ARM Cross-debugging with gdb

1 Installation

1. Get the latest gdb version
2. Extract the downloaded archive anywhere (YourPath)
3. Configure gdb

```
./configure –host=i686-pc-linux-gnu –target=arm-elf
```

4. Run make

```
make
```

You should have a binary named „gdb“ in the sub directory „gdb“. (YourPath/gdb/gdb)
This executable must be used when debugging your applications using **gdbserver** or via JTAG.

2 Getting Started

2.1 Preparations

2.1.1 Build your program

Before you start debugging your program you have to build it **without optimizations**. This can be done passing the **-O0** switch to the compiler. Additionaly you will need to generate debug info using the switch **-ggdb**. Example:

```
arm-9tdmi-linux-gnu -O0 -ggdb source.cpp -o source.o
```

2.1.2 Start the on chip debugger

.. to write ..

2.1.3 Ready to debug

If the on chip debugger has been started you can start debug. There are many frontends available for gdb - it's up to you which of them you prefer.

2.2 GDB configuration

You have to use gdb in remote mode. Furthermore there are settings concerning the onchip debugger itself. Because the settings are required for each debugging session a gdb initialization script has been made. (gdb.scr) Those gdb scripts can be passed to gdb using the **-x** command line switch.

2.3 Using the command line frontend

```
gdb -x gdb.scr
```

If you need a gdb reference take a look at: http://www.cs.dal.ca/studentservices/refcards/gdbref.pdf

2.4 Using kdbg

Make the following settings in „Global Settings“[1]. Make the following settings:

[1] You may also try „This Program settings“ but this was not working for me

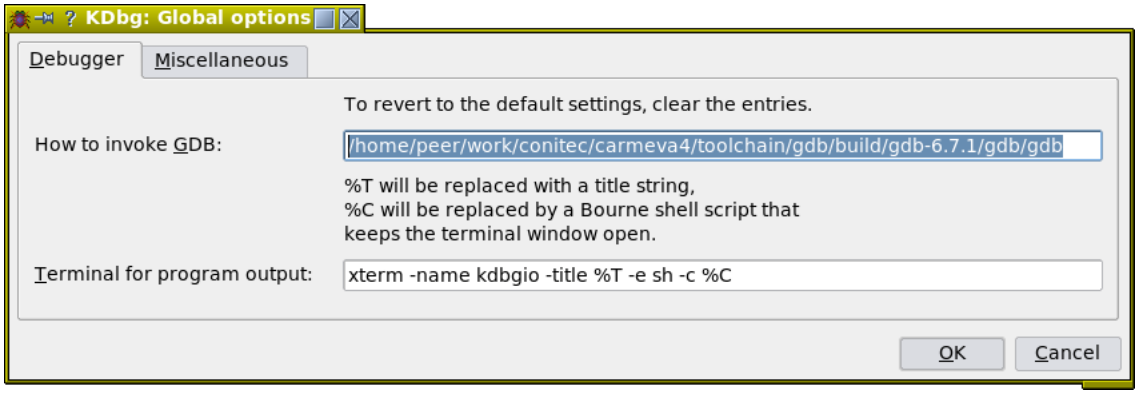

Abbildung 1: Kdbg settings.

Assuming the onchip debugger is reachable at the ip address 192.168.1.14 then start kdbg:

```
kdbg -r 192.168.1.14:3333 ../../.build/APP/cArmEvaLCD/cArmEvaLCD
```

The filename should be the ELF file generated by linker. (With debug info.)