Part I
Arm&Eva on chip debugging using „BDI2000”

Contents

NOTE

All information, examples and software on this document is provided "AS IS" and "WITH ALL FAULTS", without warranty or conditions of any kind concerning the quality, safety, or suitability of the software, either express or implied, including, without limitation, any implied warranties or conditions of merchantability, fitness for a particular purpose, or non-infringement.

Wir haften nicht für eventuelle Schäden am BDI2000 die wg. Fehlbedienung entstehen... etc.

1 Preface

On chip debugging is necessesary for tracing applications at lowest possible level. It's is a non-intrusive method to find out mistakes in low-level applications.

Our evaluation board „Eva" offers the JTAG / ICE interface provided by ARM on a 20pin male connector. This connector may be used for boundary scan tests and even on chip debugging. There are various JTAG debugger available on the marked but only a few of them are ready for use with linux.

This chapter derscribes the usage of Abatron's BDI2000 on an example.

1.1 Preparations on your development PC

1.1.1 Installing the GNU-debugger

You will need a „gdb" version able to understand ARM-thumb instructions. The most linux distributions only provide gdb packages with support for the platform it runs (for instance iX86). We plan to provide a debian package (and tarball) with a precompiled version of gdb allowing to debug ARM code. All files will be located at: /usr/local/cross-dev/arm/gdb-build/gdb/

1.1.2 Installing tftp

A „*tftp*"-server is needed to get the BDI2000 running. On debian systems the package „tftpd" provides such a „trivial" ftp server. Additionaly you have to configure the servers directory. The default on debian is „/srv/tftp". The directory „/tftpboot" may be used on other distributions. Make sure the directory exists and is accessable for every one.
You have to place at least two files to this directory. These files provides the startup-configuration for the BDI2000 and a register definition file for the used MCU (AT91RM9200). In the first example the following files will be used:

1. „**carmeva.cnf**"
   (The startup / config-file for the BDI2000)

2. **„reg92000.def**"
   A register definition file for the used MCU. This file is a part of the BDI2000 distribution.

1.2 Configuring the BDI2000

You should read the BDI2000-manual to understand the following steps!
You will have to setup it's IP-address („BDI-IP"), the IP-address of the tftp-server (your development PC in most cases) and the name of the it's config-file („*carmeva.cnf*")

1.2.1 Testing the configuration

Turn on the BDI2000 **without** connecting the target system. Now the BDI2000 is reachable via telnet.

telnet *BDI-IP*

After these command you should be connected with the BDI2000 and see list of possible commands. If not please refer the BDI2000 manual to find out what's went wrong.

```
... (cutted)
bx  : a data byte, two hex digits
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>]]]]
HELP                        display command list
QUIT                        terminate the Telnet session

- CONFIG: loading configuration file passed
- CONFIG: loading register definition passed
# TARGET: Init JTAG communication failed
# TARGET: target has no power
# TARGET: target powerfail detected
- TARGET: waiting for target Vcc
- TARGET: waiting for target Vcc
```

1.3 Connecting the target

The connection will be done using the ARM adapter shipped with the BDI2000.

Note Turn off the BDI2000 and Eva before connecting booth together.

Figure 1: Connection between BDI2000 and Eva.

1.3.1 Setting up „ICE-Mode”

The debug port provides different modes. For use with the BDI2000 you have to setup the so called „ICE-Mode”. The mode will be selected by jumper „J3”. The following picture shows the configuration:

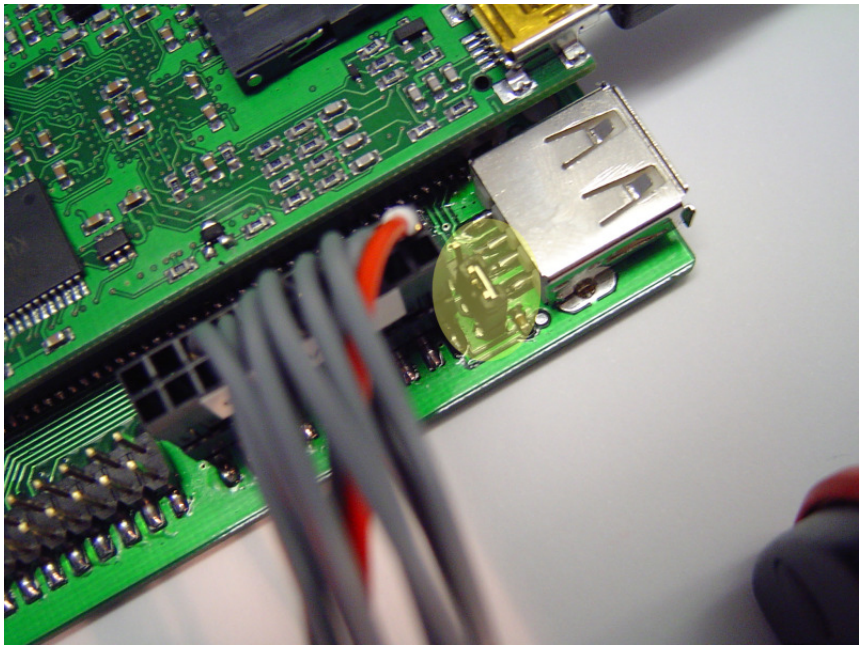

Figure 2: Setting up ICE-mode on Eva.

Power on sequence

According the manual of the BDI2000 you have to turn on the BDI2000 first, the Eva after that.
After connecting via telnet to the BDI20000 you may enter the „reset”-command.

```
Core#0>reset
- CONFIG: loading configuration file passed
- CONFIG: loading register definition passed
- TARGET: processing reset request
- TARGET: BDI asserts TRST and RESET
- TARGET: BDI removes TRST
- TARGET: Bypass check 0x00000001 => 0x00000001
- TARGET: JTAG exists check passed
- Core#0: ID code is 0x05B0203F
- TARGET: All ICEBreaker access checks passed
- TARGET: BDI removes RESET
- TARGET: BDI waits for RESET inactive
- TARGET: resetting target passed
- TARGET: processing target startup ....
- TARGET: processing target startup passed
Arm&Eva>
```

2 Example

We assume that you plan to debug an application („*cArmEvaLCD*”) executable in SDRAM. Before this program gets running the bootloader (LdBigAppUSB) initializes the SDRAM controller and PLL's. For transfering the applications you will use the „*ebSuite*” shipped with your Arm&Eva distribution.

Note

You have to remove the bootloader to start the application directly via USB!

To debug this application the following steps are required:

1. Build your application with debug-info
2. Power on your development environment
   (BDI2000 first, then Arm&Eva)
3. Transfer the application you wan't to debug („cArmEvaLCD”) using the „*ebSuite*”
4. Connect to the target with gdb
5. Debugging steps follow...

2.1 Build cArmEvaLCD with debug info

Take a look to the Makefile shipped with cArmEvaLCD. Locate the lines containing the „**CXX_FLAGS**" and „**C_FLAGS**". Debug info will be generated with the switch „**-ggdb**". Additionaly you **must not** use any kind of optimizations. So remove possible „-Os" from the compiler flags. Also remove „-fomit-frame-pointer" to keep information about the stack frames available.

Note

Optimizations will derange your debugger and are known pitfalls! It's also recommended connecting Arm&Eva via serial port (RS232) to see all low-level outputs.

Generated outupt

To build the application simply type:

```
make clean
make
```

Two interesting files will be generated:

1. /usr/local/carmeva/sw/boot/sw.arm9/**.**build/app/cArmEvaLCD/**cArmEvaLCD**
   (The binary executable on your target (ARM))
2. /usr/local/carmeva/sw/boot/sw.arm9/**.**build/app/cArmEvaLCD/**cArmEvaLCD.build**
   (An ELF-binary ready to load in gdb - this file contains the debug-info)

2.2 Power on

After powering up the BDI2000 and Eva you will see the „C"-character in your serial terminal connected with Arm&Eva. If not type „reset" in your BDI-telnet console and press the reset-button on Eva.
Now it's time to transfer the application.

2.3 Transfering the application to the target

Type:

```
./start
```

in /usr/local/carmeva/sw/user-applications/cArmEvaLCD.
Now the transfer should begin. If not press the reset-button on Eva again.

2.4 Connect to the target using gdb

One possibility is using gdb on console but there are also front-ends available.

2.4.1 Invoking plain gdb for remote debugging

```
(gdb) target remote 192.168.1.3:2001
```

If you are familiar with the command line interface of gdb you may use nealy all commands as wont. One exception is that you have to type „continue" instead „run" because the application has already been started and the MCU still waits.

2.4.2 Invoking a remote debug session using kdbg

For debugging ARM code your debugger has to support this kind of code. We choose a debugger compiled with support for ARM9-Thumb code.
This debugger is located at /usr/local/cross-dev/arm/gdb-build/gdb/gdb.

Assuming the BDI uses 192.168.1.3 as IP-Address type:

```
kdbg -r 192.168.168.1.3:2001 cArmEvaLCD.build
```

2.4.3 Using the integrated debugger of kdevelop3

Kdevelop3 offers the possibility for debugging inside the IDE. The following steps are required for setting up kdevelop3:

1. Write a small gdb startup-script containing all information needed for connecting to the target. An example script („gdb-remote.scr") is given below.
2. Configure kdevelop3 to use the startup-script
   Projects ▷ Project Options ▷ Debugger
   (a) „Directory where gdb resides:" /usr/local/cross-dev/arm/gdb-build/gdb/
   (b) „Run gdb script": gdb-remote.scr
3. Setup the „program to run"
   Projects ▷ Project Options ▷ Run Options
   (a) Main Program: cArmEvaLCD.**build**

GDB config script („gdb-remote.scr")

```
target remote 192.168.1.3:2001
```