ICEbear User Manual

Martin Strubel <hackfin@section5.ch>

17.5.2006

Revision:
v1.0.1

Preface

The ICEbear is a USB JTAG connector designed to run with the Blackfin CPUs from Analog Devices and the GNU development tools via the Blackfin Emulation Library (libbfemu - see Chapter 2). The libbfemu is part of the ICEbear package and provides a documented API to test, debug and remote control Blackfin hardware via JTAG.The main applications that the ICEbear is designed for, are:

- Target programming and debugging
- Flash programming
- Hardware test for production

Blackfin™ is a registered trade mark of Analog Devices Inc. Cygwin™ is a registered trademark of Red Hat Inc. Linux™ is a registered trademark of Linus Torvalds. Windows™ is a registered trademark of Microsoft Inc.

The ICEbear is a part of an evaluation or development kit and *not* intended to be an end customer device. The developer is responsible for compliance with the local EMI rules when integrating the ICEbear into a system.

The ICEbear uses RoHS compliant components since May 2006.

0.1 Warranty

section5 warrants that the Hardware ("Hardware") shall be free from material defects in design, materials, and workmanship and will function, under normal use and circumstances, materially in accordance with this documentation for a period of *one year* from the date of delivery. Components or the design of the Hardware are subject to change without notice, as long as functionality or operation are not affected.
Defective Hardware returned to section5 within the warranty period will be replaced and sent back to the customer at no charge, solely upon confirmation of a defect or failure of Hardware to perform as warranted.
The foregoing warranties shall be void due to any of the following:

1. if the Hardware has been opened, modified, altered, or repaired, except by section5 or its authorized agents
2. if the Hardware has not been installed or maintained or used in accordance with instructions provided by section5
3. misuse, abuse, accident, thermal or electrical irregularity (voltage excess), fire, water or other peril
4. damage caused by containment and/or operation outside the environmental specifications for the Hardware
5. connection of the Hardware to other systems, equipment or devices or use with other third party software without the prior approval of section5
6. removal or alteration of identification labels or EEPROM serials on the Hardware or its parts
7. failure to comply with all warranty return terms and conditions as set forth herein

To request a warranty service, please contact section5 via the link provided in Appendix B, describing the problem or malfunction. section5 will contact the Customer and confirm the warranty request. The Customer must package the Hardware in the original or appropriate

packaging such that further defects during shipping can be avoided. Shipments without confirmation of the warranty request will not be accepted. section5 is not liable for loss or damage during shipment and requires the shipping to be insured for its full value outside the EC.
section5 shall not be responsible for any software, information, or memory data of Customer contained in, stored on, or integrated with any Hardware returned to section5 for replacement whether under warranty or not. Customer is responsible for backing up its programs and data to protect against loss or corruption.
The Hardware and Software is not designed, manufactured or intended for use in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems, direct life-support machines, or any other application in which the failure of the Hardware could lead directly to death, personal injury, or severe physical or property damage or environmental damage (collectively, "High Risk Activities"). section5 expressly disclaims any express or implied warranty of fitness for such High Risk Activities.

1 Getting Started

1.1 Supported platforms

This is the list of currently well tested and stable platforms:

1. ADSP-533, 537 and 561 EZKIT lite from Analog Devices, Inc.
2. STAMP BF533, BF537 from Analog Devices, Inc.
3. DSPStamp from Cambridge Signal Processing Ltd.
4. ZBrain BF533 core module and eval platform from Schmid Engineering AG

If connecting the ICEbear to custom hardware, make sure that the JTAG interface was designed according to the application note EE-68 ([bib_ee68] App. B).
The software supporting the ICEbear is working under Linux and Windows NT 32 bit operating systems.
If you would like to see support for other platforms, or get your hardware listed above, please contact us (Appendix B, please also send a short test report).

1.2 Installing the software

This section contains the installation instructions for various system configurations. It is assumed, that you either have a CD that came with the ICEbear or that you have downloaded the software at the section5 homepage.

1.2.1 Windows/Cygwin

It is very recommended to install a Cygwin system on your Windows PC. However, the basic software does not require Cygwin to run. See http://www.cygwin.com for more info. Alternatively you may use MSYS/MinGW (http://www.mingw.org).

Run the Self Installer (`ICEbear-<version>.exe`, on the CD, this is in `win32`). The installation process should be self explaining. If you have trouble installing the drivers, you can manually install them by right clicking on the file `ftd2xx.inf` in the `drivers` folder and choosing **Install**. When experiencing difficulties, please refer to Appendix A.
After finishing the installation, plug in the ICEbear to complete the driver installation. You can safely ignore the Windows Certification warnings. Now you can start the programs from the ICEbear program menu.
Note that your personal firewall (which might be active on your machine) must allow the port 2000 to be accessed locally. If you get a warning when starting gdbproxy, you can safely unblock it.

1.2.2 Linux

The ICEbear is known to run smoothly with the following configuration:

- Kernel 2.4.X or later
- libusb-0.1.7 or greater

For 2.6 kernels, there is an issue with the ftdi_sio driver, preventing ICEbears with Rev. B being properly acknowledged. See Troubleshooting section (Section A.3) for a workaround. With Rev. C (sold after 1.4.2006), this problem is solved. You may have an early release of the Rev. C adapter which does not have the proper identification code. In this case, check the download section for the icebear-pid-updater tool.
Depending on your linux distribution, you will want to choose an installation method from the sections below. The ICEbear software package contains some support files (examples, scripts) that might be important for the user. See package description below about the location of these files.

Debian

It is assumed that you have a recent debian installation like the following (tested):

- Debian 3.1 (sarge)
- Knoppix 3.2 or later

Install the Debian packages in the `linux/debian` subdirectory according to the `README` file. You need to be root for this action.
To automatically update these packages via the Debian web based package manager (apt-get), you may want to put the following entry in your APT configuration file `/etc/apt/sources.list`:

```
deb http://www.section5.ch/dsp/icebear/debian sarge main non-free
```

Update your package list with the command `apt-get update`. Then you can automatically install or upgrade your ICEbear software with the command `apt-get install icebear-gdbproxy`. Note that you may have to add other Debian mirrors to `sources.list`, if library dependencies (libftdi0, libusb) can not be resolved.

Important support files

- Documentation (you will have to install the libbfemu-dev package also): `/usr/share/doc/libfemu-dev/html/index.html`
- Examples: `/usr/share/doc/libfemu-dev/examples`

RPM based Linux distributions

For Redhat package based systems (Redhat, SuSe, etc.), the above Debian packages have been converted to the RPM format - see `linux/rpm` subdirectory. However, they have not been extensively tested. If you experience problems, please report them.

Any distribution (from tar file)

Make sure you have one of the newer kernels (2.4.XX or later) with USB device filesystem enabled and libusb installed. Unpack the ICEbear tar file by
`tar xfz ICEbear-<version>.tgz`
In the `lib/` folder you will find the necessary DLLs (.so's) to link against. If you are not installing them into one of the standard library directories, add their path into the `LD_LIBRARY_PATH` environment variable and modify your Makefile library flags ('-L'), if necessary.
If you are installing the source package, you will have to compile and install each software separately. Read the installation notes inside the directories.

1.3 Connecting the ICEbear

First, remove the power from the board or unplug the ICEbear from the USB port. Never plug the ICEbear into the board while any JTAG software is running.
Connect the ICEbear as shown in Fig. 1.1 below. The JTAG connector has a key on pin 3 which prevents it from plugging it in the wrong way.

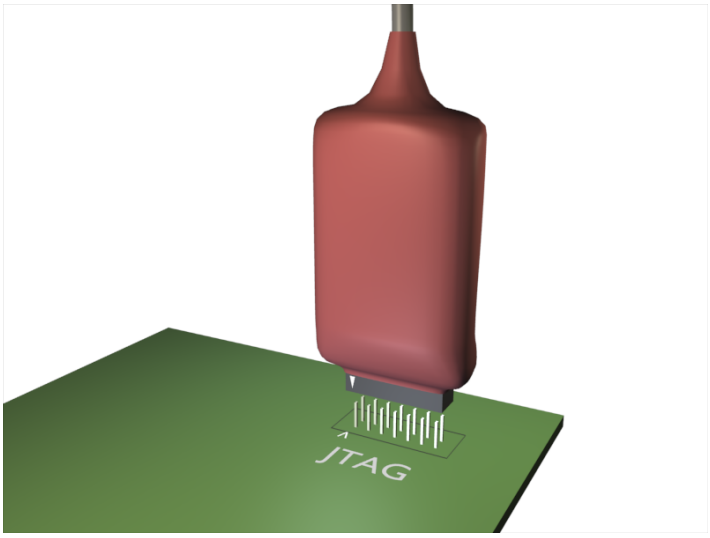

Figure 1.1: Connecting the ICEbear to the JTAG port

Do not connect the ICEbear to a STAMP board that was modified to run with a ByteBlaster (R4 bypass). This will destroy either your board or the ICEbear and make all warranty void.

When removing the ICEbear, do not pull on the cable to avoid cable stress which can damage the USB connection to the ICEbear. Hold the bottom connector firmly with two fingers and gently wiggle it out lengthwise. Do not apply too much force to not bend your JTAG pins or stress your PCB.

1.4 Running the software

To test whether the adapter is running, execute one of the examples, for instance in Linux:
`icebear-dumpreg`
or **Run Demos** from the ICEbear program menu in Windows.
If everything is properly connected, this will give you a register dump of the current cpu state.
Alternatively, you can start gdbproxy and fire up the debugger - see next chapter.

2 Software

Please note that there is no free support for the GNU toolchain. This software is supplied AS IS, please read the GNU license and usage statements. However, we will try our best to keep the GNU toolchain working optimally with the *ICEbear*. New with the 1.0 release is the Insight debugger. This is a GDB port made by Red Hat and Cygnus Solutions, adapted to Blackfin by section5. More details below.
Developer documentation is found in [bib_bfemu] App. B.

2.1 gdbproxy

For debugging a JTAG hardware target, **gdbproxy** serves as a remote debugging daemon which the GNU Debugger (GDB) can connect to. gdbproxy translates all the GDB commands into the appropriate JTAG sequences via libbfemu, which is driven by an added target 'bfin'.
To get help about the target specific options for gdbproxy, type `gdbproxy --help bfin`
Omitting the target choice 'bfin' shows the common options.
An important option is the `--speed` option. For example, to use maximum speed of the ICEbear, start gdbproxy with the following arguments:

```
gdbproxy bfin --speed=0
```

The complete list of options is show below.

Option name	Description
`speed=`<value>	Change JTAG clock speed value. The maximum JTAG clock frequency on the ICEbear (6 MHz) will divide by the value plus one. The default speed value is 1.
`buffersize=`<size>	Change write cache buffer size to specified size. If 0, disable write cache.
`debug`	Enables debug output mode. Use this mode only, when you suspect a bug in the target handling

Table 2.1: gdbproxy Blackfin specific options

The source code of gdbproxy (v0.7.3) is freely available under the author's (Quality Quorum, Inc.) license.

2.2 Insight Debugger

The Insight debugger is a GDB variant with built in GUI. It has some extra features and is very powerful. For beginners, it is recommended that you start with Insight. Note that Insight is free software, source code is available at http://sources.redhat.com/insight/.

2.2.1 Setting up and configuring Insight

Before you can start debugging your target, you will have to do a bit of setting up your environment. For example, if the executable that you want to run on your target lives in SDRAM, you will have to do the necessary setup of the SDRAM controller (Synchronous EBIU).

You can do all these setups manually by writing to a specified address, but GDB scripting can simplify that task greatly.

Auxiliary scripts

A few auxiliary GDB scripts are provided in the location below. These are:

mmr.gdb Some of the important MMR register definitions

init.gdb Some example initialization scripts for standard boards

dump.gdb Some register dump helper functions to display interrupt status, etc.

catch_exc.gdb Example on how to catch exceptions

OS-dependent locations:

Windows	`$(PROGRAM_FILES)/section5/ICEbear/scripts`
Debian Linux	`/usr/share/gdbscripts`

To load these scripts, either use the 'source' command or load it via the Menu **File->Source**. You can then use these register definitions for example in a Watch window (see below in Section 2.2.2).

.gdbinit Startup script

For a specific project, you certainly do not want to repeat initialization procedures over and over. Therefore, it is recommended to write a `.gdbinit` script in your project's directory. When you start **Insight** from that directory, it will automatically load the `.gdbinit` script. The sample script below demonstrates how to automatically connect to the target (provided that gdbproxy is running) and do the necessary initialization. Note that you have to strip the comments (include the '#') when copy pasting the file.

```
file blink.dxe                  # Select blink.dxe executable
target remote :2000             # Connect to the local gdbproxy

source ../scripts/mmr.gdb       # Load MMR definitions
source ../scripts/dump.gdb      # Load Dumper functions

set prompt (bfin-jtag-gdb)\     # Set the prompt

define target_init              # Define target initialization function
    monitor reset               # Reset the target
end
```

If your program uses SDRAM, you will have to put some SDRAM initalization code in your function 'target_init', like below:

```
define target_init              # Target initialization with SDRAM
    monitor reset
    set *$EBIU_SDGCTL = 0x0091998d
    set *$EBIU_SDBCTL = 0x0025
    set *$EBIU_SDRRC  = 0x0817
end
```

Note that this example is for a specific board. See `init.gdb` for a few default initializations. For your own board, you must possibly adapt these values to SDRAM configuration and system clock. See your Blackfin Hardware Reference.

If the EBIU and system settings are not correct, the Blackfin can core fault on SDRAM access. This can not be caught by the debugger. You will then have to reset the system.

You can also put system wide declarations in a .gdbinit script residing in your home directory. Note however, that all variable declarations are cleared when you switch executables. So, calling `mmr.gdb` from `$(HOME)/.gdbinit` has no effect. However, a function definition ('define') remains. If you are accidentally including a script with function definitions twice, Insight will complain.
Insight is not very communicative on startup script errors. Before trusting a script, test it by starting Insight with the '-n' option (no startup script execution) and load it explicitely in the console using the 'source' command. Whenever a user defined command such as `target_init` fails, the console will display an error message.

When starting Insight in Windows via the Start menu, the .gdbinit script in the scripts directory (as specified in Section 2.2.1) is executed by default. You can change the working directory via the **Properties** menu of the Insight debugger's red bug icon (Right click on the icon to call this menu).

Setting up Insight

Before you can connect to the target, you need to specify some parameters. This is done by calling the configuration dialog below (Fig. 2.1) via **File->Target Settings**.
When your program fails and you need to reconnect and download again, you would want to call the initialization function `target_init()` whenever you connect or run the program.Click on 'More Options' to expand the dialog like shown below and enter 'target_init' in the text field.

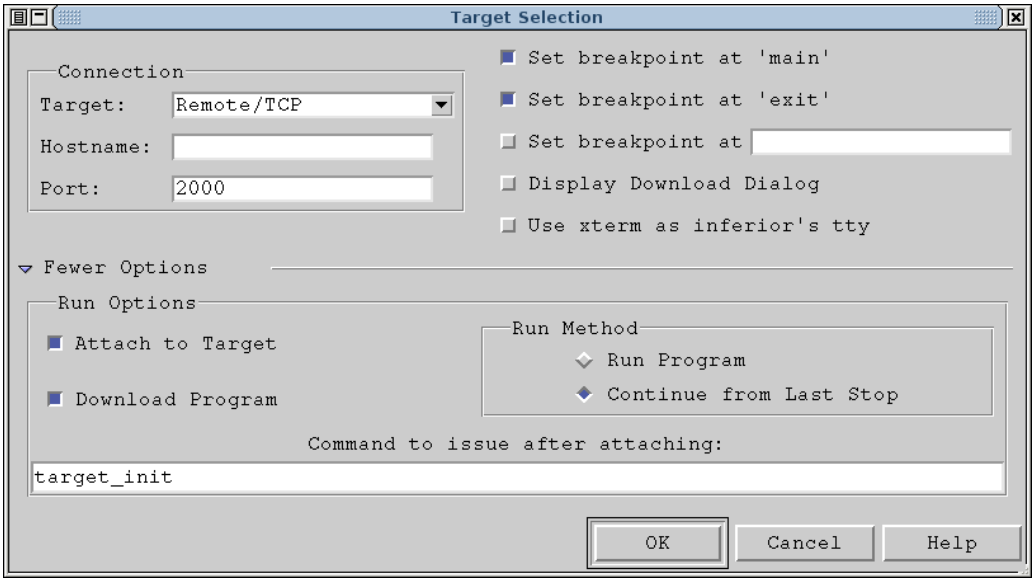

Figure 2.1: Target configuration

After you have set up your target, make sure that gdbproxy is running and has properly detected your target. Then you can connect to the target within Insight via **Run->Connect to target**.The next step is, to download the program (**Run->Download**) and call ' Continue'. You can also execute ' Run' immediately. The target will then halt at the first breakpoint ('main', as specified in the Target Selection dialog).
When you need to reload the program, you can either cycle through **Disconnect/Run** via the **Run** menu or define a script with the necessary initializations.

Special commands

Please refer to the GDB documentation for all GDB commands. All 'monitor' commands are implemented in gdbproxy and considered 'special'. These are:

monitor reset Resets the target (core and system)

All GDB commands can be entered via the console window (**View->Console**).

2.2.2 Examining programs and variables

Source code and assembly debugging

By default, and if you have compiled your program with the '-g' flag, you will see the native source code (C, C++, Assembly) in the main window. The current location of the program counter is marked by a green bar. With the drop down menus on the top you can select other files and functions. On the right drop down, you can change the display mode. For example, the mixed mode displays interleaved C source and disassembly as shown in Fig. 2.2.

Note that mixed or assembly mode display can take a long time to read from the target, when your program lives in L1 onchip memory. If you want to avoid that, use the technique explained below.

If you run a program without debugging information, there is no source code or disassembly display available. You can still look at the executed opcodes via the console (**View->Console**). The command `display/i $pc` displays the assembly command at the current PC location. For convenience, you can step into or step over using the `si` and `ni` command on the console.

Simply pressing 'Return' on the console repeats the last command

Variable watching

You can view the current content of a variable any time by moving the mouse pointer over its location in the source code. It will automatically retrieve the value and display it. Note that this can take a long time on variables such as buffers or strings. You can turn this behaviour off via **Preferences->Source->Variable Balloons**.
If you want to repeatedly watch a register or global variable, you can use the Watch window (opened via **View->Watch Expressions**). For example, if you want to look at the interrupt state of the core, you can display the IPEND register (which is defined in mmr.gdb) by entering the expression '`*$IPEND`' into the text field left to the Add_Watch button like shown below (Fig. 2.3). Normally, variables are displayed as decimal. If you want to change the display, click with the right mouse button on the variable entry and choose via the **Format** menu.

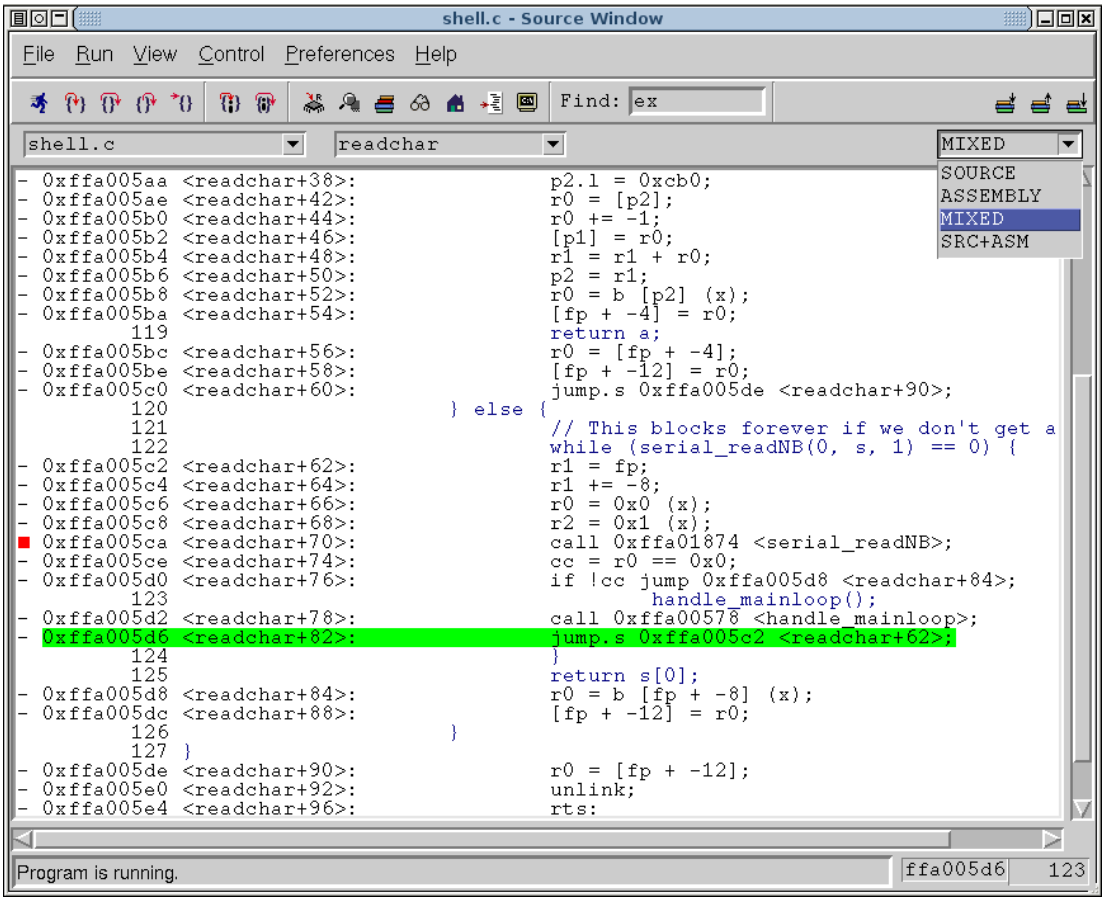

Figure 2.2: Source Window

2.2.3 Finding bugs

Finding a bug can sometimes be an art, especially on a complex embedded system as a Blackfin driven platform. Most common errors when starting with a standalone developent (without uClinux) on Blackfin systems are due to hardware or other exceptions. Generally, it is a good idea to install exception handlers in L1 SDRAM and set break points on the exception handler routines - look at the `catch_exc.gdb` script. This must obviously be called *after* the exception vectors have been initialized. For concrete tips on how to track down a problem, please see 'Debugging Tips' section in the the support forum at http://www.section5.ch/forum/. A few general hints are listed below.
If Insight stalls and does not react to user input after a certain sequence of stepping over the code, it is mostly not to blame to a debugger flaw. Find out by reproducing the situation what exactly happens on the target. Single instruction stepping should always work, however, stepping over C source code can sometimes jump into subroutines and single step through them which can take a long time. If you can wait, it helps to be patient. In case of such events, you can try to track down what is going on by several ways:

- Look at the gdbproxy console. If there are core faults, they mostly are due to a failing exception handler, for example a handler living in SDRAM which was not properly initialized. Such errors cause double faults and make the core stall. You then have to reconnect (within Insight) or reset the target via 'monitor reset' or in heavy cases by pressing the reset button.

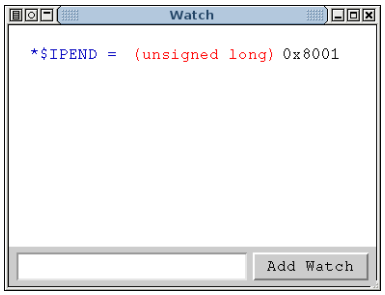

Figure 2.3: Watch window

- Enable the debug mode of gdbproxy by enabling the `--debug` option. If you are lost, send a dump of this output to section5, possibly we can help you.
- Write a script that single steps though the code until a certain event occurs. This can be done with a while/end loop, see GDB script syntax in the GDB online manuals

2.3 bfin-jtag-gdb (GDB)

This is the hardware debugging version of the GDB port for Blackfin.The core functionality is the same as of Insight's. Note that this GDB version is different from the bfin-elf-gdb variant provided by blackfin.uclinux.org and no longer actively supported by section5.
A HOWTO on getting started is found in the bfemu documentation. Documentation about GDB in general can be retrieved using the 'info gdb' command on Linux or from various web resources (http://www.gnu.org).You may also want to try the graphical debugger frontend DDD which is available on most Linux systems. To run bfin-jtag-gdb with ddd, execute

```
ddd -debugger bfin-jtag-gdb <program to debug>
```

To save yourself some typing, you may want to define an alias in your `.bashrc` or `.profile` startup file.
The source code of this GDB version is available under the Gnu Public License.

2.4 bfloader

bfloader stands for 'Blackfin Flash Loader' and consists of a collection of GDB scripts and a standalone executable to program the flash of common Blackfin targets. This software is provided for free and as source code. Users can easily adapt the source for the backend target to their customized hardware. The bfloader distribution can be downloaded from http://www.section5.ch/software/. Documentation can be found in the `doc/` sub folder of this package. Below, only the standalone flash loader commands are listed.

2.4.1 Flashloader Usage

This standalone executable does not require gdbproxy to run. Parameters are passed with option flags like shown below:

```
./flashload --info --backend=zbrain/bfloader.dxe --eraseall
--program=zbrain/vmImage
```

The description of command options is listed in Table 2.2. Running the flashloader without arguments displays a list of supported options.

Command options

Note that all file related actions (program, dump, compare) can only be used at a time.

Examples

Here is a list of examples on how to use the flashloader. Note that this assumes the appropriate backend (`bfloader.dxe`) in the current working directory.

Erase and program a flash with a full flash image `./flashload --eraseall --program=flash.img`

Program a flash with a partial flash image from offset 0x10000 `./flashload --program=flash.img --offset=0x10000`

Partial erase of flash `./flashload --erasesectors=0,32`

Verify flash content against binary file image `./flashload --compare=flash.img`

Read first 1M partition of a boot flash `./flashload --dump=flash_read.img --offset=0x0 --size=0x100000`

2.5 libbfemu

The Blackfin Emulation library *libbfemu* (or bfemu) is a library which supports you in testing your hardware via JTAG. In conjunction with the ICEbear, it is a very important component of a hardware production chain. A short feature overview:

- Register read/write
- Memory read/write
- Program sequencer control: Stop, go, single step
- Software breakpoints. Hardware breakpoints are currently not used due to Blackfin anomalies.

libbfemu comes with examples. The examples have been tested under Linux and MinGW/Cygwin.

Option name	Description
info	Displays information about flash driver and flash geometry
version	Displays the program version
program=<filename>	Write the binary image in file with name filename to flash. If none of the size and offset options are specified, offset is assumed 0 and size the size of the file.
dump=<filename>	Dump the content of the flash specified by offset and size to a file. Specifying size is mandatory.
compare=<filename>	Compare flash content with the binary image in the given file. If specified, the comparison is started from offset up to size. Otherwise, offset is assumed to be 0 and size to be the size of the file.
eraseall	Erase entire Flash chip. This can take up to a few minutes, depending on the flash chip size.
erasesectors=<off>,<n>	Specify the number of sectors n to erase, starting from sector number off.You must know the sector geometry of the Flash in order to determine the corresponding addresses. This option is only relevant when you use a partitioned flash.
offset=<Address in hex>	Specifies an offset in hexadecimal format. You must use a prefix '0x' for this hexadecimal number, e.g. --offset=0x1000.
size=<Size in hex>	Specifies a maximum size to read, write, or compare. The format is the same as in the offset option.
noverify	Turn of verify after write.Verify is always on by default, if this option is not specified.
backend=<backend>	Specify the loader backend to load on initialization. If not specified, the default compiled in backend name ('bfloader.dxe' in the current working directory) is used. This must be a valid bfloader backend, VDSP backends are not compatible.
speed=<value>	Change JTAG clock speed. 0: fastest, 255: slowest. Note that too low clock speeds (values > 4) might not work. Default is 1 (3 MHz TCLK).

Table 2.2: List of command options

3

Technical Specifications

These specifications only apply when connected to approved hardware as listed under Section 1.1.

Supported Blackfin CPUs	BF531, (BF532), BF533, (BF536), BF537, (BF539), BF561
USB compatibility	1.1, 2.0
Memory read speed	10kB/s(min), 80kB/s(typ), 130kB/s (peak)
Memory write speed	40kB/s(min), 120kB/s (typ), 240kB/s (peak)

Table 3.1: General features

Blackfin CPUs that are listed in brackets are supposed to be recognized, but not extensively tested.
Memory access times depend on OS and on used block sizes. Via GDB/Insight, the download speed can be significantly lower.

VCC Supply Voltage	5 V (USB bus powered), 6V maximum
Supply current	max. 100mA (typ. 25mA)
Operation Temperature Range	0 - 70°C

Table 3.2: Absolute maximum ratings

max. JTAG clock (TCLK)	6 MHz
TCLK rise time	typ. 6ns
Output voltage levels	L: 0V H: 3.3V +- 5%
max. DC output current	20mA
Mean Time between JTAG failure	not measureable within 150 hours

Table 3.3: JTAG operation

A

Troubleshooting

This is a collection of the most common errors that we could think of. If your question is not answered here, please contact section5, see Appendix B.

A.1 General

The ICEbear is not recognized

See Section A.3 for more information.

The software recognizes the ICEbear, but not the target board

Make sure the JTAG connector is properly attached and that the power is supplied to the board. If your board is not listed in the hardware support list (Section 1.1), make sure that the JTAG connector has the pinout and voltage specifications according to [bib_ee68] App. B . See also Section 1.1 for more information. If you can not solve the problem, consult us.

A.2 gdbproxy errors

This section helps you to determine a connection problem according to the output of gdbproxy (>v0.8). When gdbproxy fails to recognize the target or the ICEbear, you will in general get the error message: `error: Could not open ICEbear connection`. The following possible errors are explained in detail below:

`error: Make sure that no other program/driver (ftdi_sio?) is claiming the port`

You have a 2.6.X kernel: see section 'USB problems'. Are you sure there is no gdbproxy or other program running and grabbing the device?

`error: Could not detect target, check connection and power`

The ICEbear was recognized, but connection to the target could not be made. That means, that the target ID was not properly read. Please go through the following check list:

- Is your Blackfin CPU in the list of supported CPUs?
- Are you sure that there is no USB problem such as listed below (Section A.3)?
- Does your JTAG chain only contain the Blackfin device?
- Could there be issues with the JTAG frequency or extra cable lengths to the target? See also 'Target hardware issues' below. If this randomly works, this might be well the case.

A.3 USB problems

Windows: The driver shows a yellow '!' mark in the Device Manager

Try to reinstall the driver by right-clicking on the device icon, choosing **Uninstall...**, and installing the FTDI driver again. Try running the FTClean utility from the FTDI website (www.ftdichip.com) to clean up old driver relicts. If the problem persists, check whether the ICEbear works on another computer. If this fails, please contact us.

Make sure you always install the driver before plugging in the ICEbear!

Windows: The ICEbear is not recognized

Do you have any USB to serial port converter installed on the system? Try again with unplugging this converter and re-plugging the ICEbear, possibly to another port. The ICEbear software will support several USB adapters in future releases.

Windows: The ICEbear does not properly work anymore after interrupting a test program

When hitting Ctrl-C on a console program such as **memtest.exe**, it may happen on some systems, that the ICEbear can not be initialized again after. This is something we can not fix (except by catching Ctrl-C), as obviously the USB driver failed to free resources when the program was cancelled. Workaround: Unplug the ICEbear, wait a few seconds, and plug it back again.

Linux 2.4.X kernels: The ICEbear is not recognized

Try the following to recover from the problem

1. Log in as 'root' and type 'dmesg' after plugging in the ICEbear. If the following message does not appear, your kernel might not properly be configured for USB (missing usbdevfs module?). If you contact us, report your Linux distribution.

   ```
   hub.c: new USB device 00:04.2-1, assigned address 4
   usb.c: USB device 4 (vend/prod 0x403/0x6010) is not claimed by any active driver
   ```

   Also, check `/proc/bus/usb/devices`. The above vendor/product number should be identified as from FTDI.
2. Make sure that no other software is using the device, for example another USB port serial driver. If a kernel module `ftdi_sio` is active, remove it (`rmmod ftdi_sio`).
3. Make sure you have the permissions to access the USB port in `/proc/usb/<hub_number>`. To grant a user the permission, use an entry like the one below in your `/etc/fstab`.

   ```
   usbdevfs /proc/bus/usb usbdevfs   defaults,devmode=0666   0    0
   ```

Linux 2.6.X kernels: ICEbear not recognized

Try the following steps:

1. Type `dmesg` after plugging in the ICEbear. If it was properly recognized, the following message appears:

   ```
   usb 2-1: new full speed USB device using uhci_hcd and address 2
   usb 2-1: configuration #1 chosen from 1 choice
   ```

   If you have a Rev. B ICEbear, your Linux kernel might be configured to automatically load the `ftdi_sio` driver which claims the ICEbear device. Thus, the ICEbear software is not able to access the device. Check for the ftdi_sio module (`lsmod`) and remove it (as root) from the kernel after plugging in the ICEbear (`rmmod ftdi_sio`). You may also change your kernel module configuration such that ftdi_sio is not loaded automatically. This can be achieved by uncommenting the line in `/lib/modules/<your-kernel-version>/modules.usbmap` containing:

```
ftdi_sio   0x0003  0x0403  0x6010 ....
```

This prevents automatic loading of the ftdi_sio driver. If you use a FTDI based USB to serial converter in your system, you will have to load the ftdi_sio driver manually after plugging in the other adapter.

2. Look at /proc/bus/usb/devices. You should find an entry with the lines Revision B:

```
P: Vendor=0403 ProdID=6010 Rev=5.00
S: Manufacturer=FTDI
S: Product=Dual RS232
```

Revision C:

```
P: Vendor=0403 ProdID=c140 Rev=5.00
S: Manufacturer=section5
S: Product=ICEbear JTAG adapter
```

If you do not see any of these lines after plugging in the ICEbear, there might be something wrong with your USB connection. If the Rev. C ProdID does not match with the one listed, you may have to run an updater tool to avoid the ftdi_sio problem - see Section 1.2.2. If the ICEbear neither works under Windows, a hardware error has to be assumed - please consult us before returning the ICEbear for check-up.

3. Check user permissions: The file `/etc/fstab` should contain a line like:

```
usb         /proc/bus/usb   usbfs   defaults,devmode=0666   0   0
```

If in doubt, try running the ICEbear software as root to verify a permission problem.

A.4 Target hardware issues

Generally, when connecting to custom hardware, failures can occur due to too long JTAG cabling. You must keep your JTAG connection between CPU and ICEbear as short as possible. If this can not be achieved, try decreasing the JTAG clock speed (see Section 2.1).

Emulation not ready

If the error message "Emulation not ready" (ERR_NOTREADY) appears, the system may have hung completely and can not be reset. Try the hard reset button or reconnecting the power. When doing that, make sure that any program driving the emulation is no longer running.

SDRAM Memory test fails

If you get memory read or write failures when accessing SDRAM, this is most likely not an issue with the adapter itself. There can be several reasons:

1. The SDRAM is not configured correctly, either by your embedded application or by your program using bfemu. If you are using GDB, read the notes about initialization in [bib_bfemu] App. B.
2. Your target hardware can not handle the clock speed of the *ICEbear*. Try lowering the clock frequency by using the function `jtag_config()` from the bfemu library. Too low clock frequencies can again cause misfunction. Normally, you should only use clock wait cycles from 0 to 4. See also gdbproxy options (Section 2.1).

Register does not read correctly

Try to repeat the procedure several times when the target system is halted. If you still get garbage values, the JTAG connection is probably instable. This should never happen, if your board is among the ones in the supported platforms list (Section 1.1). Check for externally connected hardware that may cause a problem.

Target memory reading is much faster under Windows than under Linux

This is due to the USB driver implementation under Linux. Even if memory reading is implemented via an efficient command queue, it can not be accelerated further with the Linux driver.

B

Literature and Links

For remarks or support, please use the order/feedback form at
http://www.section5.ch/order.php.

B.1 Bibliography

A list of documents and further pointers:

[bib_bfemu] **bfemu - The Blackfin emulation library**
08/2005, section5::ms <hackfin@section5.ch>
URL: http://www.section5.ch/bfemu

[bib_ee68] **EE-68 JTAG Emulation Technical Reference Application Note**
Analog Devices
URL: http://www.analog.com