bdiGDB

JTAG debug interface for GNU Debugger

QorIQ P3/P4/P5/T1/T2/T4

User Manual

Manual Version 1.08 for BDI3000

abatron

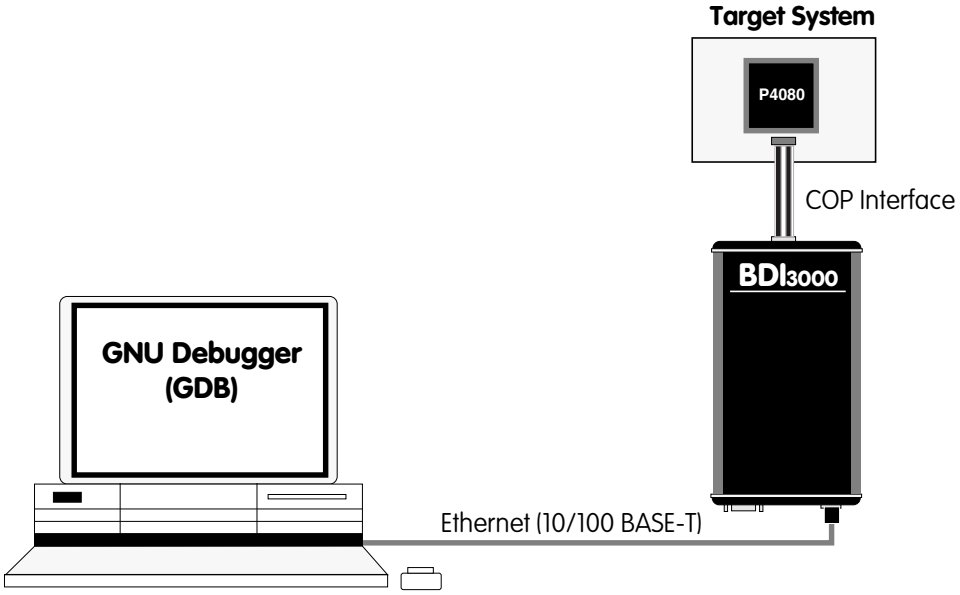
Appendices

1 Introduction

bdiGDB enhances the GNU debugger (GDB), with JTAG/COP debugging for QorIQ P4 based targets. With the built-in Ethernet interface you get a very fast code download speed. No target communication channel (e.g. serial line) is wasted for debugging purposes. Even better, you can use fast Ethernet debugging with target systems without network capability. The host to BDI communication uses the standard GDB remote protocol.

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

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



1.1 BDI3000

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

1.2 BDI Configuration

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

Following an example of a typical configuration file:

```
;bdiGDB configuration file for P4080-DS
;-------------------------------------
;
[INIT]
; Initialize LAWBAR's
WM32    0xfe000c00      0x00000000      ;LAWBAR0 : Flash @0_e0000000
WM32    0xfe000c04      0xe0000000
WM32    0xfe000c08      0x81f0001b      ;LAWAR0  : eLBC 256MB

       .......

; Release cores for booting
WM32    0xfe0E00E4      0x00000003      ;BRR: release core 0 and 1
;
;
[TARGET]
; common parameters
POWERUP     5000                ;start delay after power-up detected in ms
JTAGCLOCK   1                   ;use 16 MHz JTAG clock
WAKEUP      1000                ;give reset time to complete
;
; CoreID#0 parameters (active vCPU after reset)
#0 CPUTYPE      P4080 0 0       ;Core0 / SOC0
#0 STARTUP      HALT            ;halt at the reset vector (this halts all cores !!!)
;
; CoreID#1 parameters
#1 CPUTYPE      P4080 1 0       ;Core1 / SOC0
#1 STARTUP      HALT            ;halt at the reset vector
;

[HOST]
IP          151.120.25.112
FILE        E:\temp\dump1024k.bin
FORMAT      BIN 0x10000
;
#0 PROMPT      P4080#0>
#1 PROMPT      P4080#1>
;

[FLASH]
; only to test execution of target code
WORKSPACE   0x80001000          ;workspace in CPC1/SRAM
CHIPTYPE    AM29BX16            ;Flash type
CHIPSIZE    0x00200000          ;The size of one flash chip in bytes
BUSWIDTH    16                  ;The width of the flash memory bus in bits
FILE        E:\temp\dump16k.bin
FORMAT      BIN 0x00300000

[REGS]
FILE        $regP4080.def
```

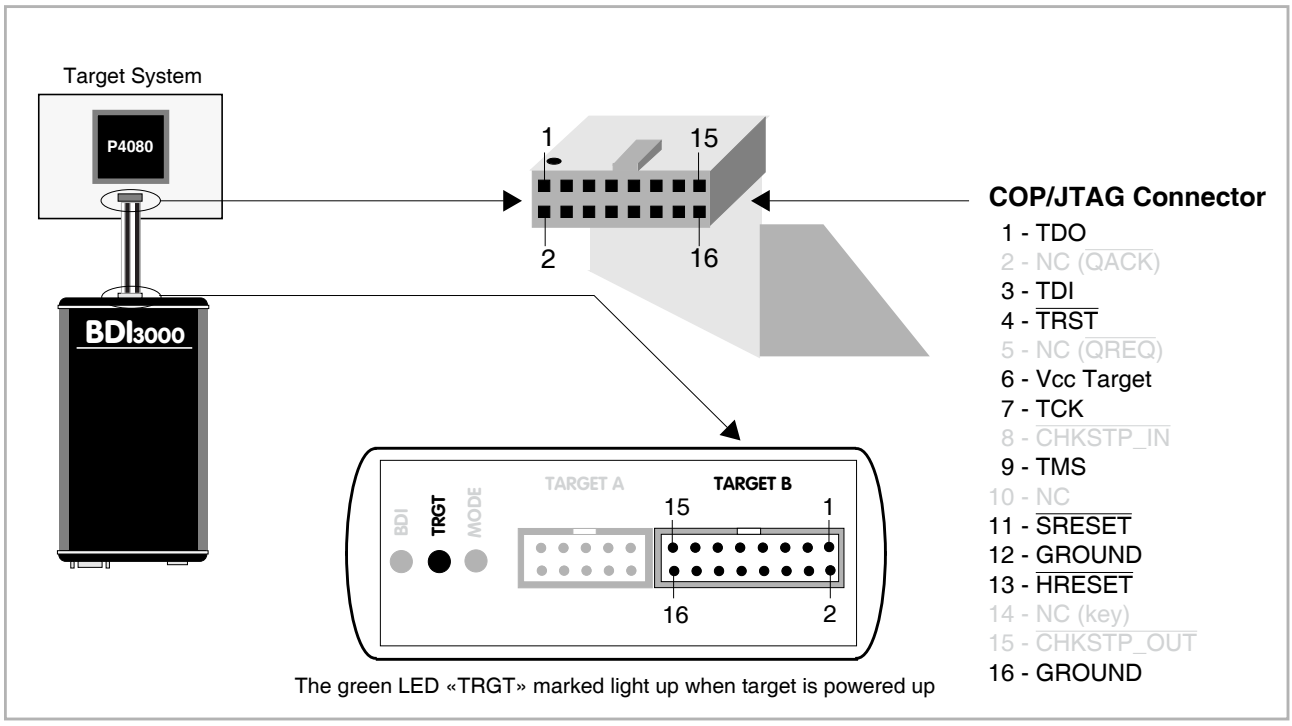
Based on the information in the configuration file, the target is automatically initialized after every reset.

2 Installation

2.1 Connecting the BDI3000 to Target

The 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 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

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

Note:
For critical designs (long traces on the target board) there is a shorter target cable available (p/n 90020-S).This may improve JTAG communication reliability. But best is to keep the JTAG traces on the board as short as possible.

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

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

BDI TARGET B Connector Signals:

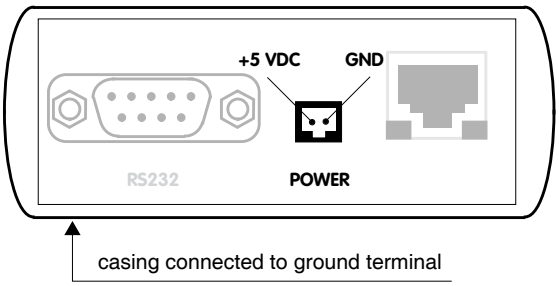
Pin	Name	Describtion
1	TDO	**JTAG Test Data Out** This input to the BDI3000 connects to the target TDO pin.
2	IO2	**General purpose I/O** Currently not used.
3	TDI	**JTAG Test Data In** This output of the BDI3000 connects to the target TDI pin.
4	$\overline{\text{TRST}}$	**JTAG Test Reset** This output of the BDI3000 resets the JTAG TAP controller on the target.
5	IN0	**General purpose Input** Currently not used.
6	Vcc Target	**1.2 – 5.0V:** This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally connected to Vdd I/O on the target board.
7	TCK	**JTAG Test Clock** This output of the BDI3000 connects to the target TCK pin.
8	IO8	**General purpose I/O** This output of the BDI3000 connects to the target CKSTP_IN pin. Currently not used.
9	TMS	**JTAG Test Mode Select** This output of the BDI3000 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 BDI3000 connects to the target HRESET pin.
12	GROUND	**System Ground**
13	$\overline{\text{HRESET}}$	**Hard-Reset** This open collector output of the BDI3000 connects to the target PORESET pin.
14	<reseved>	
15	IN1	**General purpose Input** This input to the BDI3000 connects to the target CKSTP_OUT pin. Currently not used.
16	GROUND	**System Ground**

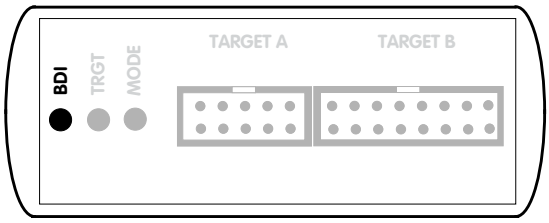
2.2 Connecting the BDI3000 to Power Supply

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

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

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





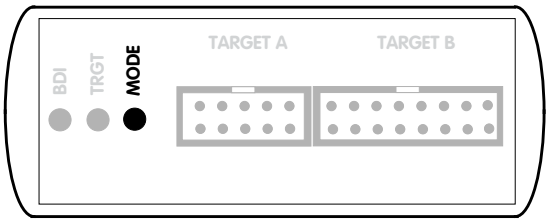
The green LED «BDI» marked light up when 5V power is connected to the BDI3000

Please switch on the system in the following sequence:

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

2.3 Status LED «MODE»

The built in LED indicates the following BDI states:



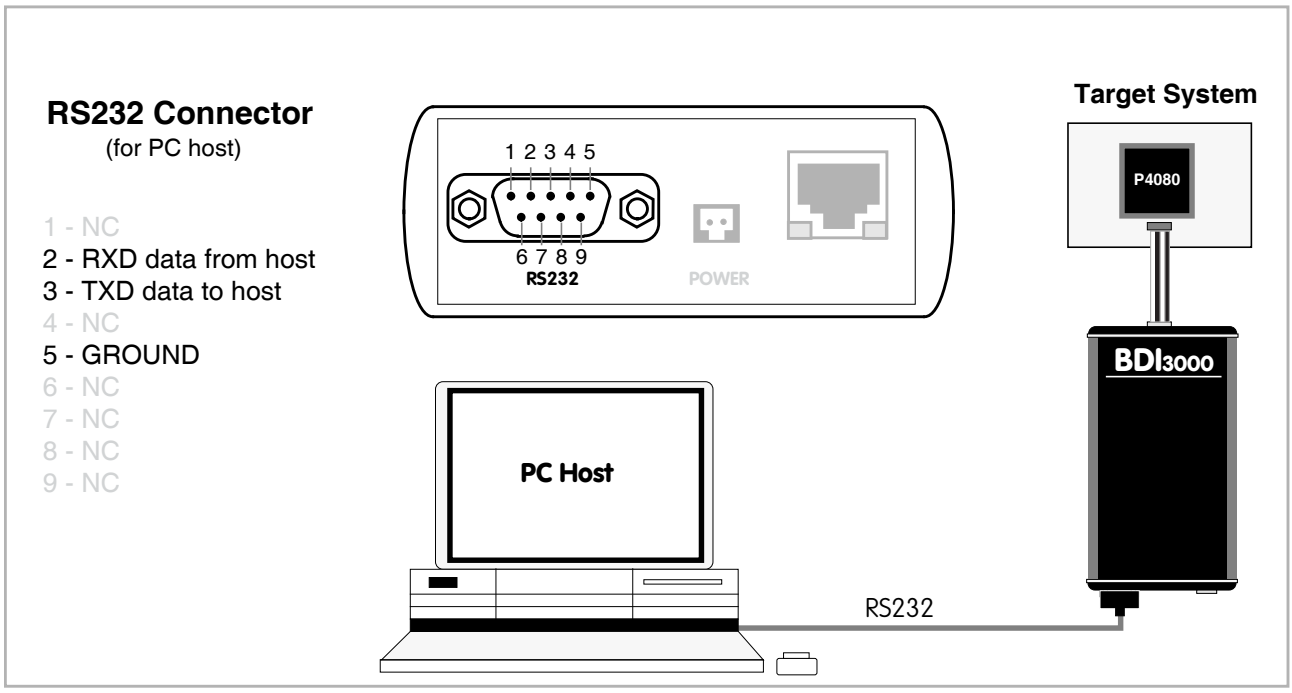
MODE LED	BDI STATES
OFF	The BDI is ready for use, the firmware is already loaded.
ON	The output voltage from the power supply is too low.
BLINK	The BDI «loader mode» is active (an invalid firmware is loaded or loading firmware is active).

2.4 Connecting the BDI3000 to Host

2.4.1 Serial line communication

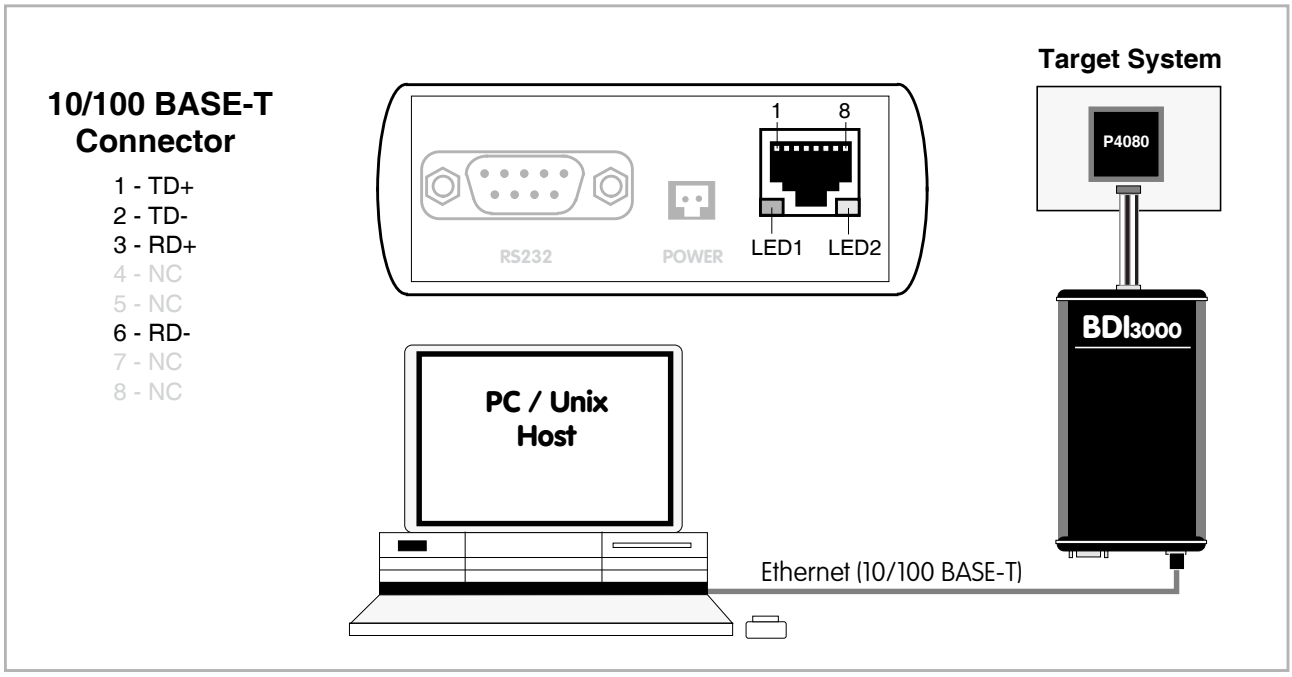
Serial line communication is only used for the initial configuration of the bdiGDB system.

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



2.4.2 Ethernet communication

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



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

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

2.5 Installation of the Configuration Software

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

The following files are on the diskette.

b30qp4gd.exe	Windows Configuration program
b30qp4gd.xxx	Firmware for the BDI3000
tftpsrv.exe	TFTP server for Windows (WIN32 console application)
*.cfg	Configuration files
*.def	Register definition files
bdisetup.zip	ZIP Archive with the Setup Tool sources for Linux / UNIX hosts.

Overview of an installation / configuration process:

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

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

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

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

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

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

2.5.1 Configuration with a Linux / Unix host

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

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

Following the steps to bring-up a new BDI3000:

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

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

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

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

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

```
$ ./bdisetup -u -p/dev/ttyS0 -b115 -aGDB -tP4080
Connecting to BDI loader
Programming firmware with ./b30qp4gd.100
Erasing firmware flash ....
Erasing firmware flash passed
Programming firmware flash ....
Programming firmware flash passed
```


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

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

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

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

```
$ ./bdisetup -v -p/dev/ttyS0 -b115 -s
BDI Type : BDI3000 (SN: 30000154)
Loader   : V1.00
Firmware : V1.00 bdiGDB for P4080
MAC      : 00-0c-01-30-00-01
IP Addr  : 151.120.25.102
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 151.120.25.112
Config   : /bdi3000/mytarget.cfg
```

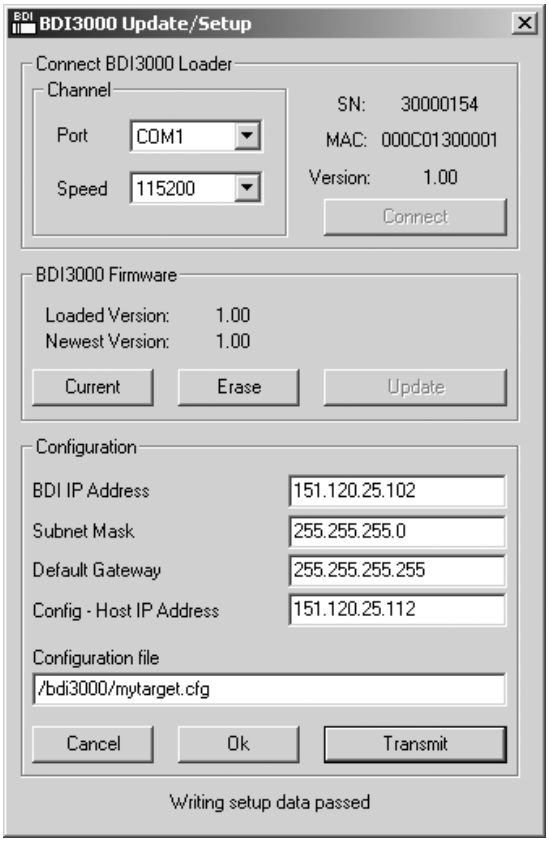
The Mode LED should go off, and you can try to connect to the BDI via Telnet.

```
$ telnet 151.120.25.102
```


2.5.2 Configuration with a Windows host

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

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



dialog box «BDI3000 Update/Setup»

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

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

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

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

2.5.3 Configuration via Telnet / TFTP

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

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

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

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

Update the network parameters so it matches your needs:

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

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


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

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

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

```
LDR>boot loader
```


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

```
$ telnet 151.120.25.102

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

LDR>fwload e:/temp/b30qp4gd.100
erasing firmware flash ... passed
programming firmware flash ... passed

LDR>info
    BDI Firmware: 23 / 1.00
    BDI CPLD ID : 01285043
    BDI CPLD UES: ffffffff
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg
LDR>
```

To boot now into the firmware use:

```
LDR>boot
```

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

```
telnet 151.120.25.102
```


2.6 Testing the BDI3000 to host connection

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

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

2.7 TFTP server for Windows

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

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

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

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

`tftpsrv p`	Starts the TFTP server and enables protocol output
`tftpsrv p w`	Starts the TFTP server, enables protocol output and write accesses are allowed.
`tftpsrv dC:\tftp\`	Starts the TFTP server and allows only access to files in C:\tftp and its subdirectories. As file name, use relative names. For example "bdi\mpc8548.cfg" accesses "C:\tftp\bdi\mpc8548.cfg"

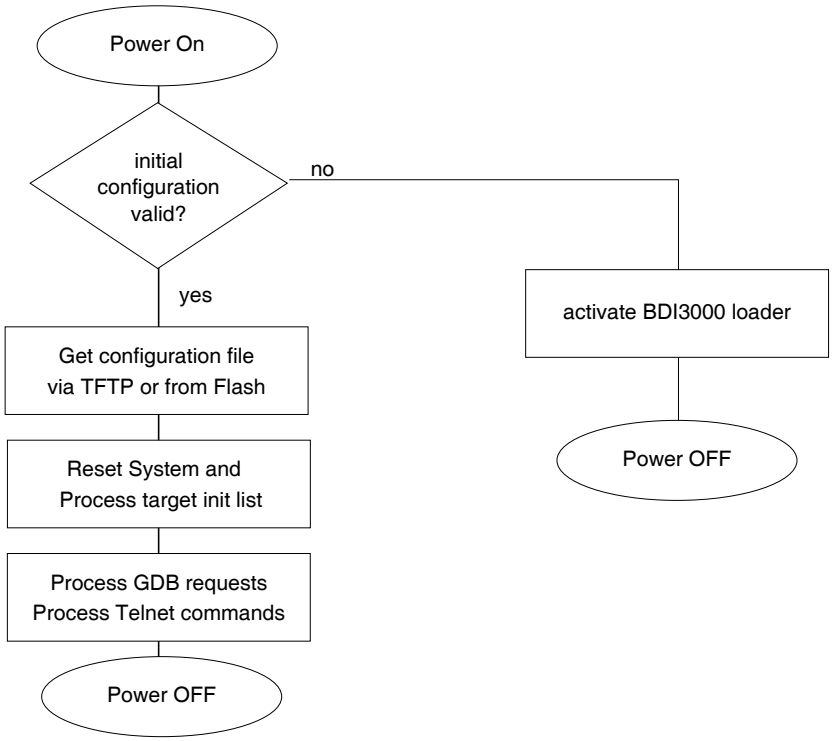
You may enter the TFTP server into the Startup group so the server is started every time you login.

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 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: 0xnnnnnnnnnnnnnnn

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

WDCSR address value
Write value to the selected register in DCSR space
address address / offset in DCSR space
value the value to write into the register
Example: WDCSR 0x2220c 0x0000000e ;CGCR1: Core Group 1

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.
address the memory address
value the value to write to the target memory
Example: WM64 0xFFF00000 0x123456789abcdef0

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

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 [n_]opcode [data] — This entry causes the processor to execute one instruction. If a load instruction should get the data from the JTAG port instead from normal memory, the optional n_ and data parameters are used.
n 0 = load data from normal memory
1 = load 8/16/32-bit data from JTAG port
2 = load 64-bit data from JTAG port
opcode the opcode of the PowerPC instruction
data the data to present to the instruction
Example:
```
EXEC 0_0x3c60aba4      ;load GPR 3 via lui
EXEC 1_0x80c00000 1234 ;load GPR 6 via lwz
EXEC 2_0xc8c00000 0    ;clear FPR 6 via ldf
```


WTLB mas1_mas2 mas0/mas7_mas3

Adds an entry to the TLB0 or TLB1 array. The two 64-bit values of an init list entry are used to define MAS0 (upper 16 bits), MAS1, MAS2, MAS3 and MAS7 (lower 16 bits) before a tlbwe instruction is executed. If other MASx registers need a special value, use the WSPR init list entry. A TLB entry can also be addd via a Telnet command (enter WTLB at the Telnet for a description).

mas1	value to load into MAS1
mas2	value to load into MAS2 (only lower 32 bits)
mas3	value to load into MAS3
mas0	value to load into upper 16-bits of MAS0
mas7	value to load into lower 16-bits of MAS7

Some examples how to write TLB entries:

```
; Setup TLB1
;           MAS1       MAS2       MAS0/MAS7     MAS3
WTLB    0x80000700_0xfe00000a   0x10010000_0xfe00003f   ;1/1: fe000000->0_fe000000  16MB -I-G- RWXRWX
WTLB    0x80000900_0xe000000a   0x10020000_0xe000003f   ;1/2: e0000000->0_e0000000 256MB -I-G- RWXRWX
WTLB    0x80000a00_0x00000000   0x10030000_0x0000003f   ;1/3: 00000000->0_00000000   1GB ----- RWXRWX
WTLB    0x80000a00_0x40000000   0x10040000_0x4000003f   ;1/4: 40000000->0_40000000   1GB ----- RWXRWX
WTLB    0x80000500_0x80000000   0x10050000_0x8000003f   ;1/5: 80000000->0_80000000   1MB ----- RWXRWX

; Setup TLB0
;           MAS1       MAS2       MAS0/MAS7     MAS3
WTLB    0x80000100_0xc0000000   0x00000000_0x0000003f   ;WAY0: c0000000->0_00000000   4KB ----- RWXRWX
WTLB    0x80000100_0xc0001000   0x00000000_0x0000103f   ;WAY0: c0001000->0_00001000   4KB ----- RWXRWX
WTLB    0x80000100_0xc0002000   0x00000000_0x0000203f   ;WAY0: c0002000->0_00002000   4KB ----- RWXRWX
WTLB    0x80000100_0xc0003000   0x00000000_0x0000303f   ;WAY0: c0003000->0_00003000   4KB ----- RWXRWX
;
```

In order to set the upper 32 bits of MAS2 in 64-bit mode write directly to MAS2:

```
WREG    MAS2                    0x00000064_0x00000000   ;set MAS2 upper word
WTLB    0x80000700_0xff00000a   0x10060000_0xff00003f   ;1/1: 64_ff000000->0_ff000000  16MB -I-G- RWXRWX
```

MAS2 is a predefined register name but WSPR will also work.

```
WSPR    626                     0x00000064_0x00000000   ;set MAS2 upper word
WTLB    0x80000700_0xff00000a   0x10060000_0xff00003f   ;1/1: 64_ff000000->0_ff000000  16MB -I-G- RWXRWX
```


3.2.2 Part [TARGET]

The part [TARGET] defines some target specific values.

CPUTYPE type core [soc] This value gives the BDI information about the connected CPU/core.

type	P2040, P3041, P4040, P4080, B4860, B4420 P5010, P5020, P5021, P5040, T1020, T1040, T2080, T4240, T4160
core	the core/thread number within the SOC (0...n)
soc	the SOC number (0...3)
Example:	CPUTYPE P4080 0 0 ; Core0 / SOC0 CPUTYPE P4080 1 ; Core1 / SOC0

JTAGCLOCK value With this value you select the JTAG clock frequency.

value The JTAG clock frequency in Hertz or an index value from the following table:

0 = 32 MHz	5 = 4 MHz	10 = 50 kHz
1 = 16 MHz	6 = 1 MHz	11 = 20 kHz
2 = 11 MHz	7 = 500 kHz	12 = 10 kHz
3 = 8 MHz	8 = 200 kHz	13 = 5 kHz
4 = 5 MHz	9 = 100 kHz	

Example: CLOCK 1 ; JTAG clock is 16 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 drives the HRESET line during startup. If reset type is NONE, the BDI does not assert a hardware reset during startup. This entry can also be used to change the default reset time.

type	NONE HARD (default) KEEP keep HRESET assert 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

EDBCR0 list This parameter allows to change the default EDBCR0 value. By default the EDM, DNH and EFT bits are set.

list defines the bits to set (EDM, DN, EFT, DNI, CTH)

Example:
```
EDBCR0 EDM DNH EFT ;this is the default
EDBCR0 EDM DNH     ;do not freeze timers
EDBCR0 DNH         ;do not halt on debug events
```


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]

This parameter selects the target startup mode.
The following modes are supported:

HALT	This mode forces the target to debug mode immediately out of reset. If HALT for core number 0 is defined, then all cores within an SOC will be halted immediatelly out of reset.
STOP	In this mode, the BDI lets the target execute code for "runtime" milliseconds after reset. This mode is useful when monitor code should initialize the target system.
RUN	After reset, the target executes code until stopped by the Telnet "halt" command.
Example:	STARTUP STOP 3000 ; let the CPU run for 3 seconds

BREAKMODE mode

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

SOFT	This is the normal mode. Breakpoints are implemented by replacing code with a DNH instruction.
HARD	In this mode, the target 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).

ICMP	This is the mode, single step is implemented via the instruction complete (ICMP) debug event.
HWBP	In this mode, a hardware breakpoint on the next instruction is used to implement single stepping.
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

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

MEMACCES mode [attr] [SNOOP | NOSNOOP]

There are two possible ways to access memory. Via the current core by executing ld/st instructions or via the System Access Port (SAP).
See also Telnet chapter. The following modes are supported:

SAP	Memory access via SAP (default). The attr is a delay sometimes necessary when accessing slow memory. Accesses maybe snooped (default) or not snooped.
CORE	Memory access via current core. The optional attr parameter is explained in the Telnet chapter.
Example:	MEMACCES CORE ; access via core MEMACCES SAP 100 ; access via SAP, 100us delay

REGLIST list

This parameter defines the transferred GDB registers packet. By default STD and FPR are read and transferred in 32-bit mode. The following names are use to select a register group and register size:

STD	The standard register block. The FP registers are not read from the target. Placeholders are transferred.
FPR	The floating point registers are read and transferred.
64BIT	The register packet is sent as expected by GDB for a 64-bit PowerPC target.
Example:	REGLIST STD ; standard registers in 32-bit mode REGLIST STD FPR 64BIT ;use 64-bit mode

SDCRESP response

If this parameter is present the BDI tries to open JTAG debugging via the challenge/response operation of the secure debug controller (SDC).

response	The 64-bit response value the BDI writes to the Secure Debug Response register in order to allow JTAG debugging.
Example:	SDCRESP 0x01aa00bbcafeca77

CGROUP cores

This parameter gives the BDI information about how to restart multiple cores at the same time in response to a GDB continue command. See chapter Multi-Core Support.

cores	The selected cores as bitmap.
Example:	#0 CGROUP 0x0f ;GDB continue core group (restart) #1 CGROUP 0x02 ;GDB continue core group (prepare)

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

Overriding Reset Configuration Word (RCW):
The BDI supports overriding the RCW Source and also overriding individual RCW values. If there is no valid RCW present at the currently via pin selected RCW Source a Hard-coded RCW should be selected.

RCWSRC source — Defines a new RCW Source to be used.

source	The RCW Source number (0x000 ... 0x1FF)
Example:	RCWSRC 0x18 ;Hard-Coded RCW 1_1000

RCWOVR index data — Override the value of an individual 32-bit RCW.

index	The RCW index (0 ...15, means RCW1...RCW16)
data	The new RCW value
Example:	RCWOVR 11 0x00830000 ;RCW bits 352-383

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, 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 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

PROMPT string

This entry defines a new Telnet prompt. The current prompt can also be changed via the Telnet interface.

Example: PROMPT P4080#0>

DUMP filename

The default file name used for the Telnet DUMP command.

filename the filename including the full path

Example: DUMP dump.bin

TELNET mode

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

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

DEBUGPORT port [RECONNECT]

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

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 or ELF).

format	SREC, BIN or ELF
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
```

For MIRRORX16 and S29GLSX16 the "unlock" command erases the PPB bits. Use this command only for Spansion devices where Persistent Protection Bits (PPB) are implemented.

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
PMR	Performance monitor register
CCSR	Relative to CCSRBAR memory mapped register.
DCSR	Memory mapped register in DCSR 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\regP4080.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, pc, pc64, iar, iar64** and **fpscr** and are predefined.

Example for a register definition:

Entry in the configuration file:

[REGS]
FILE $regP4080.def

The register definition file:

```
;name         type   addr          size
;-----------------------------------------
;
sp            GPR    1
;
;
atbl          SPR    526
atbu          SPR    527
bucsr         SPR    1013
csrr0         SPR    58
csrr1         SPR    59

       .....

tlb0cfg       SPR    688
tlb1cfg       SPR    689
tsr           SPR    336
usprg0        SPR    256
xer           SPR    1
;
;
;       Local Bus Controller
br0           CCSR    0x124000
br1           CCSR    0x124008
br2           CCSR    0x124010
br3           CCSR    0x124018

              ....

fcr           CCSR    0x1240E8
fbar          CCSR    0x1240EC
fpar          CCSR    0x1240F0
fbcr          CCSR    0x1240F4
```

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 GDB server runs within the 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.

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 optionally loads your application code. BDI now waits for GDB request from the debugger running on the host.

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

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

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

If not already suspended, this stops the execution of application code and the target CPU changes to background debug mode.
Remember, every time the application is suspended, the target CPU is freezed. During this time, no hardware interrupts will be processed.

Note: For convenience, the GDB detach command triggers a target reset sequence in the BDI.
(gdb)detach
... Wait until BDI has resetet the target and reloaded the image
(gdb)target remote bdi3000:2001

3.3.3 GDB monitor command

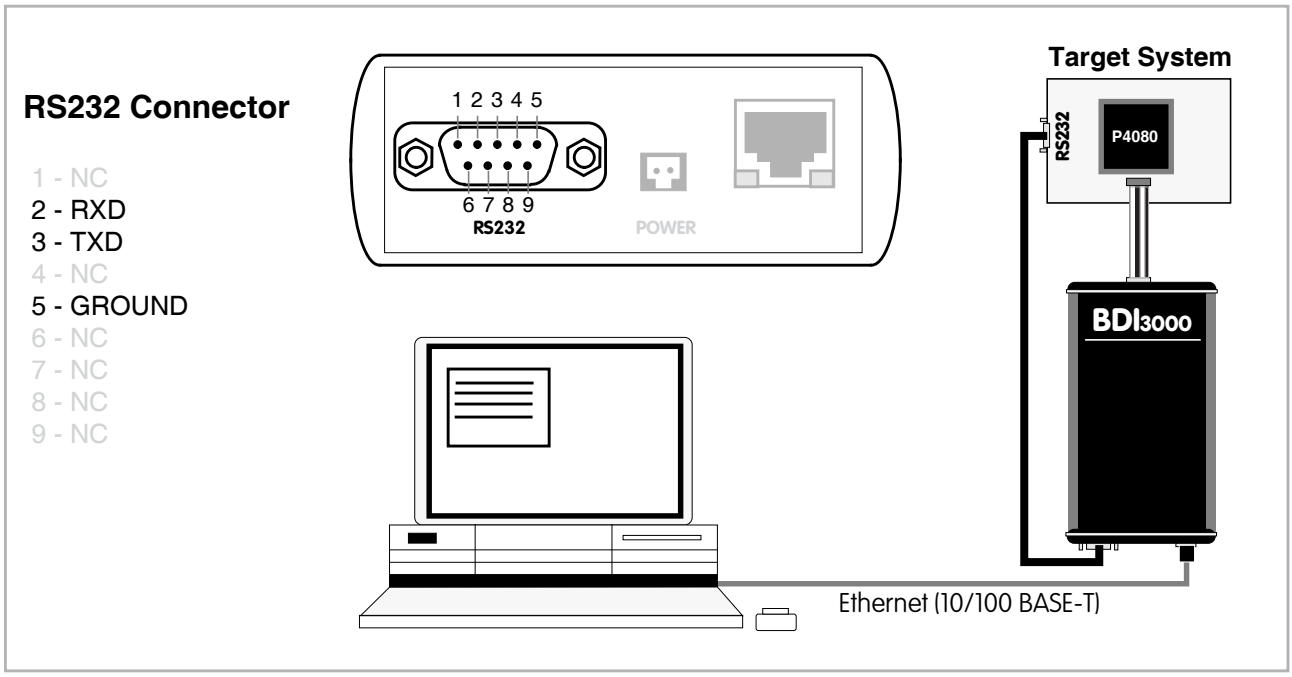
The BDI supports the GDB V5.x "monitor" command. Telnet commands are executed and the Telnet output is returned to GDB. This way you can for example switch the BDI breakpoint mode from within your GDB session.

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

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


3.3.4 Target serial I/O via BDI

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



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

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

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

3.3.5 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 kernel/user page table.

If PTBASE is not defined, the BDI does TLB1, TLB0 and if enabled default translation (in this order).

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 L2 CAM (L2 TLB1) 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.

Example of a Telnet session:

```
P4080#0>info
    Target CPU        : P4080 Core#0
    Core state        : halted
    Debug entry cause : device event
    Current PC        : 0xfffffffc
    Current CR        : 0x00000000
    Current MSR       : 0x00000000
    Current LR        : 0x00000000
    Current CCSRBAR   : 0x0_fe000000
P4080#0>rd
GPR00: c11bc002 06278553 80028188 aba40000
GPR04: 609db195 ad1944b2 deadbeef 002883a6
GPR08: 20119520 2032dc90 94110404 29038003
GPR12: 9422cf8e 0c105000 613b00b0 4e0d4548
GPR16: 0f15d163 3820d4a3 806b42d8 4c005402
GPR20: b0010949 846310d8 c0d53502 4c41d854
GPR24: c0602409 4443cd98 a8911575 e0021810
GPR28: 200842c0 c890cc15 2c2390ce 604bc0c1
CR   : 00000000     MSR: 00000000
P4080#0>md 0
0_00000000 : deadbeef deadbeef deadbeef deadbeef  ................
0_00000010 : deadbeef deadbeef deadbeef deadbeef  ................
0_00000020 : deadbeef deadbeef deadbeef deadbeef  ................
0_00000030 : deadbeef deadbeef deadbeef deadbeef  ................
0_00000040 : deadbeef deadbeef deadbeef deadbeef  ................
               ......
```

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",
"RDFPR                        display floating point registers",
"RDVR                         display vector registers",
"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",
"RMVR  <nbr><val val val val> modify vector register (four 32bit values)",

"RDCSR <addr>  [<count>]      display register(s) in DCSR space (Run Control)",
"WDCSR <addr>   <value>       write to a register in DCSR space (Run Control)",

"DCACHE <addr | set>          display L1 data cache content",
"ICACHE <addr | set>          display L1 inst cache content",
"L2CACHE <set> [<bank> [XOR]] display L2 cache content",
"TLB0  <from> [<to>]          display L2 TLB0 entry",
"TLB1  <from> [<to>]          display L2 TLB1 entry",
"WTLB0 <way> <epn> <rpn>      write to a L2 TLB0 entry",
"WTLB1 <idx> <epn> <rpn>      write to a L2 TLB1 entry",

"EXEC  <inst> [<r0> [<r1>]]   execute an instruction",
"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>                restart multiple cores (<cores> = core bit map)",
"TI    [<pc>]                 trace on instruction (single step)",
"TC    [<pc>]                 trace on change of flow",
"HALT  [<cores>]              force core(s) to debug mode (<cores> = core bit map)",

"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 core",
"STATE                        display information about all cores",

"LOAD   [<offset>] [<file> [<format>]] load program file to target memory",
"VERIFY [<offset>] [<file> [<format>]] verify a program file to target memory",
"PROG   [<offset>] [<file> [<format>]] program flash memory",
"                                      <format> : SREC, BIN, AOUT or ELF",
```


The Telnet commands (cont.):

```
"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",
"MEMACC {CORE | SAP} [<attr>] 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"
```

There are two memory access modes implemented. The default is via System Access Port (SAP). Via SAP physical addresses are used and the access does not make use of any of the cores. SAP accesses memory like an additional bus master. If memory access CORE is selected, the current core executes load/store instructions in its current context. In this mode MMU translation takes place unless the use of real addresses is forced.

For memory accesses via core the <attr> parameter in the Telnet MEMACC command has the following meaning (default is 0):

```
+-+-+-+-+-+-+-+-+
|R|-|W|I|M|G|E|-|
+-+-+-+-+-+-+-+-+

R    : 1 = Real Addressing Mode (no MMU translation)
WIMGE: Page attributes for load/store instruction, only used if R = 1
```

For example to access the real address 0x3_8400_0000 with I and G set via the current core:

```
BDI> memacc core 0x94
BDI> md 0x384000000 1
```

For memory accesses via SAP the <attr> defines a delay sometimes necessary when accessing slow memory. The following example define a 100us delay during memory accesses via SAP.

```
BDI> memacc sap 100
```

Memory access mode is a global selection. It is not possible to select different modes for different cores.

Note:
For information about the registers in DCSR space please contact Freescale.

3.5 Multi-Core Support

The bdiGDB system supports concurrent debugging of up to 32 cores/threads. For every core you can start its own GDB session. The default port numbers used to attach the remote targets are 2001 ... 2032. In the Telnet you switch between the cores with the command "select <0..31>". 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 4 cores for debugging. For a complete example, look at the configuration examples.

```
[TARGET]
; common parameters
POWERUP     5000                ;start delay after power-up detected in ms
JTAGCLOCK   16000000            ;use 16 MHz JTAG clock
WAKEUP      200                 ;give reset time to complete
;
;=========================================================
; !!!! define the core ID (the #x) without any holes !!!!
; !!!! no need that core ID matches the core number  !!!!
; !!!! A valid example is: #1 CPUTYPE  P4080 5 0     !!!!
;=========================================================
;
; Core#0 parameters (active core after reset)
#0 CPUTYPE      P4080 0 0       ;Core0 / SOC0
#0 STARTUP      STOP 5000       ;let U-boot setup the system
#0 MEMACCESS    CORE
#0 CGROUP       0x0f            ;GDB continue core group (resume)
;
; Core#1 parameters
#1 CPUTYPE      P4080 1 0       ;Core1 / SOC0
#1 STARTUP      RUN             ;let core run
#1 MEMACCESS    CORE
#1 CGROUP       0x02            ;GDB continue core group (prepare)
;
; Core#2 parameters
#2 CPUTYPE      P4080 2 0       ;Core2 / SOC0
#2 STARTUP      RUN             ;let core run
#2 MEMACCESS    CORE
#2 CGROUP       0x04            ;GDB continue core group (prepare)
;
; Core#3 parameters
#3 CPUTYPE      P4080 3 0       ;Core3 / SOC0
#3 STARTUP      RUN             ;let core run
#3 MEMACCESS    CORE
#3 CGROUP       0x08            ;GDB continue core group (prepare)
;

[HOST]
#0 PROMPT       P4080#0>
#1 PROMPT       P4080#1>
#2 PROMPT       P4080#2>
#3 PROMPT       P4080#3>
;
```

Note:
Be aware that via Telnet you select the core/thread by its BDI core ID (#n). This BDI core ID is not necessary the core/thread number within the SOC. Assuming there are two P4040 daisy chained, you may use BDI core ID #4 to select core 0 in the second P4040.

Multi-Core related Telnet commands:

STATE	Display information about all cores.
SELECT <core>	Change the current Telnet core
CONT <cores>	Restart one or multiple cores <cores> core bit map Example: cont 0x000d ; restart core #0, #2, #3
HALT [<cores>]	Force one or multiple cores to debug mode. If there is no <cores> parameter, the currently selected core is forced to debug mode. <cores> core bit map Example: halt 0x00ff ; halt 8 cores #0...#7

Telnet session:

```
P4080#0>info
    Target CPU        : P4080 Core#0
    Core state        : halted
    Debug entry cause : debug halt
    Current PC        : 0x7ff74c34
    Current CR        : 0x22000084
    Current MSR       : 0x00029200
    Current LR        : 0x7ff74c38
    Current CCSRBAR   : 0x0_fe000000
P4080#0>state
Core#0: halted 0x7ff74c34 debug halt
Core#1: running
Core#2: running
Core#3: running
Core#4: running
Core#5: running
Core#6: running
Core#7: running
P4080#0>halt 0xf0
- TARGET: core #4 has entered debug mode
- TARGET: core #5 has entered debug mode
- TARGET: core #6 has entered debug mode
- TARGET: core #7 has entered debug mode
P4080#0>state
Core#0: halted 0x7ff74c34 debug halt
Core#1: running
Core#2: running
Core#3: running
Core#4: halted 0xfffff0f8 debug halt
Core#5: halted 0xfffff0f8 debug halt
Core#6: halted 0xfffff0f8 debug halt
Core#7: halted 0xfffff0f8 debug halt
P4080#0>cont 0x30
P4080#0>state
Core#0: halted 0x7ff74c34 debug halt
Core#1: running
Core#2: running
Core#3: running
Core#4: running
Core#5: running
Core#6: halted 0xfffff0f8 debug halt
Core#7: halted 0xfffff0f8 debug halt
P4080#0>
```


Multi-Core Restart via GDB continue:

Then core specific parameter CGROUP allows to define a group of cores that should be restarted when GDB sends the "continue" command to the BDI. This has the same effect as the Telnet "cont" command.To halt a group of cores use the Cross-Trigger functions of the processor. Have a look at the configuration example below.There are two alternatives, one using a device event and the other more complex one using an EPU event. Via the new CGROUP parameter you define what the BDI does in response to the GDB continue command:

- If there is no CGROUP defined then the core is restarted as usual.
- If the CGROUP core mask defines only the actual core then this core is prepared for restart but the final step to actually restart is made pending. To actually restart it a "continue" command from the master GDB session (see next) or the Telnet "cont" command is necessary.
- If the CGROUP core mask includes other cores beside the actual one, then all cores in the mask are prepared for restart (if not already done) and finally the whole core group is restarted at the same time.

This supports two different debug scenarios where the first one is actually a special case of the second one:

- Debug only one core via GDB but make sure that always all cores are either halted or running. For this only one CGROUP for the debugged core is necessary. The core mask defines all the cores.
- Debug multiple cores (not necessary all cores) with different GDB sessions. Here one core will be let's say the master core with the attached master GDB session. Always continue all other GDB session (cores) before entering the continue command in the master GDB session. For the master core define the CGROUP mask with all cores. For other cores set only the bit in the core mask that represents the core itself.

```
[INIT]
;
; setup device trigger, debug halt always all cores
WREG  cgcr0     0x0000000f  ;CGCR0: Core Group 0 (0,1,2,3)
WREG  cgcr1     0x0000000f  ;CGCR1: Core Group 1 (0,1,2,3)
WREG  cgcr2     0x0000000f  ;CGCR2: Core Group 2 (0,1,2,3)
WREG  csttacr0  0x00020001  ;CSTTACR0: trigger if a core from group 1 enter debug halt
;
; use device event 4 to halt cores
WREG  cgacrd4   0x00000022  ;CGACRD4: if device event, halt cores in group 2
;

or

; use an EPU event to halt cores
WREG  epsmcr13  0x53000000  ;EPSMCR13[ISEL0] = 83 (RCPM Concerntrator 0 Event)
WREG  epecr13   0x80000000  ;EPECR13[IC0]    = 2  (Input 0 is sufficient)
WREG  cgacre13  0x00000022  ;CGACRE13: if EPU event, halt cores in group 2
```

Writing to the DSCR space via the init list is possible even when all cores are running out of reset. This allows to setup the Cross-Trigger logic also in this case.

4 Specifications

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

Specifications subject to change without notice

5 Environmental notice

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

6 Declaration of Conformity (CE)

CE

DECLARATION OF CONFORMITY

This declaration is valid for following product:

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

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

EMC Directive 89/336/EEC

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

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

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

Manufacturer:

ABATRON AG
Lettenstrasse 9
CH-6343 Rotkreuz

Authority:

Max Vock
Marketing Director

Ruedi Dummermuth
Technical Director

Rotkreuz, 7/18/2007

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

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

Possible reasons

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

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

Possible reasons

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

B Maintenance

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

C Trademarks

All trademarks are property of their respective holders.

