFF940000  ;erase sector 5 of flash SIMM
ERASE       0xFF980000  ;erase sector 6 of flash SIMM
ERASE       0xFF9c0000  ;erase sector 7 of flash SIMM
```

the above erase list maybe replaces with:

```
ERASE       0xFF900000  0x40000  4 ; erase sector 4 to 7 of flash SIMM
```


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.

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

```
name   type   addr   [size [SWAP]]
```

name — The name of the register (max. 15 characters)

type — The register type

GPR	General purpose register
SPR	Special purpose register
CCSR	Relative to CCSRBAR memory mapped register. The BDI knows the current position of the CCSR space.
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 (default is 32)

SWAP — If present, the bytes of a 16bit or 32bit register are swapped. This is useful to access little endian ordered registers (e.g. PCI bridge configuration registers).

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\mpc8260.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 0x01000

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

Remark:
The registers **msr**, **cr, iar** and **acc** and are predefined.

Example for a register definition:

Entry in the configuration file:

```
[REGS]
FILE       E:\cygwin\home\bdidemo\e500\reg8560.def
```

The register definition file:

```
;name         type   addr          size
;-----------------------------------------
;
sp            GPR    1
;
;
csrr0         SPR    58
csrr1         SPR    59
ctr           SPR    9
dac1          SPR    316
dac2          SPR    317
dbcr0         SPR    308

       .....

pid0          SPR     48
pid1          SPR    633
pid2          SPR    634
spefscr       SPR    512
tlb0cfg       SPR    688
tlb1cfg       SPR    689
;
;
;       Local Bus Controller
br0           CCSR    0x05000
br1           CCSR    0x05008
br2           CCSR    0x05010
br3           CCSR    0x05018

              ....

lteatr        CCSR    0x050BC
ltear         CCSR    0x050C0
lbcr          CCSR    0x050D0
lcrr          CCSR    0x050D4
```

Now the defined registers can be accessed by name via the Telnet interface:

```
BDI>rd csrr0
BDI>rm br0 0x00000801
```


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 reset 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 0x7D821008 (tw 12,r2,r2).

GDB version >= V5.x:
GDB version >= 5.x uses the Z-packet to set breakpoints (watchpoints). For software breakpoints, the BDI replaces code with 0x7D821008 (tw 12,r2,r2). When breakpoint mode HARD is selected, the BDI sets an appropriate hardware breakpoint.

3.3.4 GDB monitor command

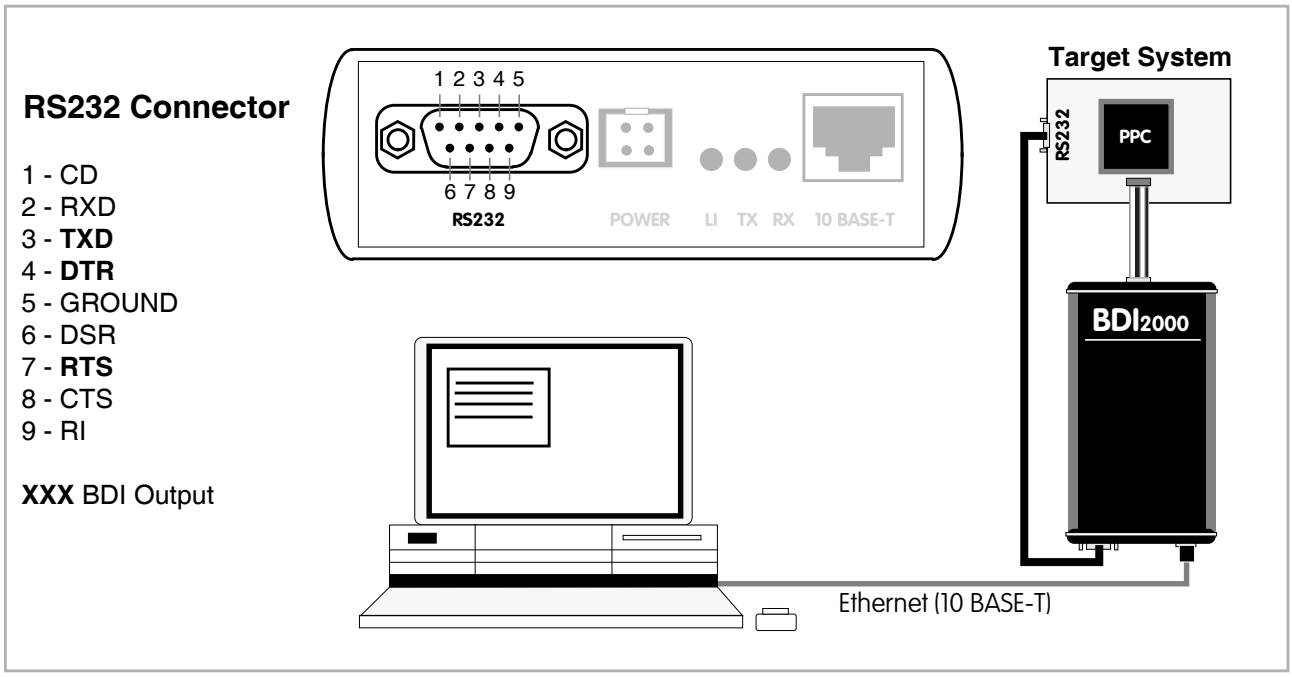
The BDI supports the GDB V5.x "monitor" command. Telnet commands are executed and the Telnet output is returned to GDB. This way you can for example switch the BDI breakpoint mode from within your GDB session.

```
(gdb) target remote bdi2000:2001
Remote debugging using bdi2000:2001
0x10b2 in start ()
(gdb) monitor 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 used framing parameters are 8 data, 1 stop and not parity.
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 Linux kernel debugging when MMU is on. The MMU configuration parameter enables this mode of operation. In this mode, all addresses received from GDB are assumed to be virtual. Before the BDI accesses memory, it translates this address into a physical one based on information found in the TLB's or, if PTBASE is defined, in kernel/user page table.

In order to search the page tables, the BDI needs to know the start addresses of the first level page table. The configuration parameter PTBASE defines the physical address where the BDI looks for the virtual/physical address of an array with two virtual/physical addresses of first level page tables. The first one points normally to the kernel page table, the second one can point to the current user page table. As long as the base pointer or the first entry is zero, the BDI does only TLB1/0 and default translation. Default translation maps a 256 Mbyte range starting at KERNELBASE to 0x00000000. The second 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 (physical address) ->
       PTE pointer pointer(virtual or physical address) ->
              PTE kernel pointer (virtual or physical address)
              PTE user pointer (virtual or physical address)
```

The pointers are assumed virtual if they are >= KERNELBASE. In that case, default translation is applied to get the physical address.

Newer versions of "arch/ppc/kernel/head.S" support the automatic update of the BDI page table information structure. Search "head.S" for "abatron" and you will find the BDI specific extensions.

Extract from the configuration file:

```
[INIT]
......
WM32    0x000000f0      0x00000000      ;invalidate page table base

[TARGET]
....
MMU           XLAT          ;translate effective to physical address
PTBASE      0x000000f0  ;here is the pointer to the page table pointers
```


To debug the Linux kernel when MMU is enabled you may use the following load and startup sequence:

- Load the compressed linux image
- Set a hardware breakpoint with the Telnet at a point where MMU is enabled. For example at "start_kernel".
  BDI> BI 0xC0061550
- Start the code with GO at the Telnet
- The Linux kernel is decompressed and started
- The system should stop at the hardware breakpoint (e.g. at start_kernel)
- Disable the hardware breakpoint with the Telnet command CI.
- If not automatically done by the kernel, setup the page table pointers for the BDI.
- Start GDB with vmlinux as parameter
- Attach to the target
- Now you should be able to debug the Linux kernel

To setup the BDI page table information structure manually, set a hardware breakpoint at "start_kernel" and use the Telnet to write the address of "swapper_pg_dir" to the appropriate place.

```
BDI>bi 0xc0061550          /* set breakpoint at start_kernel */
BDI>go
..                         /* target stops at start_kernel */
BDI>ci
BDI>mm 0xf0 0xc00000f8     /* Let PTBASE point to an array of two pointers*/
BDI>mm 0xf8 0xc0057000     /* write address of swapper_pg_dir to first pointer */
BDI>mm 0xfc 0x00000000     /* clear second (user) pointer */
```


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.

How to enter 64bit values:
The syntax for 64 bit parameters is : <high word>_<low word>
The "high word" and "low word" can be entered as decimal or hexadecimal. They are handled as two separate values concatenated with an underscore.

Examples:

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

Example of a Telnet session:

```
BDI>info
    Target CPU        : MPC8548 Rev.1
    Target state      : halted
    Debug entry cause : COP halt
    Current PC        : 0x0ffe0d1c
    Current CR        : 0x20004082
    Current MSR       : 0x00021200
    Current LR        : 0x0ffe0d4c
    Current CCSRBAR   : 0x0_e0000000
BDI>md 0xfffff000
0_fffff000 : 7c1f42a6 3c208020 60210010 7c000800  |.B.< . `!..|...
0_fffff010 : 40820020 38002000 7c11f3a6 3c401000  @.. 8. .|...<@..
               ......
```

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:

```
"PHYS  <address>              converts an effective to a physical address",
"MD    [<address>] [<count>]  display target memory as word (32bit)",
"MDD   [<address>] [<count>]  display target memory as double word (64bit)",
"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",
"MMD   <addr> <value> [<cnt>] modify double word(s) (64bit) 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",
"RDSPR <number>               display special purpose register",
"RDPMR <number>               display performance monitor register",
"RM    {<nbr>|<name>} <value> modify general purpose or user defined register",
"RMSPR <number>   <value>     modify special purpose register",
"RMPMR <number>   <value>     modify performance monitor register",

"DCACHE <addr | set>          display L1 data cache content",
"ICACHE <addr | set>          display L1 inst cache content",
"L2CACHE <set>                display L2 cache content",
"L2SRAM  <addr>               display L2 SRAM  content",
"L2TLB <from> [<to>]          display L2 TLB0 entry",
"L2CAM <from> [<to>]          display L2 TLB1 entry",
"DTLB <from> [<to>]           display L1 data TLB entry",
"DCAM <from> [<to>]           display L1 data CAM entry",
"ITLB <from> [<to>]           display L1 inst TLB entry",
"ICAM <from> [<to>]           display L1 inst CAM entry",
"WL2TLB <way> <epn> <rpn>     write to a L2 TLB0 entry",
"WL2CAM <idx> <epn> <rpn>     write to a L2 TLB1 entry",
"UPMR <MxMR> <MDR> <addr>     read selected UPM array",

"RESET [LOOP|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",
"GO    <n> <n>                start multiple cores",
"TI    [<pc>]                 trace on instruction (single step)",
"TC    [<pc>]                 trace on change of flow",
"HALT  [<n>[<n>]]             force core(s) to enter debug mode (n = core number)",
"BI  <addr>                   set instruction hardware breakpoint",
"CI  [<id>]                   clear instruction hardware breakpoint(s)",
"BD  [R|W] <addr>             set data watchpoint",
"CD  [<id>]                   clear data watchpoint(s)",
"INFO                         display information about the current state",
"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, BIN, 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",
```


The Telnet commands (cont.):

```
"DELAY  <ms>                  delay for a number of milliseconds",
"MEMACC {CORE | SAP}          select memory access mode (normally SAP)",
"SELECT <core>                change the current core",

"HOST   <ip>                  change IP address of program file host",
"PROMPT <string>              defines a new prompt string",
"QUERY [<core>]               display target configuration",
"CONFIG                       display or update BDI configuration",
"CONFIG <file> [<hostIP> [<bdiIP> [<gateway> [<mask>]]]]",
"UPDATE                       reload the configuration without a reboot",
"HELP                         display command list",
"JTAG                         switch to JTAG command mode",
"BOOT  [loader]               reboot the BDI and reload the configuration",
"QUIT                         terminate the Telnet session"
```


3.5 Dual-Core Support

The bdiGDB system supports concurrent debugging of the two e500 cores present in a dual-core processor (MPC8572, P2020, ...). For every core you can start its own GDB session. The port numbers used to attach the remote targets are 2001 and 2002.
In the Telnet you switch between the cores with the command "select {0 | 1}".

In the configuration file, simply begin the init list line with the appropriate core number. If there is no #n in front of a line, the BDI assumes core #0.

```
[INIT]
; init core register
#0 WREG    MSR             0x00001002      ;MSR : ME,RI
#0 WSPR    1008            0x00000000      ;HID0:
;
#1 WREG    MSR             0x00001002      ;MSR : ME,RI
#1 WSPR    1008            0x00000000      ;HID0:
```

The BDI supports different startup modes. The startup mode is defined via an entry in the [TARGET] section of the BDI configuration file. The second e500 core (core #1) is handled by the BDI only if there is a second "mode" parameter present in the STARTUP line. Because after reset the second core maybe disabled, the BDI writes to the EEBPCR and enables it in cases when HALT or LOOP is selected as startup mode for the second core. Following some examples:

STARTUP HALT HALT or STARTUP LOOP LOOP
The second core will be enabled via EEBPCR and both core are halted at the reset vector via an IABR breakpoint.

STARTUP RUN RUN
Both core are let running after reset. You can halt them individually via the Telnet "halt" command. The BDI does not write to EEBPCR. Halting the second core will only succeed if it has been enabled out of reset or form the code running on the first core. The init list is not processed in this case.

STARTUP STOP 4000 HALT
The second core will be enabled via EEBPCR and halted at the reset vector. The first core is let running for 4 seconds and the halted.

STARTUP STOP 4000 STOP
Both core are let running after reset for 4 seconds and then halted. The BDI does not write to EEBPCR. Halting the second core will only succeed if it has been enabled out of reset or form the code running on the first core during this 4 second runtime. After halting, the init list is processed.

STARTUP HALT RUN
Useful if you want to debug boot code on core#0 but want to be able to access core#1 later. The BDI does not write to EEBPCR in this case.

STARTUP RUN HALT
The first core is let running while the second core will be enabled via EEBPCR and halted at the reset vector.

STARTUP HALT WAIT
The second core is halted once enable by the first core or by writing to EEBPCR.

SMP Mode:
In this mode the BDI halts also the other core when one core halts because of a breakpoint.
Also in SMP mode a "continue" command from GDB is handled different. If only one core (either core#0 or core#1) is connected to a GDB session then a "continue" command from GDB always starts both cores. If both cores are connected to GDB sessions then the first "continue" from either GDB prepares the attached core for restart but the final step to actually restart is made pending. Then a "continue" command from the GDB session assigned to the other core prepares also this core for restart and finally both cores are restarted with a minimal delay.

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 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. 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)

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 the Hardware to be free of defects in materials and workmanship for a period of 3 years following the date of purchase when used under normal conditions. In the event of notification within the warranty period of defects in material or workmanship, ABATRON will repair or replace the defective hardware. The cost for the shipment to Abatron must be paid by the customer. Failure in handling which leads to defects are not covered under this warranty. The warranty is void under any self-made repair operation.

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 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

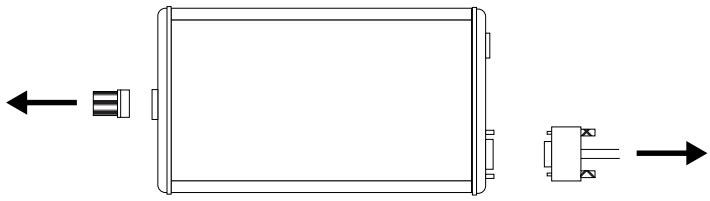
The BDI needs no special maintenance. Clean the housing with a mild detergent only. Solvents such as gasoline may damage it.

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

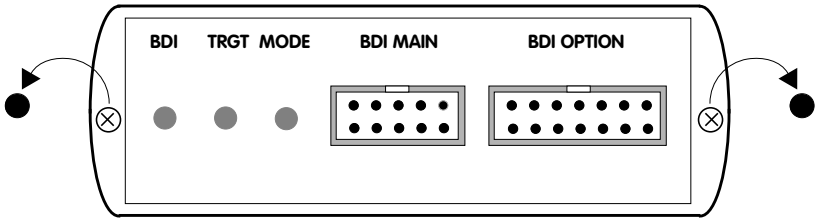
1.1 Unplug the cables



2

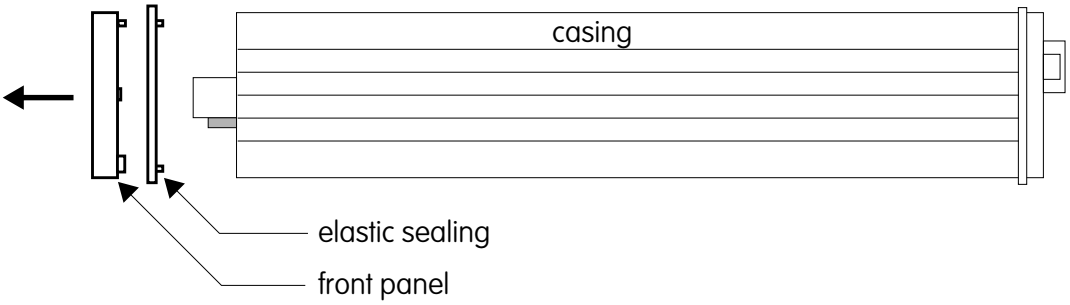
2.1 Remove the two plastic caps that cover the screws on target front side (e.g. with a small knife)

2.2 Remove the two screws that hold the front panel



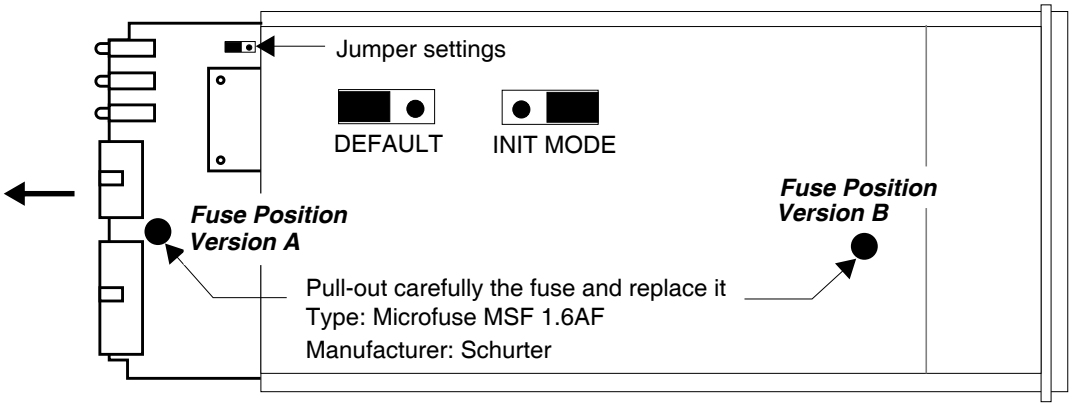
3

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



4

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

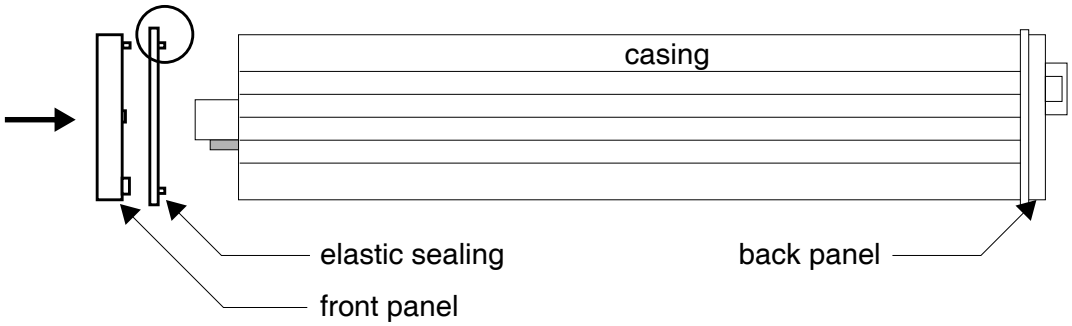


5

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 elastig 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.

