

*bd*iGDB

JTAG debug interface for GNU Debugger

*PPC4xx / APM8xxx*



# User Manual

Manual Version 1.25 for BDI2000

**abatron**

©1997-2014 by Abatron AG

<b>1 Introduction .....</b>	<b>4</b>
1.1 BDI2000.....	4
1.2 BDI Configuration .....	5
<b>2 Installation .....</b>	<b>6</b>
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 Installation of the Configuration Software .....	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
<b>3 Using bdiGDB .....</b>	<b>19</b>
3.1 Principle of operation .....	19
3.2 Configuration File.....	20
3.2.1 Part [INIT].....	21
3.2.2 Part [TARGET] .....	25
3.2.3 Part [HOST].....	31
3.2.4 Part [FLASH] .....	33
3.2.5 Part [REGS] .....	37
3.3 Debugging with GDB .....	40
3.3.1 Target setup .....	40
3.3.2 Connecting to the target.....	40
3.3.3 Breakpoint Handling.....	41
3.3.4 GDB monitor command.....	41
3.3.5 Target serial I/O via BDI.....	42
3.3.6 Embedded Linux MMU Support .....	43
3.4 Telnet Interface.....	45
3.5 Multi-Core Support.....	48
3.6 Low level JTAG mode.....	53
<b>4 Specifications .....</b>	<b>54</b>
<b>5 Environmental notice .....</b>	<b>55</b>
<b>6 Declaration of Conformity (CE) .....</b>	<b>55</b>
<b>7 Warranty and Support Terms .....</b>	<b>56</b>
7.1 Hardware .....	56
7.2 Software .....	56
7.3 Warranty and Disclaimer .....	56
7.4 Limitation of Liability .....	56

## Appendices

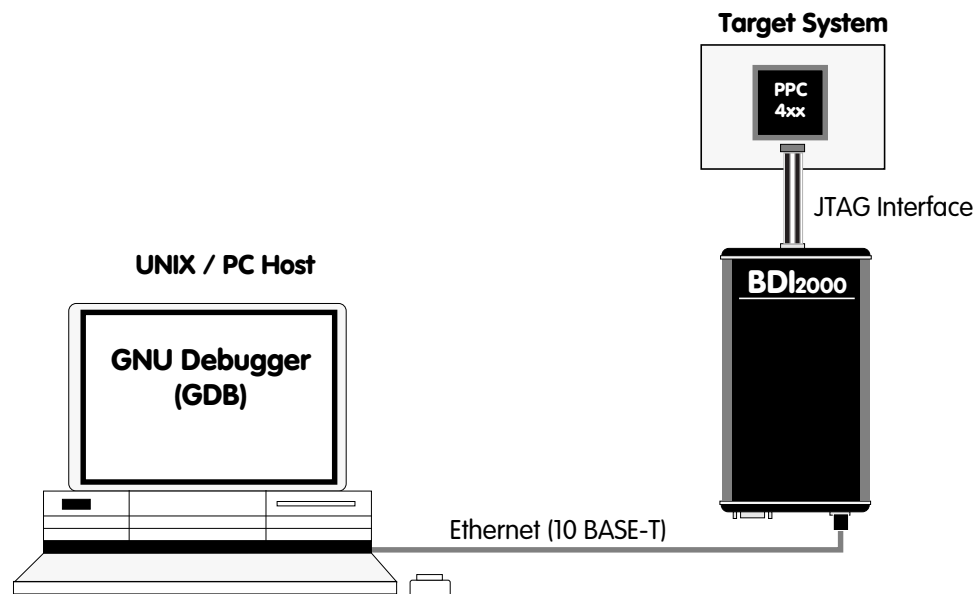
<b>A Troubleshooting .....</b>	<b>57</b>
<b>B Maintenance .....</b>	<b>58</b>
<b>C Trademarks .....</b>	<b>60</b>

## 1 Introduction

bdiGDB enhances the GNU debugger (GDB), with JTAG debugging for PowerPC 4xx 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 IBM 405GP Reference Board
; -----
;
[INIT]
; init core register
WSPR 954 0x00000000;DCWR: Disable data cache write-thru
WSPR 1018 0x00000000;DCCR: Disable data cache
WSPR 1019 0x00000000;ICCR: Disable instruction cache
WSPR 982 0x00000000;EVPR: Exception Vector Table @0x00000000
; Setup Peripheral Bus
WDCR 18 0x00000010;Select PB0AP
WDCR 19 0x9B015480;PB0AP: Flash and SRAM
WDCR 18 0x00000000;Select PB0CR
WDCR 19 0xFFFF18000;PB0CR: 1MB at 0xFFFF00000, r/w, 8bit
; Setup SDRAM Controller
WDCR 16 0x00000080;Select SDTR1
WDCR 17 0x0086400D;SDTR1: SDRAM Timing Register
WDCR 16 0x00000040;Select MB0CF
WDCR 17 0x00046001;MB0CF: 16MB @ 0x00000000
WDCR 16 0x00000048;Select MB2CF
WDCR 17 0x01046001;MB2CF: 16MB @ 0x01000000
WDCR 16 0x00000030;Select RTR
WDCR 17 0x05F00000;RTR: Refresh Timing Register
WDCR 16 0x00000020;Select MCOPT1
WDCR 17 0x80800000;MCOPT1: Enable SDRAM Controller

[TARGET]
JTAGCLOCK 0 ;use 16 MHz JTAG clock
CPUTYPE 405 ;the used target CPU type
BDIMODE AGENT ;the BDI working mode (LOADONLY | AGENT)
BREAKMODE SOFT ;SOFT or HARD, HARD uses PPC hardware breakpoint
VECTOR CATCH ;catch unhandled exceptions

[HOST]
IP 151.120.25.115
FILE E:\cygnus\root\usr\demo\evb405\vxworks
FORMAT ELF
LOAD MANUAL ;load code MANUAL or AUTO after reset
DEBUGPORT 2001

[FLASH]
WORKSPACE 0x00004000 ;workspace in target RAM for fast programming algorithm
CHIPTYPE AM29F ;Flash type (AM29F | AM29BX8 | AM29BX16 | I28BX8 | I28BX16)
CHIPSIZE 0x80000 ;The size of one flash chip in bytes (e.g. AM29F040 = 0x80000)
BUSWIDTH 8 ;The width of the flash memory bus in bits (8 | 16 | 32)
FILE E:\cygnus\root\usr\demo\evb405\evb405gp.hex ;The file to program
ERASE 0xFFFF80000 ;erase sector 0 of flash in U7 (AM29F040)
ERASE 0xFFFF90000 ;erase sector 1 of flash
```

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

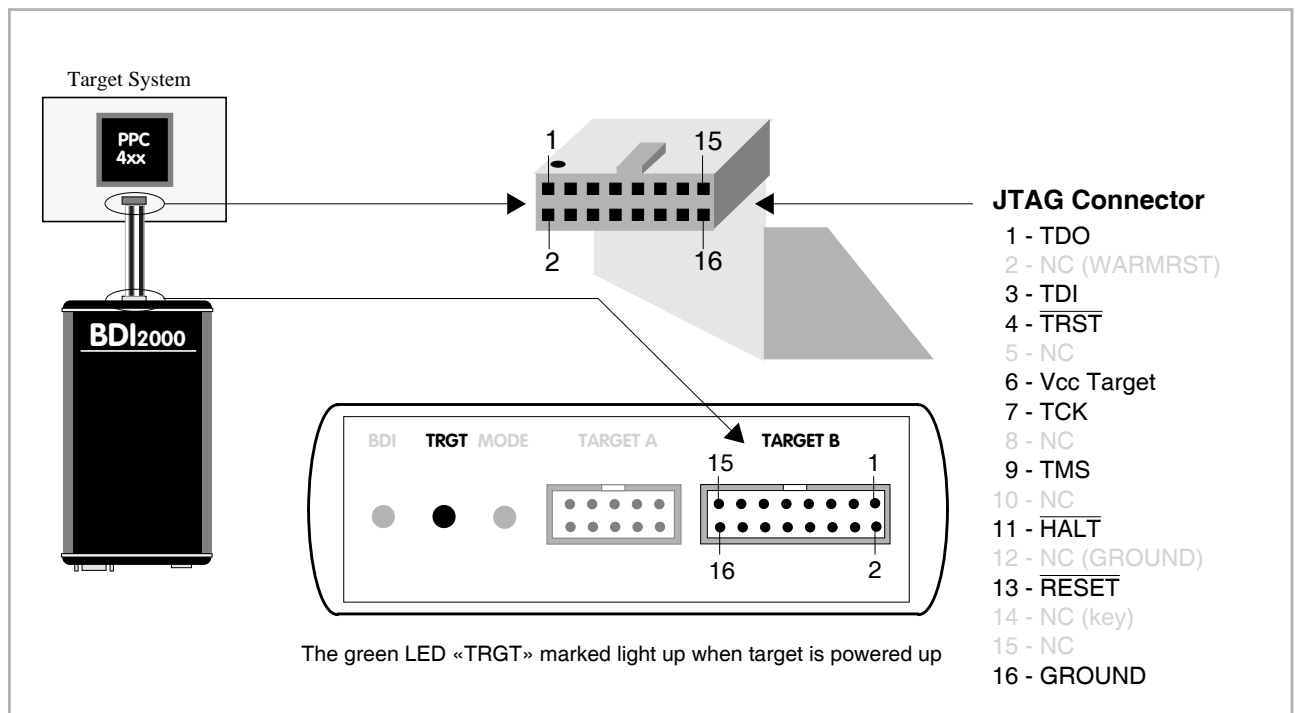
## 2 Installation

### 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 4xx JTAG connector specification.



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



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

## BDI TARGET B Connector Signals:

Pin	Name	Description
1	TDO	<b>JTAG Test Data Out</b> This input to the BDI2000 connects to the target TDO pin.
2	<reserved>	
3	TDI	<b>JTAG Test Data In</b> This output of the BDI2000 connects to the target TDI pin.
4	$\overline{\text{TRST}}$	<b>JTAG Test Reset</b> This output of the BDI2000 resets the JTAG TAP controller on the target.
5	<reserved>	
6	Vcc Target	<b>1.8 – 5.0V:</b> This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vdd I/O on the target board.  <b>3.0 – 5.0V with Rev. B :</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	<b>JTAG Test Clock</b> This output of the BDI2000 connects to the target TCK pin.
8	<reserved>	
9	TMS	<b>JTAG Test Mode Select</b> This output of the BDI2000 connects to the target TMS line.
10	<reserved>	
11	$\overline{\text{HALT}}$	<b>Processor Halt</b> This output of the BDI2000 connects to the target HALT line.
12	GROUND	<b>System Ground</b>
13	$\overline{\text{RESET}}$	<b>System Reset (optional)</b> This open collector output of the BDI2000 is used to hard reset the target system. This is an optional signal and only driven if RESET HARD is selected in the BDI configuration. The standard IBM debug connected specification does not include this signal.
14	<reserved>	
15	<reserved>	
16	GROUND	<b>System Ground</b>

### 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.**

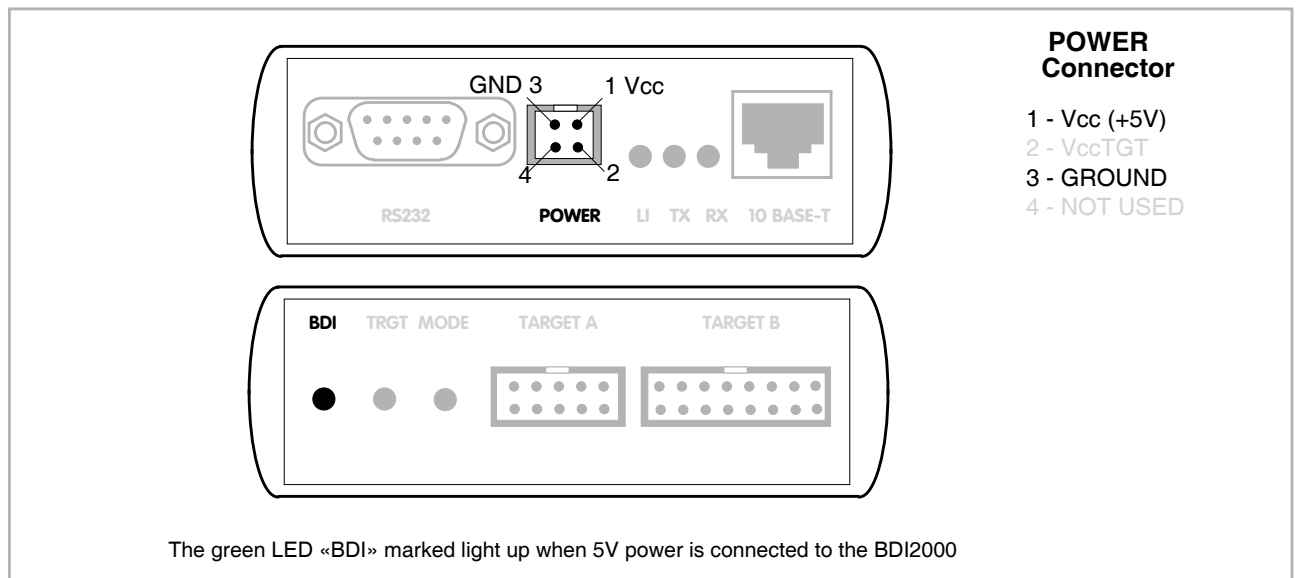


## 2.2 Connecting the BDI2000 to Power Supply

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.**

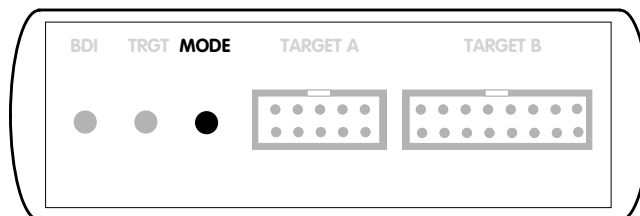


**Please switch on the system in the following sequence:**

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

## 2.3 Status LED «MODE»

The built in LED indicates the following BDI states:



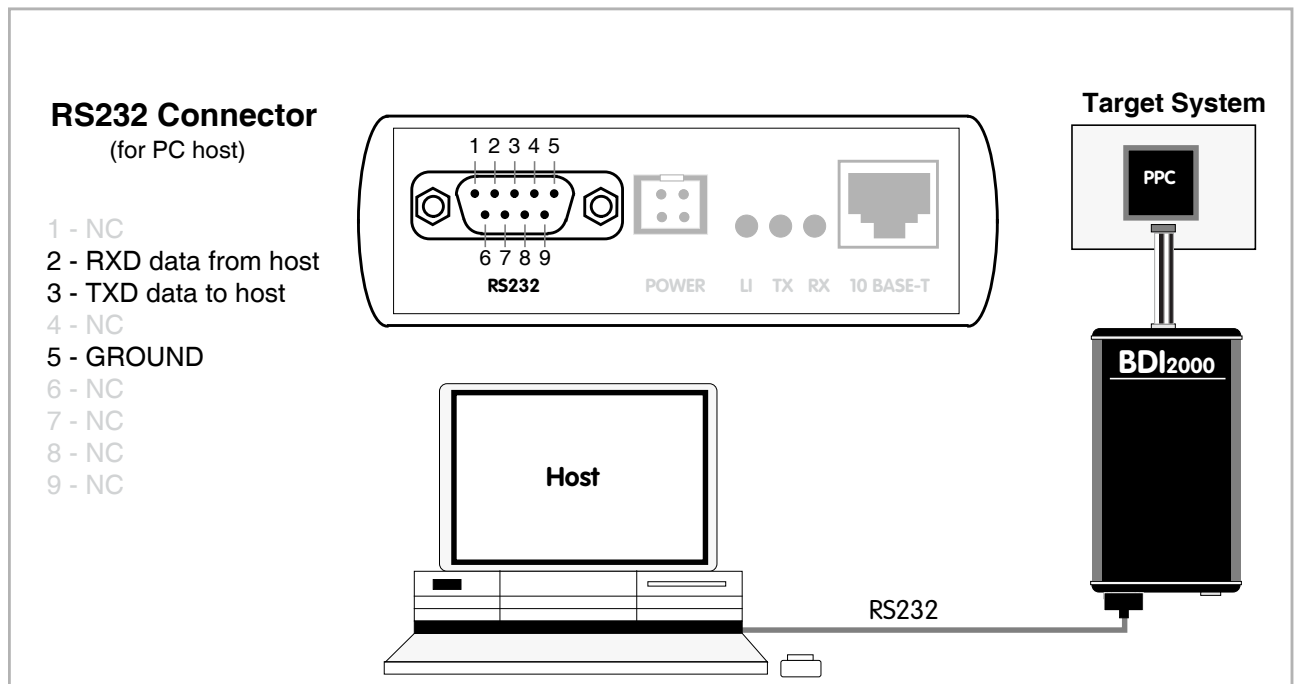
MODE LED	BDI STATES
OFF	The BDI is ready for use, the firmware is already loaded.
ON	The power supply for the BDI2000 is < 4.75VDC.
BLINK	The BDI «loader mode» is active (an invalid firmware is loaded or loading firmware is active).

## 2.4 Connecting the BDI2000 to Host

### 2.4.1 Serial line communication

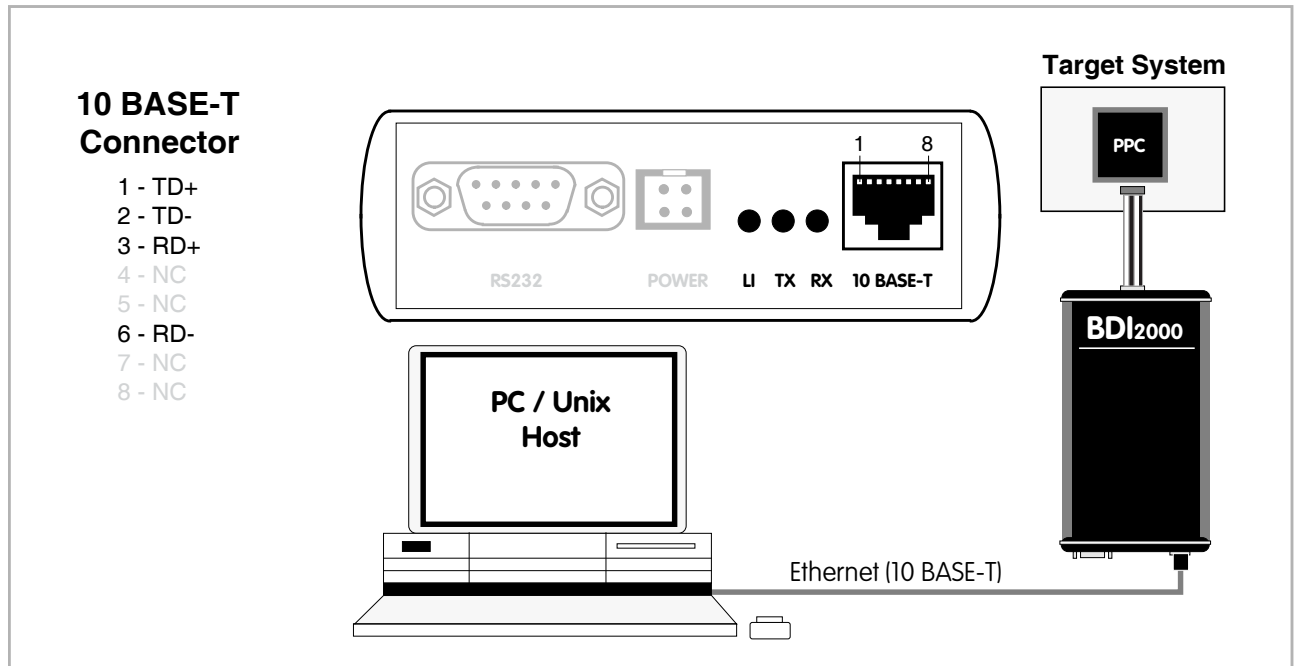
Serial line communication is only used for the initial configuration of the bdiGDB system.

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 (Unshielded 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 Installation of the Configuration Software

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

The following files are on the diskette.

b20pp4gd.exe	Windows Configuration program
b20pp4gd.xxx	Firmware for the BDI2000
pp4jed20.xxx	JEDEC file for the BDI2000 (Rev. B) logic device
pp4jed21.xxx	JEDEC file for the BDI2000 (Rev. C) logic device
ftpsrv.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 to load/update the BDI firmware/logic  
**Note:** A new BDI has no firmware/logic loaded.
- Use the setup tool to transmit the initial configuration parameters
  - IP address of the BDI.
  - IP address of the host with the configuration file.
  - Name of the configuration file. This file is accessed via TFTP.
  - Optional network parameters (subnet mask, default gateway).

### Activating BOOTP:

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

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

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

The MAC address is derived from the serial number as follows:

MAC: 00-0C-01-xx-xx-xx , replace the xx-xx-xx with the 6 left digits of the serial number

Example: SN# 93123457 ==>> 00-0C-01-93-12-34

### 2.5.1 Configuration with a Linux / Unix host

The firmware / logic update and the initial configuration of the BDI2000 is done with a command line utility. In the ZIP Archive bdisetup.zip are all sources to build this utility. More information about this utility can be found at the top in the bdisetup.c source file. There is also a make file included. Starting the tool without any parameter displays information about the syntax and parameters.



**To avoid data line conflicts, the BDI2000 must be disconnected from the target system while programming the logic for an other target CPU (see Chapter 2.1.1).**

Following the steps to bring-up a new BDI2000:

#### 1. Build the setup tool:

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

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

#### 2. Check the serial connection to the BDI:

With "bdisetup -v" you may check the serial connection to the BDI. The BDI will respond with information about the current loaded firmware and network configuration.

**Note:** Login as root, otherwise you probably have no access to the serial port.

```
[root@LINUX_1 bdisetup]# ./bdisetup -v -p/dev/ttyS0 -b57
BDI Type : BDI2000 Rev.C (SN: 92152150)
Loader : V1.05
Firmware : unknown
Logic : unknown
MAC : 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 -tPPC400
Connecting to BDI loader
Erasing CPLD
Programming firmware with ./b20pp4gd.103
Programming CPLD with ./pp4jed21.101
```

#### 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 \
> -fevb405gp.cfg
Connecting to BDI loader
Writing network configuration
Writing init list and mode
Configuration passed
```

#### 5. Check configuration and exit loader mode:

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

```
[root@LINUX_1 bdisetup]# ./bdisetup -v -p/dev/ttyS0 -b57 -s
BDI Type : BDI2000 Rev.C (SN: 92152150)
Loader : V1.05
Firmware : V1.03 bdiGDB for PPC400
Logic : V1.01 PPC400
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 : evb405gp.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).**

BDI2000 Firmware / Logic			
	Current	Newest	
Loader	1.05		Current
Firmware	1.15	1.15	Erase
Logic	1.03	1.03	Update

*dialog box «BDI2000 Update/Setup»*

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

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

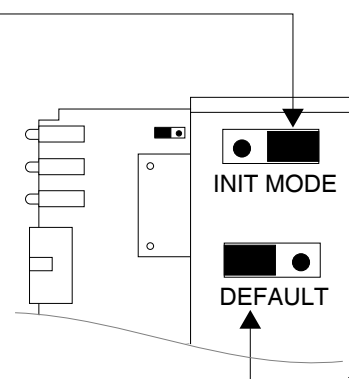


BDI IP Address	Enter the IP address for the BDI2000. Use the following format: xxx.xxx.xxx.xxx e.g.151.120.25.101 Ask your network administrator for assigning an IP address to this BDI2000. Every BDI2000 in your network needs a different IP address.
Subnet Mask	Enter the subnet mask of the network where the BDI is connected to. Use the following format: xxx.xxx.xxx.xx.e.g.255.255.255.0 A subnet mask of 255.255.255.255 disables the gateway feature. Ask your network administrator for the correct subnet mask.
Default Gateway	Enter the IP address of the default gateway. Ask your network administrator for the correct gateway IP address. If the gateway feature is disabled, you may enter 255.255.255.255 or any other value..
Config - Host IP Address	Enter the IP address of the host with the configuration file. The configuration file is automatically read by the BDI after every start-up via TFTP. If the host IP is 255.255.255.255 then the setup tool stores the configuration read from the file into the BDI internal flash memory. In this case no TFTP server is necessary.
Configuration file	Enter the full path and name of the configuration file. This file is read by the setup tool or via TFTP.
Transmit	Click on this button to store the configuration in the BDI2000 flash memory.

### 2.5.3 Recover procedure

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

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



## 2.6 Testing the BDI2000 to host connection

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

- If not already done, connect the 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.

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

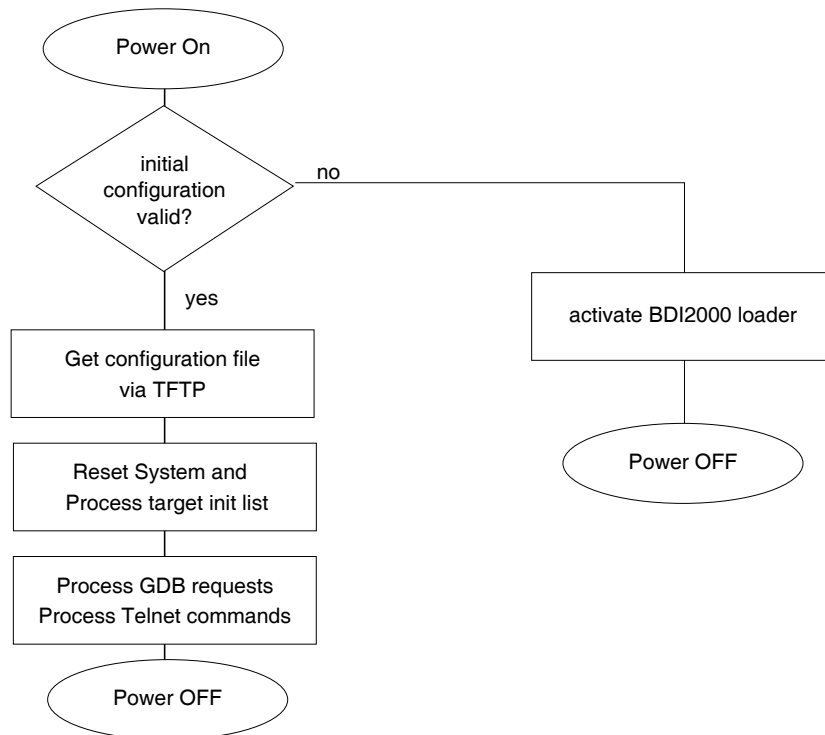
You may enter the TFTP server into the Startup group so the server is started every time you login.

## 3 Using bdiGDB

### 3.1 Principle of operation

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

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



**Breakpoints:**

There are two 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 4 (2 for 401/403) breakpoints can be active at the same time.

The following example selects SOFT as the breakpoint mode:

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

All the time the application is suspended (i.e. caused by a breakpoint) the target processor remains freezed.

### 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).

### 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	<p>Write value to the selected general purpose register.</p> <p>register      the register number 0 .. 31</p> <p>value        the value to write into the register</p> <p>Example: WGPR 0 5</p>
WSPR register value	<p>Write value to the selected special purpose register.</p> <p>register      the register number</p> <p>value        the value to write into the register</p> <p>Example: WSPR 27 0x00001002 ; SRR1 : ME,RI</p>
WDCR register value	<p>Write value to the selected device control register. Some special register numbers are use to access the PPC476 Multi-core debug registers.</p> <p>register      the register number</p> <p>value        the value to write into the register</p> <p>Example: WDCR 0x12 0x00000010 ; Select EBC0_B0AP</p>
WREG name value	<p>Write value to the selected register/memory by name</p> <p>name        the case sensitive register name from the reg def file</p> <p>value        the value to write to the register/memory</p> <p>Example: WREG msr 0x00001002</p>
WM8 address value	<p>Write a byte (8bit) to the selected memory place.</p> <p>address      the memory address</p> <p>value        the value to write to the target memory</p> <p>Example: WM8 0xFFFFFA21 0x04 ; SYPCR: watchdog disable ...</p>
WM16 address value	<p>Write a half word (16bit) to the selected memory place.</p> <p>address      the memory address</p> <p>value        the value to write to the target memory</p> <p>Example: WM16 0x02200200 0x0002 ; TBSCR</p>
WM32 address value	<p>Write a word (32bit) to the selected memory place.</p> <p>address      the memory address</p> <p>value        the value to write to the target memory</p> <p>Example: WM32 0x02200000 0x01632440 ; SIUMCR</p>

RM8 address value	<p>Read a byte (8bit) from the selected memory place.</p> <p>address        the memory address</p> <p>Example: RM8 0x00000000</p>
RM16 address value	<p>Read a half word (16bit) from the selected memory place.</p> <p>address        the memory address</p> <p>Example: RM16 0x00000000</p>
RM32 address value	<p>Read a word (32bit) from the selected memory place.</p> <p>address        the memory address</p> <p>Example: RM32 0x00000000</p>
SIDCR cfgaddr cfgdata	<p>Sets the DCR addresses of the Configuration Address and Data Register used for Indirectly accessed Device Control Registers.</p> <p>cfgaddr        the address of the Configuration Address Register</p> <p>cfgdata        the address of the Configuration Data Register</p> <p>Example:       SIDCR 0x10 0x11 ; set SDRAM configuration</p> <p>                 SIDCR 0x12 0x13 ; set EBC configuration</p>
WIDCR offset data	<p>Write to an Indirectly accessed Device Control Register using the Configuration Address and Data Registers define with the last SIDCR entry.</p> <p>offset        offset of the register, will be written to cfgaddr</p> <p>data        value for the register, will be written to cfgdata</p> <p>Example:       SIDCR 0x10 0x11 ; set SDRAM Config</p> <p>                 WIDCR 0x0040 0x00007201 ; SDRAM_MB0CF</p> <p>                 WIDCR 0x0044 0x08007201 ; SDRAM_MB1CF</p> <p>                 WIDCR 0x0048 0x00000000 ; SDRAM_MB2CF</p>
DELAY value	<p>Delay for the selected time. A delay may be necessary to let the clock PLL lock again after a new clock rate is selected.</p> <p>value        the delay time in milliseconds (1...30000)</p> <p>Example: DELAY 500 ; delay for 0.5 seconds</p>
MMAP start end	<p>Because a memory access to an invalid memory space via JTAG leads 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.</p> <p>start        the start address of a valid memory range</p> <p>end        the end address of this memory range</p> <p>Example: MMAP 0xFFE00000 0xFFFFFFFF ;Boot ROM</p>
MMAP TLB	<p>Only for 440/464/465: If this entry is present, the BDI checks every memory access against the current TLB setting. This avoids illegal memory accesses. Don't mix the two different MMAP entry types.</p>

### Adding entries to the 440/464/465 TLB:

For 440/464/465 cores, it is necessary to setup the TLB before memory can be accessed. This is because the MMU is always enabled. The init list entries STLB/WTLB allows an initial setup of the TLB array. The first WTLB entry also clears the whole TLB array.

WTLB epn rpn

Only for 440/464/465 (for 476 see next page): Adds an entry to the TLB array. For parameter description see below. A TLB entry can also be added via a Telnet command (enter WTLB at the telnet for a description).

epn the effective page number, size and WIMG flags

rpn the real page number and access rights

Example: WTLB 0xF0000095 0x1F00003F ;Boot Space 256MB

STLB index attrib

Only for 440/464/465 (for 476 see next page): Sets a new start index, the TID and some TLB attributes for the following TLB writes (WTLB).

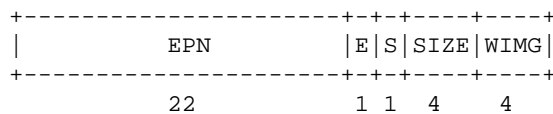
index the start index of the following TLB writes.

attrib defines the TID and some TLB attributes

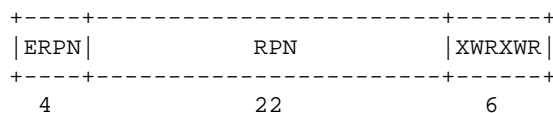
Example: STLB 3 0x00000005 ; Index=3, TID=5

STLB 7 0x00007000 ; Index=7, U1,U2,U3, TID=0

The epn parameter defines the effective page number, endian, space, size and WIMG flags:

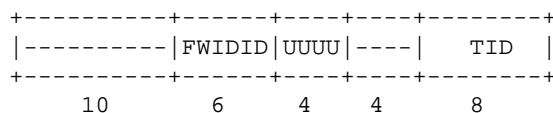


The rpn parameter defines the real page number and access rights:



Not all fields of a TLB entry are defined with the above values. The other values except the valid bit are set to zero unless defined with the optional STLB init list entry. The XWRXWR field starts with the user access rights. See also 440/464/465 user's manual part "Memory Management".

The attrib parameter of the STLB entry has the following bit definitions:



The following example clears the TLB and adds two entries to access ROM and SDRAM:

```
[INIT]
; Setup TLB
WTLB 0xF0000095 0x1F00003F ;Boot Space 256MB, cache inhibited, guarded
WTLB 0x00000098 0x0000003F ;SDRAM 256MB @0x00000000, write-through
```

### Adding entries to the 476 TLB:

For 476 cores, it is necessary to setup the TLB before memory can be accessed. This is because the MMU is always enabled. The init list entries STLW/WTLB allows an initial setup of the TLB array.

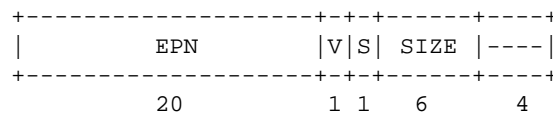
**WTLB epn rpn** Only for 476: Adds an entry to the TLB array. For parameter description see below. A TLB entry can also be added via a Telnet command (enter WTLB at the telnet for a description).

epn defines TLB Word 0 [0:27]  
rpn defines RPN and TLB Word 2 [20:31]

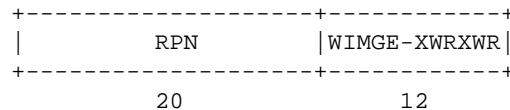
**STLB way erpn** Only for 476: Defines the way, the TID, ERPN and some TLB attributes for the following TLB write (WTLB).

way defines way, bolted and TID  
erpn defines ERPN and TLB Word 2 [0:19]

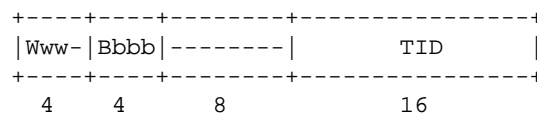
The epn parameter defines the effective page number, space and size:



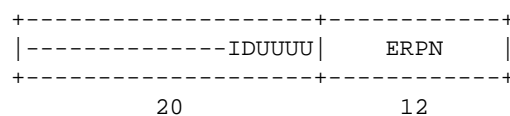
The rpn parameter defines the real page number, WIMG flags, endian and access rights:



The way parameter selects the way, bolted and TID (see also 476 tlbwe instruction):



The erpn parameter defines cache flags, user bits and the extended real page number:





### 3.2.2 Part [TARGET]

The part [TARGET] defines some target specific values.

CPUTYPE type [core [soc]] [FPU]

This value gives the BDI information about the connected CPU/core. Add FPU for chips with integrated floating point unit. Accessing FPU registers needs a workspace in target RAM. See WORKSPACE parameter.

type           The CPU type from the following list:  
                   401, 403, 405, 440, 464, 465, 476, AXX3500  
                   APM86xxx, APM86290, APM86190,  
                   APM86391, APM86392, APM86491,  
                   APM86691, APM86692, APM86771, APM86791

core           the core number within the SOC (0...7)

soc            the SOC number (0...3)

Example:       CPUTYPE 440 FPU  
                   #0 CPUTYPE APM86290 0 0  
                   #1 CPUTYPE APM86290 1 0

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	5 = 200 kHz	10 = 5 kHz
1 = 8.3 MHz	6 = 100 kHz	11 = 2 kHz
2 = 4.1 MHz	7 = 50 kHz	12 = 1 kHz
3 = 1.0 MHz	8 = 20 kHz	
4 = 500 kHz	9 = 10 kHz	

Example:       JTAGCLOCK 1 ; JTAG clock is 8.3 MHz

RESET type [time]

Defines the reset type the BDI uses when resetting the target via the JTAG debug port or via debug connector pin 13.

type           NONE, CORE, CHIP, SYSTEM (default)  
                   HARD (via debug connector pin 13)

time           The time in milliseconds the BDI assert the reset signal.

Example:       RESET CHIP ; IOP480 does not support system reset

WAKEUP time

This entry in the init list allows to define a delay time (in ms) the BDI inserts between forcing a target reset and starting communicating with the target.

time           the delay time in milliseconds

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

MEMDELAY time

This entry in the init list allows to define a delay time (in TCK's) the BDI inserts for memory block reads between stuffing the "lwzu" instruction and reading the loaded GPR. Maybe used when dumping slow memory.

time           the delay time in multiple of 8 TCK's (default 10 x 8)

Example:       MEMDELAY 2 ;16 TCK's memory read access delay

## STARTUP mode [runTime] [RUN|HALT]

This parameter selects the core startup mode and for APM86xxx if the core should be halted after sleep/powerdown.

The following startup modes are supported:

- HALT** This default mode forces the core to debug mode immediately out of reset. No code is executed after reset.
- STOP** In this mode, the BDI lets the core execute code for "run-time" milliseconds after reset. This mode is useful when monitor code should initialize the target system.
- RUN** After reset, the core executes code until stopped by the Telnet "halt" command.
- WAIT** Same as RUN but don't poll core status until selected via Telnet "select" command.

[RUN|HALT] APM86xxx: Defines if a core should be halted after sleep/powerdown. The default is RUN.

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 TRAP instruction.
- HARD** In this mode, the PPC breakpoint hardware is used. Only 2 / 4 breakpoints at a time are supported.

Example: BREAKMODE HARD

## STEPMODE mode

This parameter defines how single step (instruction step) is implemented. The alternate step mode (HWBP) is useful when stepping instructions that causes a TLB miss exception.

- JTAG** This is the default mode. The step feature of the JTAG debug interface is used for single stepping.
- HWBP** In this mode, one or two hardware breakpoints are used to implement single stepping. Use this mode when debugging a Linux kernel.

Example: STEPMODE HWBP

## REGLIST list

In order to optimize the time spent to read registers, this parameter can be used. You can define which register group is really read from the target. By default STD and FPR are read and transferred.

The following names are use to select a register group:

- STD** The standard 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.
- SPR** Some additional special purpose register.
- ALL** Include all register groups

Example: REGLIST STD ; only standard registers

VECTOR CATCH	<p>When this line is present, the BDI catches all unhandled exceptions. Catching exceptions is only possible if the vector table is writable.</p> <p>Example: VECTOR CATCH ; catch unhandled exception</p>
MMU XLAT [kb]	<p>The BDI supports Linux kernel debugging when MMU is on. If this line is present, the BDI assumes that all addresses received from GDB and Telnet are virtual addresses. The optional parameter defines the kernel virtual base address (default is 0xC0000000) and is used for default address translation. If necessary the BDI creates appropriate TLB entries before accessing memory based on information found in the kernel page table. For more information see also chapter "Embedded Linux MMU Support". If not zero, the 12 lower bits of "kb" defines the position of the page present bit in a page table entry. By default 0x002 (440: 0x001) is assumed for the page present bit. The position depends on the Linux kernel version.</p> <p>kb The kernel virtual base address (KERNELBASE)</p> <p>Example: MMU XLAT ;enable support for virtual addresses MMU XLAT 0xC0000020 ; page present bit is 0x020</p>
PTBASE addr	<p>This parameter defines the physical memory address where the BDI looks for the virtual address of the array with the two page table pointers. For more information see also chapter "Embedded Linux MMU Support".</p> <p>addr Physical address of the memory used to store the virtual address of the array with the two page table pointers.</p> <p>Example: PTBASE 0xf0</p>
SIO port [baudrate]	<p>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.</p> <p>port The TCP/IP port used for the host communication.</p> <p>baudrate The BDI supports 2400 ... 115200 baud</p> <p>Example: SIO 7 9600 ;TCP port for virtual IO</p>
WORKSPACE address	<p>In order to access the floating-point registers, the BDI needs a workspace of 8 bytes in target RAM. Enter the base address of this RAM area.</p> <p>address the address of the RAM area</p> <p>Example: WORKSPACE 0x00000000</p>
HALT [HIGH   LOW]	<p>With this parameter it is possible to define if the HALT signal is active low (default) or active high.</p> <p>Example: HALT HIGH ;HALT signal is active high</p>
CGROUP cores	<p>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.</p> <p>cores The selected cores as bitmap.</p> <p>Example: #0 CGROUP 0x0f ;GDB continue core group (restart) #1 CGROUP 0x02 ;GDB continue core group (prepare)</p>

### Daisy chained JTAG devices:

For PPC4xx targets, the BDI can also handle systems with multiple devices connected to the JTAG scan chain. In order to put the other devices into BYPASS mode and to count for the additional bypass registers, the BDI needs some information about the scan chain layout. Enter the number (count) and total instruction register (irlen) length of the devices present before the PPC4xx chip (Predecessor). Enter the appropriate information also for the devices following the PPC4xx chip (Successor):

SCANPRED count irlen	This value gives the BDI information about JTAG devices present before the PPC4xx chip in the JTAG scan chain.	
count	The number of preceding devices (0 ... 31)	
irlen	The sum of the length of all preceding instruction registers (IR) (0 ... 1024)	
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 PPC4xx chip in the JTAG scan chain.	
count	The number of succeeding devices (0 ... 31)	
irlen	The sum of the length of all succeeding instruction registers (IR) (0 ... 1024)	
Example:	SCANSUCC 2 12 ; two device with an IR length of 8+4	
SCANMISC len val pos	This option has been added to support Xilinx Virtex-II Pro 405 cores. The IR length of a 405 core is 4 (instead of 7) and if the FPGA JTAG is daisy chained ,it needs a special IR value (not bypass). Also the FPGA has actually no bypass register if IR is loaded with 100000 .	
len	The length of the 405 IR register (default is 7)	
val	The IR value for the device(s) connected after the device under test. Only 8 bits can be defined. Default is 0xFF (bypass).	
pos	The position of the LSB of special IR value. The number of bits in the scan chain after the LSB (default is 0).	
Example:	SCANMISC 4 0xE0 ; IR len = 4, IR lsb = 11100000 SCANMISC 8 ; 440GX has 8 bit IR length	

The following example shows a configuration for the a Xilinx Virtex-II Pro with one 405 daisy chained with the FPGA JTAG (405-FPGA):

```
SCANPRED    0    0
SCANSUCC    0    6    ;6 (FPGA)
SCANMISC    4    0xE0    ;IR length = 4, IR LSB = ..100000
```

The following example shows a configuration for the a Xilinx Virtex-II Pro with four 405 daisy chained with the FPGA JTAG (405-**405**-405-FPGA). The second 405 is selected for debugging :

```
SCANPRED    1    4    ;4 (405)
SCANSUCC    2    14    ;8 (2*405) + 6 (FPGA)
SCANMISC    4    0xE0    ;IR length = 4, IR LSB = ..100000
```

**Xilinx Virtex-II Pro JTAG configurations with FPGA in scan chain :****405-FPGA:**

```
SCANPRED 0 0
SCANSUCC 0 6 ;6 (FPGA)
SCANMISC 4 0xE0 ;IR length = 4, IR LSB = ..100000
```

**405-405-FPGA:**

```
SCANPRED 0 0
SCANSUCC 1 10 ;4 (405) + 6 (FPGA)
SCANMISC 4 0xE0 ;IR length = 4, IR LSB = ..100000
```

**405-405-FPGA:**

```
SCANPRED 1 4 ;4 (405)
SCANSUCC 0 6 ;6 (FPGA)
SCANMISC 4 0xE0 ;IR length = 4, IR LSB = ..100000
```

**405-405-405-405-FPGA:**

```
SCANPRED 0 0
SCANSUCC 3 18 ;12 (3*405) + 6 (FPGA)
SCANMISC 4 0xE0 ;IR length = 4, IR LSB = ..100000
```

**405-405-405-405-FPGA:**

```
SCANPRED 1 4 ;4 (405)
SCANSUCC 2 14 ;8 (2*405) + 6 (FPGA)
SCANMISC 4 0xE0 ;IR length = 4, IR LSB = ..100000
```

**405-405-405-405-FPGA:**

```
SCANPRED 2 8 ;8 (2*405)
SCANSUCC 1 10 ;4 (405) + 6 (FPGA)
SCANMISC 4 0xE0 ;IR length = 4, IR LSB = ..100000
```

**405-405-405-405-FPGA:**

```
SCANPRED 3 12 ;12 (3*405)
SCANSUCC 0 6 ;6 (FPGA)
SCANMISC 4 0xE0 ;IR length = 4, IR LSB = ..100000
```

## Xilinx Virtex-II Pro JTAG configurations without FPGA in scan chain :

### 405:

```
SCANPRED 0 0
SCANSUCC 0 0
SCANMISC 4 ;IR length = 4
```

### 405-405:

```
SCANPRED 0 0
SCANSUCC 1 4 ;4 (1*405)
SCANMISC 4 ;IR length = 4
```

### 405-405:

```
SCANPRED 1 4 ;4 (1*405)
SCANSUCC 0 0
SCANMISC 4 ;IR length = 4
```

### 405-405-405-405:

```
SCANPRED 0 0
SCANSUCC 3 12 ;12 (3*405)
SCANMISC 4 ;IR length = 4
```

### 405-405-405-405:

```
SCANPRED 1 4 ;4 (1*405)
SCANSUCC 2 8 ;8 (2*405)
SCANMISC 4 ;IR length = 4
```

### 405-405-405-405:

```
SCANPRED 2 8 ;8 (2*405)
SCANSUCC 1 4 ;4 (1*405)
SCANMISC 4 ;IR length = 4
```

### 405-405-405-405:

```
SCANPRED 3 12 ;12 (3*405)
SCANSUCC 0 0
SCANMISC 4 ;IR length = 4
```

### 3.2.3 Part [HOST]

The part [HOST] defines some host specific values.

IP ipaddress	<p>The IP address of the host.</p> <p>ipaddress     the IP address in the form xxx.xxx.xxx.xxx</p> <p>Example:     IP 151.120.25.100</p>
FILE filename	<p>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.</p> <p>filename     the filename including the full path or \$ for relative path.</p> <p>Example:     FILE F:\gnu\demo\ppc\test.elf</p> <p>              FILE \$test.elf</p>
FORMAT format [offset]	<p>The format of the image file and an optional load address offset. Currently S-record, a.out and ELF formats are supported. 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.</p> <p>format        SREC, AOUT, ELF, IMAGE* or ROM</p> <p>Example:     FORMAT ELF</p> <p>              FORMAT ELF 0x10000</p>
LOAD mode	<p>In Agent mode, this parameters defines if the code is loaded automatically after every reset.</p> <p>mode          AUTO, MANUAL</p> <p>Example:     LOAD MANUAL</p>
START address	<p>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 (0xFFFFFFFFC).</p> <p>address       the address where to start the program file</p> <p>Example:     START 0x1000</p>

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 440GX>

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	<p>This parameter defines the type of flash used. It is used to select the correct programming algorithm.</p> <p>format            AM29F, AM29BX8, AM29BX16, I28BX8, I28BX16, AT49, AT49X8, AT49X16, STRATAX8, STRATAX16, MIRROR, MIRRORX8, MIRRORX16, S29M64X8, S29M32X16, S29GLSX16, S29VSRX16, M58X32, AM29DX16, AM29DX32</p> <p>Example:        CHIPTYPE   AM29F</p>
CHIPSIZE size	<p>The size of <b>one</b> flash chip in bytes (e.g. AM29F010 = 0x20000). This value is used to calculate the starting address of the current flash memory bank.</p> <p>size             the size of one flash chip in bytes</p> <p>Example:        CHIPSIZE   0x80000</p>
BUSWIDTH width [PLXFIX]	<p>Enter the width of the memory bus that leads to the flash chips. Do not enter the width of the flash chip itself. The parameter CHIPTYPE carries the information about the number of data lines connected to one flash chip. For example, enter 16 if you are using two AM29F010 to build a 16bit flash memory bank. The additional parameter PLXFIX is necessary if you program AMD/Atmel flashes with a PLX IOP480 target system.</p> <p>with             the width of the flash memory bus in bits (8   16   32)</p> <p>Example:        BUSWIDTH   16</p>
FILE filename	<p>The default name of the file that is programmed into flash using the Telnet 'prog' command. This name is used to access the file via TFTP. If the filename starts with a \$, this \$ is replace with the path of the configuration file name. This name may be overridden interactively at the Telnet interface.</p> <p>filename        the filename including the full path or \$ for relative path.</p> <p>Example:        FILE   F:\gnu\ppc\bootrom.hex                   FILE   \$bootrom.hex</p>
FORMAT format [offset]	<p>The format of the file and an optional address offset. The optional parameter "offset" is added to any load address read from the program file.</p> <p>format           SREC, BIN, AOUT, ELF or IMAGE</p> <p>Example:        FORMAT SREC                   FORMAT ELF 0x10000</p>
WORKSPACE address	<p>If a workspace is defined, the BDI uses a faster programming algorithm that runs out of RAM on the target system. Otherwise, the algorithm is processed within the BDI. The workspace is used for a 1kByte data buffer and to store the algorithm code. There must be at least 2kBytes of RAM available for this purpose.</p> <p>address         the address of the RAM area</p> <p>Example:        WORKSPACE 0x00000000</p>

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

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

address	Address of the flash sector, block or chip to erase
increment	If present, the address offset to the next flash sector
count	If present, the number of equal sized sectors to erase
mode	BLOCK, CHIP, UNLOCK

Without this optional parameter, the BDI executes a sector erase. If supported by the chip, you can also specify a block or chip erase. If UNLOCK is defined, this entry is also part of the unlock list. This unlock list is processed if the Telnet UNLOCK command is entered without any parameters.

**Note:** Chip erase does not work for large chips because the BDI time-outs after 3 minutes. Use block erase.

wait	The wait time in ms is only used for the unlock mode. After starting the flash unlock, the BDI waits until it processes the next entry.
------	---

Example:

```
ERASE 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 PPC405 evaluation board flash memory:

```
[FLASH]
WORKSPACE 0x00004000 ;workspace in target RAM for fast programming algorithm
CHIPTYPE AM29F ;Flash type
CHIPSIZE 0x80000 ;The size of one flash chip in bytes (AM29F040 = 0x80000)
BUSWIDTH 8 ;The width of the flash memory bus in bits (8 | 16 | 32)
FILE E:\cygnus\root\usr\demo\evb405\evb405gp.sss ;The file to program
ERASE 0xFFFF80000 ;erase sector 0 of flash in U7 (AM29F040)
ERASE 0xFFFF90000 ;erase sector 1 of flash
ERASE 0xFFFFA0000 ;erase sector 2 of flash
```

the above erase list maybe replaces with:

```
ERASE 0xFFFF80000 0x10000 3 ;erase 3 sectors
```

**Supported standard parallel NOR Flash Memories:**

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

On our web site ([www.abatron.ch](http://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  0xFFFF0000  0x0060      unlock block 0
WM16  0xFFFF0000  0x00D0
WM16  0xFFFF1000  0x0060      unlock block 1
WM16  0xFFFF1000  0x00D0
      ....
WM16  0xFFFF0000  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
	PMR	Performance monitor register
	DCR	Device control register
	MM	Absolute direct memory mapped register
	DMM1...DMM4	Relative direct memory mapped register
	PMM1...PMM4	Physical relative direct memory mapped register
	IMM1...IMM4	Indirect memory mapped register
	IDCR1...IDCR8	Indirect accessed device control 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 configuration registers).	

The PMMn register type allows to access 440/464/465 registers that are located above the 4 GB effective address range. The BDI first checks if there is already a valid TLB entry present to access this physical address. If no TLB entry allows to access this address, the BDI creates a temporary TLB entry.

```
[REGS]
PMM1    0x20000          ;PCI          (base addr 2_0000_0000)
PMM2    0x14000          ;Peripheral (base addr 1_4000_0000)
FILE    $reg440gx.def
```

```
pcix0_vendid    PMM10x0EC80000    16 SWAP
pcix0_devid     PMM10x0EC80002    16 SWAP
...

emac0_mr0      PMM20x00000800    32
emac0_mr1      PMM20x00000804    32
...
```

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

FILE filename	<p>The name of the register definition file. 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. The file is loaded once during BDI startup.</p> <p>filename      the filename including the full path</p> <p>Example:      FILE C:\bdi\regs\ppc405gp.def</p>
DMMn base	<p>This defines the base address of direct memory mapped registers. This base address is added to the individual offset of the register.</p> <p>base            the base address</p> <p>Example:      DMM1 0x01000</p>
PMMn base	<p>This defines the upper 20 bits of the 36-bit physical base address of physically direct memory mapped registers. This base address is added to the individual offset of the register.</p> <p>base            the upper 20 bits of the 36-bit physical base address</p> <p>Example:      PMM1 0x14000 ;Peripheral (base addr 1_4000_0000) PMM2 0x20000 ;PCI (base addr 2_0000_0000)</p>
IMMn addr data	<p>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.</p> <p>addr            the address of the Address register</p> <p>data            the address of the Data register</p> <p>Example:      IMM1 0x04700000 0x04700004</p>
IDCRn addr data	<p>This defines the numbers of the address and data DCR of indirectly accessed DCR's. The address of an IDCRn register is first written to "Addr-DCR" and then the register value is access using the "Data-DCR".</p> <p>addr            the number of the Address DCR</p> <p>data            the number of the Data DCR</p> <p>Example:      IDCR1 16 17 ;MEMCFGADR and MEMCFGDATA</p>

## Example for a register definition (PPC405GP):

Entry in the configuration file:

```
[REGS]
IDCR1  0x010  0x011  ;MEMCFGADR and MEMCFGDATA
IDCR2  0x012  0x013  ;EBCCFGADR and EBCCFGDATA
IDCR3  0x014  0x015  ;KIAR and KIDR
FILE   E:\cygnus\root\usr\demo\evb405\reg405gp.def
```

The register definition file:

```
;name          type   addr          size
;-----
;
sp              GPR     1
;
;   Special Purpose Registers
;
ccr0            SPR     947
ctr             SPR     9
dac1            SPR     1014
dac2            SPR     1015
dbcr0           SPR     1010
dbcr1           SPR     957
dccr            SPR     1018
....
;
;   Directly Accessed DCR's
;
pesr            DCR     0x084
pear            DCR     0x086
pacr            DCR     0x087
gesr0           DCR     0x0A0
....
;
;   Indirectly Accessed DCR's
;
;   IDCR1 must be set to MEMCFGADR and MEMCFGDATA
;   IDCR2 must be set to EBCCFGADR and EBCCFGDATA
;   IDCR3 must be set to KIAR      and KIDR
;
besra           IDCR1    0x000
besrb           IDCR1    0x008
bear            IDCR1    0x010
mcopt1          IDCR1    0x020
rtr             IDCR1    0x030
....
;
;   Memory-Mapped Registers
;
pmm0la          MM       0xEF400000    32
pmm0ma          MM       0xEF400004    32
pmm0pcila       MM       0xEF400008    32
....
```

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

```
BDI> rd mcopt1
BDI> rm rtr 0x05f00000
```

### 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 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 resetet the target and reloaded the image
```

```
(gdb)target remote bdi2000:2001
```

**Note:**

After loading a program to the target you cannot use the GDB "*run*" command to start execution. You have to use the GDB "*continue*" command.



### 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.

#### User controlled hardware breakpoints:

The PPC4xx has a special watchpoint / breakpoint hardware integrated. Normally the BDI controls this hardware in response to Telnet commands (BI, BDx) or when breakpoint mode HARD is selected. Via the Telnet commands BI and BDx, you cannot access all the features of the breakpoint hardware. Therefore the BDI assumes that the user will control / setup this breakpoint hardware as soon as DBCR (DBCR0 for 405/440/464/465) is written to. This way the debugger or the user via Telnet has full access to all features of this watchpoint / breakpoint hardware. A hardware breakpoint set via BI or BDx gives control back to the BDI.

### 3.3.4 GDB monitor command

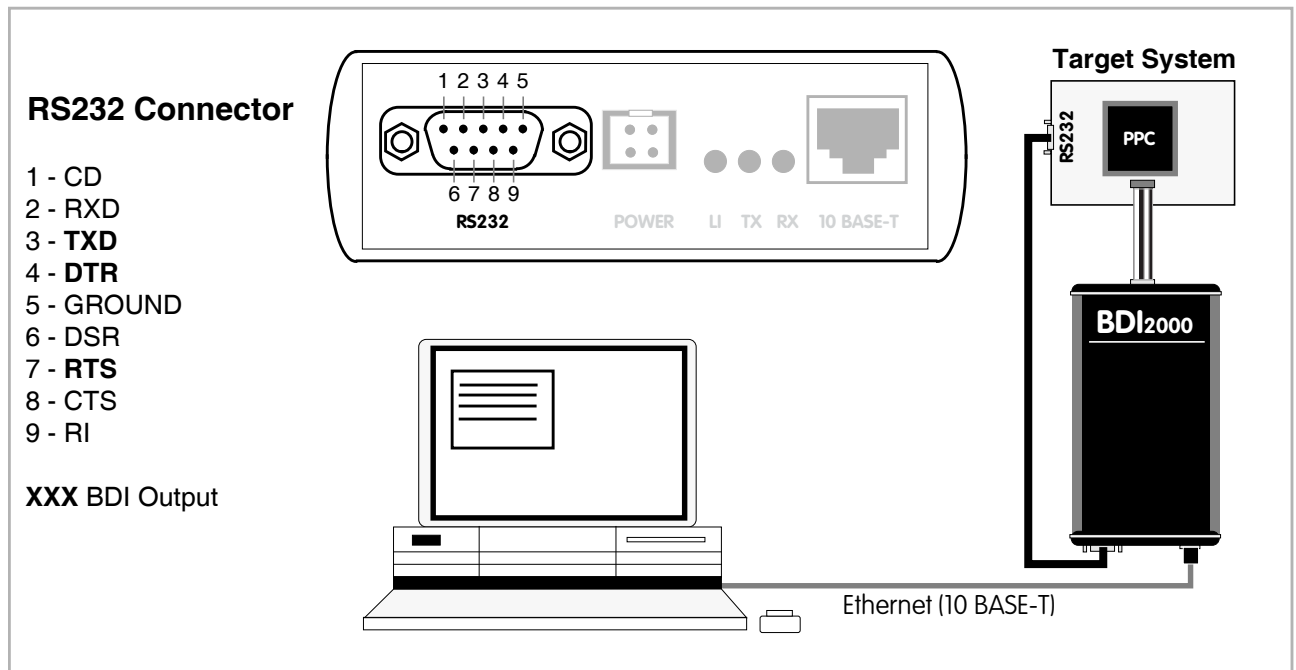
The BDI supports the GDB V5.x "monitor" command. Telnet commands are executed and the Telnet output is returned to GDB. This way you can for example switch the BDI breakpoint mode from within your GDB session.

```
(gdb) target remote bdi2000:2001
Remote debugging using bdi2000:2001
0x10b2 in start ()
(gdb) 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 or Telnet are assumed to be virtual. Before the BDI accesses memory, it either translates this address into a physical one or creates an appropriate TLB entry based on information found in the kernel page tables. A new TLB entry is only added if there is not already a matching one present.

In order to search the page tables, the BDI needs to know the start addresses of the first level page tables. The configuration parameter PTBASE defines the physical address where the BDI looks for the address of an array with two 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 default translation.

Default translation maps addresses in the range KERNELBASE...(KERNELBASE + 0x0FFFFFFF) to 0x00000000...0x0FFFFFFF. 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)
```

Newer versions of "arch/ppc/kernel/head\_4xx.S" support the automatic update of the BDI page table information structure. Search "head\_4xx.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]
....
STEPMODE   HWBP           ;JTAG or HWBP, HWPB uses one or two hardware breakpoints
MMU        XLAT           ;MMU support enabled
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 */
```

#### **Note:**

When searching the page table, the BDI needs to check the page present bit in a page table entry. For PPC4xx targets, the position of this bit has moved around in the past. By default the BDI assumes the following definition for the page present bit (see pgtable.h in your kernel sources):

```
405: #define _PAGE_PRESENT 0x002 /* software: PTE contains a translation */
440: #define _PAGE_PRESENT 0x001 /* software: PTE contains a translation */
```

If this does not match your version of "pgtable.h", use the 12 lower bits of the MMU XLAT parameter to define the correct bit position.

```
MMU XLAT 0xC0000040 ; page present bit is 0x040
```

### 3.4 Telnet Interface

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

Telnet Debug features:

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

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

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

Example of a Telnet session:

```
BDI>res
- TARGET: processing user reset request
- TARGET: resetting target passed
- TARGET: processing target init list ....
- TARGET: processing target init list passed
BDI>info
  Target state      : debug mode
  Debug entry cause : trap instruction
  Current PC       : 0xffffffffc
  Current CR       : 0x00000000
  Current MSR      : 0x00000000
  Current LR       : 0x0001ba70
BDI>md 0
00000000 : 00000000 00000004 00000008 0000000c .....
00000010 : 00000010 00000014 00000018 0000001c .....
00000020 : 00000020 00000024 00000028 0000002c ... ..$....(.,
00000030 : 00000030 00000034 00000038 0000003c ...0...4...8...<
00000040 : 00000040 00000044 00000048 0000004c ...@...D...H...L
.....
```

#### 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.

A PPC4xx target can be forced to debug mode in two different ways. HALT at the Telnet asserts the HALT pin to stop the processor. STOP at the Telnet uses the JTAG stop command. The HALT pin is deasserted with the next RESET or RUN. If a JTAG reset does not completely reset a target system (e.g. IOP480), the sequence Telnet HALT, press reset button, Telnet RESET can be used to force the target to debug mode immediately out of reset.

## The Telnet commands:

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

"RD      [<name>]                  display general purpose or user defined register",
"RDUMP   [<file>]                  dump all user defined register to a file",
"RDFPR                                     display floating point registers",
"RDSPR   <number>                  display special purpose register",
"RDDCR   <number>                  display device control register",
"RDDCRX  <number>                  display device control register (only 46x)",
"RM {<nbr> | <name>} <value>      modify general purpose or user defined register",
"RMSPR   <number> <value>         modify special purpose register",
"RMDCR   <number> <value>         modify device control register",
"RMDCRX  <number> <value>         modify device control register (only 46x)",

"TLB     <from> [<to>]              display TLB entry",
"WTLB <tid:idx> <ws0> <ws1> <ws2> write TLB entry (440/46x)",
"WTLB <way:tid> <ws0> <ws1> <ws2> write TLB entry (47x)",
"DFLUSH  [<addr>]                  flush data cache (addr = cached memory address)",
"IFLUSH                                     invalidate instruction cache",
"DCACHE  <from> [<to>]              display L1 data cache (440/46x: lines, 405: sets)",
"ICACHE  <from> [<to>]              display L1 inst cache (440/46x: lines, 405: sets)",

"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 instuction (single step)",
"TC      [<pc>]                    trace on change of flow",
"HALT                                     stop all cores via HALT pin",
"STOP    [<cores>]                stop core(s) via JTAG port (<cores> = core bit map)",
"SYNC                                     synchronize the BDI with the core(s)",

"JMCDCR <value>                  APM86xxx: set Multi-Core Debug Control Register",

"BI <addr>                        set instruction breakpoint",
"CI [<id>]                        clear instruction breakpoint(s)",
"BD [R|W] <addr>                  set data breakpoint (32bit access)",
"BDH [R|W] <addr>                  set data breakpoint (16bit access)",
"BDB [R|W] <addr>                  set data breakpoint ( 8bit access)",
"CD [<id>]                        clear data breakpoint(s)",

"INFO                                     display information about the current core",
"STATE                                     display information about all cores",
```

## The Telnet commands (cont.):

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

"DELAY <ms>                  delay for a number of milliseconds",
"SELECT <core>                change the current core",
"HOST <ip>                    change IP address of program file host",
"PROMPT <string>              defines a new prompt string",
"CONFIG                       display or update BDI configuration",
"CONFIG <file> [<hostIP> [<bdiIP> [<gateway> [<mask>]]]]",
"UPDATE                       reload the configuration without a reboot",
"HELP                         display command list",
"JTAG                         switch to JTAG command mode",
"BOOT [LOADER]                reset the BDI and reload the configuration",
"QUIT                         terminate the Telnet session"
```

### 3.5 Multi-Core Support

The bdiGDB system supports concurrent debugging of up to 8 PPC4xx cores connected to the same JTAG scan chain. For every core you can start its own GDB session. The default port numbers used to attach the remote targets are 2001 ... 2008. In the Telnet you switch between the cores with the command "select <0..7>". In the configuration file, simply begin the line with the appropriate core number. If there is no #n in front of a line, the BDI assumes core #0.

The following example defines two PPC405 cores on the scan chain.

```
[TARGET]
JTAGCLOCK      0          ;use 16 MHz JTAG clock
WAKEUP         1000       ;give reset time to complete

; 405-405-FPGA
#0 CPUTYPE      405        ;the target CPU type
#0 SCANPRED     0 0
#0 SCANSUCC     1 10       ;4 (405) + 6 (FPGA)
#0 SCANMISC     4 0xE0     ;IR length = 4, IR LSB = ..100000
#0 BREAKMODE    SOFT       ;SOFT or HARD

; 405-405-FPGA:
#1 CPUTYPE      405        ;the target CPU type
#1 SCANPRED     1 4        ;4 (405)
#1 SCANSUCC     0 6        ;6 (FPGA)
#1 SCANMISC     4 0xE0     ;IR length = 4, IR LSB = ..100000
#1 BREAKMODE    SOFT       ;SOFT or HARD
```

The following example works with the two 465 cores in a APM86290:

```
[TARGET]
; common parameters
JTAGCLOCK      0          ;BDI2000: use 16 MHz JTAG clock
WAKEUP         200        ;wakeup time after reset released
RESET          HARD 1000  ;assert cold reset for 1 second
;
;
; CoreID#0 parameters (active core after reset)
#0 CPUTYPE      APM86290 0 0 ;core#0 in SOC#0
#0 STARTUP      HALT RUN    ;halt after reset, run after resume from power-down
#0 BREAKMODE    HARD        ;SOFT or HARD, HARD uses PPC hardware breakpoint
#0 STEPMODE     HWBP        ;JTAG or HWBP, HWBP uses one or two hardware breakpoints
#0 SCANPRED     1 10       ;count for SOC TAP
#0 SCANSUCC     1 4        ;count for core#1 TAP
;
; CoreID#1 parameters
#1 CPUTYPE      APM86290 1 0 ;core#1 in SOC#0
#1 STARTUP      WAIT        ;don't handle until selected via Telnet
#1 BREAKMODE    HARD        ;SOFT or HARD, HARD uses PPC hardware breakpoint
#1 STEPMODE     HWBP        ;JTAG or HWBP, HWBP uses one or two hardware breakpoints
#1 SCANPRED     2 14       ;count for SOC and PPC0 TAP
#1 SCANSUCC     0 0        ;no TAP after PPC1 TAP
;
```



### 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 0x03 ; restart core #0, #1
STOP [<cores>]	Force one or multiple cores to debug mode. If there is no <cores> parameter, the currently selected core is forced to debug mode (stopped). <cores>          core bit map Example:          halt 0x03 ; stop 2 cores #0, #1
JMCDCCR <value>	For APM86xxx based SOC's. Set the JTAG Multi-core Debug Control Register. Can be used to stop / restart cores simultaneously.

### Following the bit definition in JMCDCCR:

```

STO0 (0x80000) Put core #0 into stop state (cleared by BDI)
STO1 (0x04000) Put core #1 into stop state (cleared by BDI)
STO2 (0x00200) Put core #2 into stop state (cleared by BDI)
STO3 (0x00010) Put core #3 into stop state (cleared by BDI)

STP0EN1 (0x40000) Stop core #0 when core #1 has a debug event
STP0EN2 (0x20000) Stop core #0 when core #2 has a debug event
STP0EN3 (0x10000) Stop core #0 when core #3 has a debug event

STP1EN0 (0x02000) Stop core #1 when core #0 has a debug event
STP1EN2 (0x01000) Stop core #1 when core #2 has a debug event
STP1EN3 (0x00800) Stop core #1 when core #3 has a debug event

STP2EN0 (0x00100) Stop core #2 when core #0 has a debug event
STP2EN1 (0x00080) Stop core #2 when core #1 has a debug event
STP2EN3 (0x00040) Stop core #2 when core #3 has a debug event

STP3EN0 (0x00008) Stop core #3 when core #0 has a debug event
STP3EN1 (0x00004) Stop core #3 when core #1 has a debug event
STP3EN2 (0x00002) Stop core #3 when core #2 has a debug event

```

**APM86xxx Multi-Core example:**

Stop both core simultaneously using the JMCDCR STOP bits:

```
MBA#0> state
Core#0: running
Core#1: running

MBA#0>jmcdcr 0x84000
- TARGET: core #0 has entered debug mode
- TARGET: core #1 has entered debug mode

MBA#0> state
Core#0: stopped 0x00000900 JTAG stop request
Core#1: stopped 0xfffa21e4 JTAG stop request
```

Start both core simultaneously:

```
MBA#0>jmcdcr 0x84000
MBA#0>cont 3
MBA#0>state
Core#0: running
Core#1: running
```

Stop core #1 when core #0 has debug event:

```
MBA>state
Core#0: stopped 0x0ffd1150 single step
Core#1: stopped 0xfffa21e4 JTAG stop request
MBA>bi 0x0ffd1148
Breakpoint identification is 0
MBA#0>jmcdcr 0x86000
MBA#0>cont 3
- TARGET: core #0 has entered debug mode
- TARGET: core #1 has entered debug mode
MBA>state
Core#0: stopped 0x0ffd1148 instruction breakpoint
Core#1: stopped 0xfffa21e4 JTAG stop request
```

## PPC476 (LSI ACP3448) Multi-Core example:

Via Telnet all the PPC476 defined Multi-core related register are accessible. Some special DCR register numbers map to these registers. With the appropriate entries in the register definition file these registers are accessible by name with the Telnet "rd" and "rm" commands.

```
; Special DCR's to access ACP3448 Multi-Core Debug Registers
;
mcdhltr      DCR      0x1001 ;low byte defines 7-bit JTAG instruction
mcdhlts      DCR      0x1009 ;low byte defines 7-bit JTAG instruction
mcdgrp0      DCR      0x1011 ;low byte defines 7-bit JTAG instruction
mcdgrp1      DCR      0x1019 ;low byte defines 7-bit JTAG instruction
mcdldbo      DCR      0x1021 ;low byte defines 7-bit JTAG instruction
;
dbimask      DCR      0x1100
dbomask      DCR      0x1101
;
```

When the Telnet "cont" command is used to restart cores, then the BDI first prepares the selected cores for restart (clears stop bit in JDCR), then clears all LDBO bits and finally clears all GRPHLT bits. This way all cores start running together. But don't forget to set the appropriate GRPHLT bit before using "cont".

In the following example all 4 cores are part of group #0. The example starts all cores together and all will halt once core#0 hits the breakpoint. We setup the Multi-core debug registers via the [INIT] section in the configuration file.

```
; Setup Multi-Core Debug Group
#0 WREG mcdgrp0      0xF0000000 ;GRP0 includes all cores
#0 WREG dbimask      0x80000000 ;DBIMASK: enable DBI[0]
#1 WREG dbimask      0x80000000 ;DBIMASK
#2 WREG dbimask      0x80000000 ;DBIMASK
#3 WREG dbimask      0x80000000 ;DBIMASK
#0 WREG dbomask      0xFFFC0000 ;DBOMASK: enable all events
#1 WREG dbomask      0xFFFC0000 ;DBOMASK
#2 WREG dbomask      0xFFFC0000 ;DBOMASK
#3 WREG dbomask      0xFFFC0000 ;DBOMASK
;
```

```
acp3448#0>stat
Core#0: stopped 0xfffff00c single step
Core#1: stopped 0xfffffff0 JTAG stop request
Core#2: stopped 0xfffffff0 JTAG stop request
Core#3: stopped 0xfffffff0 JTAG stop request
acp3448#0>bi 0xfffff018
Breakpoint identification is 0
acp3448#0>rm mcdhlts 0x80000000
acp3448#0>cont 0x0f
- TARGET: core #0 has entered debug mode
- TARGET: core #1 has entered debug mode
- TARGET: core #2 has entered debug mode
- TARGET: core #3 has entered debug mode
acp3448#0>stat
Core#0: stopped 0xfffff018 instruction breakpoint
Core#1: stopped 0xfffff05c JTAG stop request
Core#2: stopped 0xfffff058 JTAG stop request
Core#3: stopped 0xfffff058 JTAG stop request
acp3448#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 ACP3448 configuration example below. 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.

Cross-Trigger setup example (for the complete configuration look at acp3448lcm.cfg):

```
[INIT]
;
... other configurations ...
;
; Setup Multi-Core Debug Group
#0 WREG mcdgrp0      0xF0000000 ;GRP0 includes all cores
#0 WREG dbimask      0x80000000 ;DBIMASK: enable DBI[0]
#1 WREG dbimask      0x80000000 ;DBIMASK
#2 WREG dbimask      0x80000000 ;DBIMASK
#3 WREG dbimask      0x80000000 ;DBIMASK
#0 WREG dbomask      0xFFFC0000 ;DBOMASK: enable all events
#1 WREG dbomask      0xFFFC0000 ;DBOMASK
#2 WREG dbomask      0xFFFC0000 ;DBOMASK
#3 WREG dbomask      0xFFFC0000 ;DBOMASK
;
...
[TARGET]
; common parameters
JTAGCLOCK      16000000      ;16MHz JTAG clock
WAKEUP         100          ;wake up time after reset released
;
; CoreID#0 parameters (active core after reset)
#0 CPUTYPE      476
#0 CGROUP       0x0f         ;GDB continue core group master
...
```

### 3.6 Low level JTAG mode

It is possible to switch to a mode where you can enter low level JTAG commands via the Telnet interface. You activate this mode via the Telnet "jtag" command. Once the BDI has entered this mode, a new set of Telnet commands is available.

```
"TRST      {0|1}          assert (1) or release (0) TRST",
"HALT      {0|1}          assert (1) or release (0) HALT",
"CLK       <count> <tms>  clock TAP with requested TMS value",
"RIR [+] <len>            read  IR, zero is scanned in",
"RDR [+] <len>            read  DR, zero is scanned in",
"WIR [+] <len> <...b2b1b0> write IR, b0 is first scanned",
"WDR [+] <len> <...b2b1b0> write DR, b0 is first scanned",
"XIR [+] <len> <...b2b1b0> xchg  IR, b0 is first scanned",
"XDR [+] <len> <...b2b1b0> xchg  DR, b0 is first scanned",
"          +      : more data follows",
"          do not exit shift-IR/DR state",
"          len : the number of bits 1..256",
"          bx  : a data byte, two hex digits",
"RFILE <len> <file> [<succ>] dump DR to file, zero is scanned in",
"WFILE <len> <file> [<pred>] write DR from file",
"DELAY  <10...50000>      delay for n microseconds",
"HELP                                display JTAG command list",
"EXIT                                terminate JTAG mode"
```

Using this special JTAG mode is not necessary for normal debugging. But it maybe helpful to debug JTAG connection problems. For example to discover how many JTAG devices are on the scan chain.

Following a short session that analyzes the scan chain. It puts first all TAP's into bypass mode and then shifts a 1 through the DR. This 1 is shifted 3 bits left, so this tells us that there are 3 TAP's present on the scan chain. Also IR outputs ...111111 0000000001 0001 0001 indicating that the IR length of the 3 devices is 10-4-4.

```
Core#0>jtag
JTAG>xir 32 ffffffff
fffc0111
JTAG>xdr 32 00010000
00080000
```

## 4 Specifications


Operating Voltage Limiting	5 VDC $\pm$ 0.25 V
Power Supply Current	typ. 500 mA max. 1000 mA
RS232 Interface: Baud Rates	9'600, 19'200, 38'400, 57'600, 115'200
Data Bits	8
Parity Bits	none
Stop Bits	1
Network Interface	10 BASE-T
Serial Transfer Rate between BDI and Target	up to 16 Mbit/s
Supported target voltage	1.8 – 5.0 V (3.0 – 5.0 V with Rev. B)
Operating Temperature	+ 5 °C ... +60 °C
Storage Temperature	-20 °C ... +65 °C
Relative Humidity (noncondensing)	<90 %rF
Size	190 x 110 x 35 mm
Weight (without cables)	420 g
Host Cable length (RS232)	2.5 m

Specifications subject to change without notice

## 5 Environmental notice

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

## 6 Declaration of Conformity (CE)

  
**DECLARATION OF CONFORMITY**  
This declaration is valid for following product:  
**Type of device: BDM/JTAG Interface**  
**Product name: BDI2000**  
The signing authorities state, that the above mentioned equipment meets  
the requirements for emission and immunity according to  
**EMC Directive 89/336/EEC**  
The evaluation procedure of conformity was assured according to the  
following standards:  
**EN 50081-2**  
**EN 50082-2**  
This declaration of conformity is based on the test report no.  
QNL-E853-05-8-a of QUINEL, Zug, accredited according to EN 45001.  
Manufacturer:  
**ABATRON AG**  
**Stöckenstrasse 4**  
**CH-6221 Rickenbach**  
Authority:  
  
**Max Vock**  
Marketing Director  
  
**Ruedi Dummermuth**  
Technical Director  
Rickenbach, May 30, 1998

## 7 Warranty and Support Terms

### 7.1 Hardware

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

### 7.2 Software

#### License

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

#### Copies

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

#### Update and Support

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

### 7.3 Warranty and Disclaimer

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

### 7.4 Limitation of Liability

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

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



## Appendices

### A Troubleshooting

#### Problem

The firmware can not be loaded.

#### Possible reasons

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

#### Problem

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

#### Possible reasons

- Wrong pin assignment (BDM/JTAG connector) of the target system (see chapter 2).
- Target system initialization is not correctly --> enter an appropriate target initialization list.
- An incorrect IP address was entered (BDI2000 configuration)
- BDM/JTAG signals from the target system are not correctly (short-circuit, break, ...).
- The target system is damaged.

#### Problem

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

#### Possible reasons

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

## B Maintenance

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

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



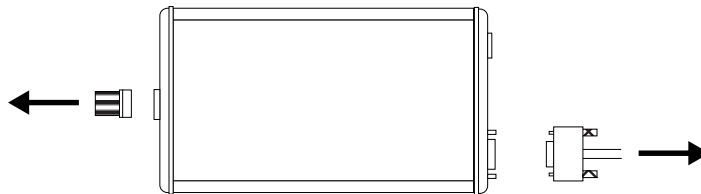
**Observe precautions for handling (Electrostatic sensitive device)**

**Unplug the cables before opening the cover.**

**Use exact fuse replacement (Microfuse MSF 1.6 AF).**

**1**

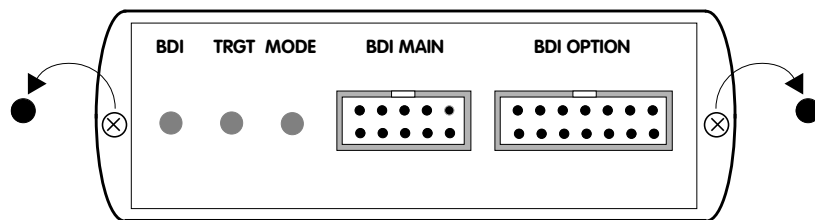
1.1 Unplug the cables



**2**

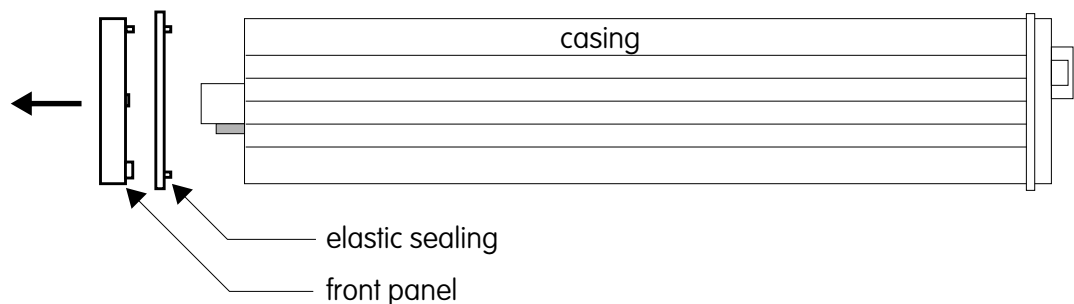
2.1 Remove the two plastic caps that cover the screws on target front side (e.g. with a small knife)

2.2 Remove the two screws that hold the front panel



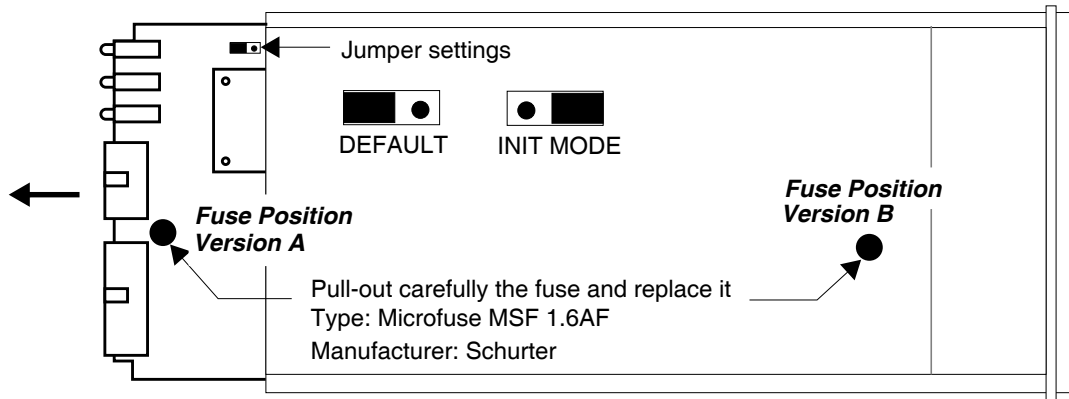
**3**

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



## 4

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

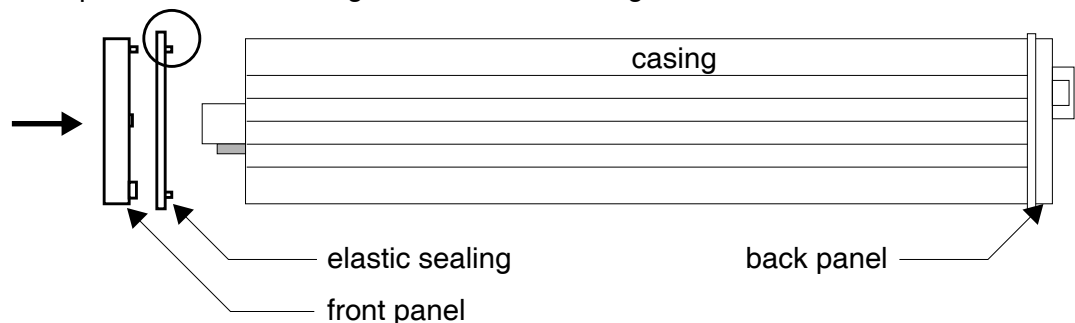


## 5

### Reinstallation

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

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



5.3 Mount the screws (do not overtighten it)

5.4 Mount the two plastic caps that cover the screws

5.5 Plug the cables



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

## **C Trademarks**

All trademarks are property of their respective holders.