Introduction

This technical note will go over the installation and operation of a Cortex M3 debug DLL in CodeLite. With this DLL and a few configuration changes to start-up, it is possible to do Cortex M3 debugging in CodeLite. It supports the following:

- Full Symbolic / Source Code based debugging
- Firmware Download
- Flash Breakpoints: The internal Cortex M3 breakpoints are not supported (yet).
- Run / Break / Pause
- All the data display functions (read RAM/FLASH, stack frames, local variables, etc.).

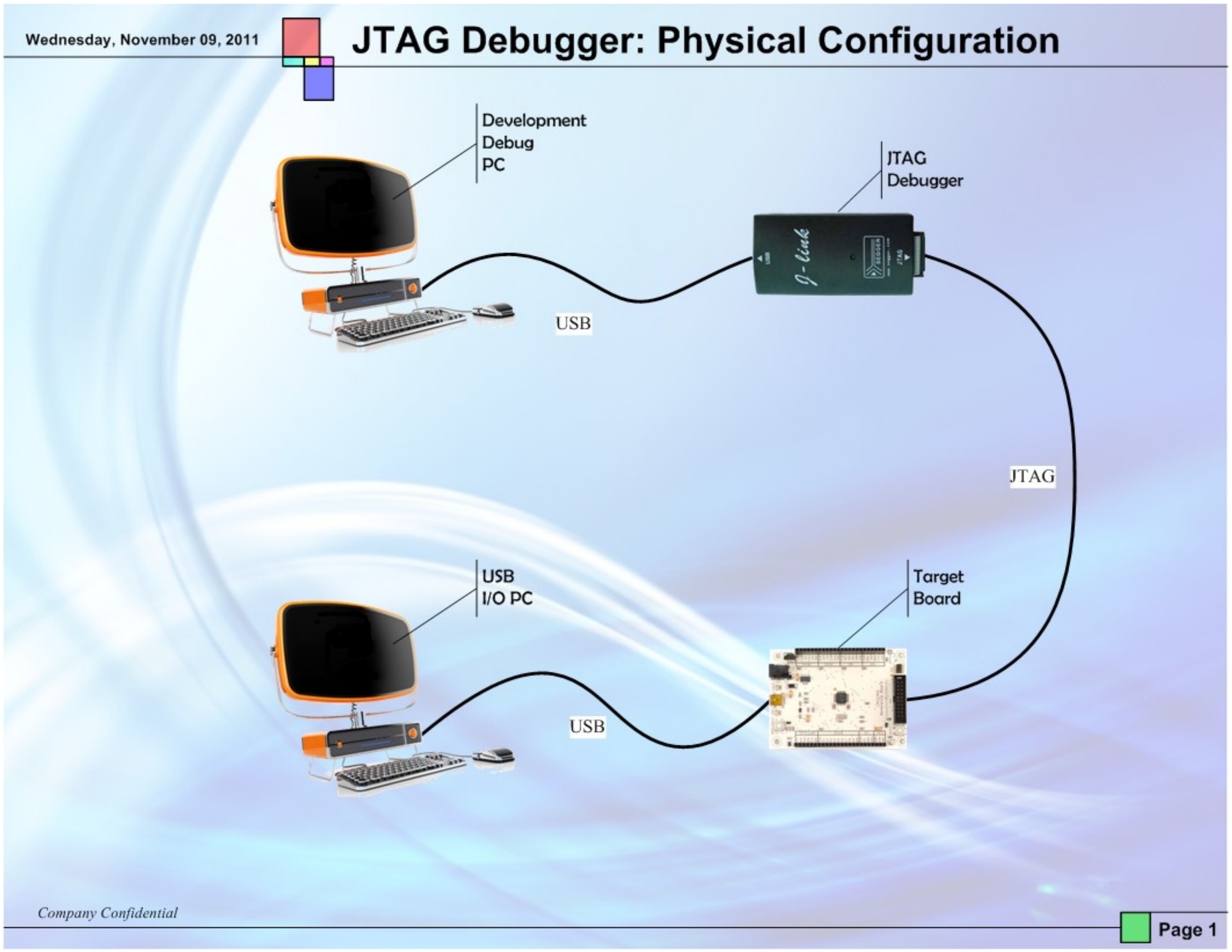
My Configuration:

- Win7 x64
- YARGTO 18.03.2011 (Binutils-2.21, Newlib-1.19.0, GCC-4.6.0, GDB-7.2)
- CodeLite v3.0.5041 and v3.0.5181..5186 (via svn)
- SEGGER J-Link GDB Server V4.36b
- SEGGER J-link
- MicroBuilder LPC1343 Reference Design
- Firmware code base is the LPC1343_CodeBase from microbuilder.eu.

Physical Configuration

The physical system assembly is straightforward:

1. There are three or four components in the physical debugging configuration
2. Download / Debug PC. This PC has the following:
   1. Target Development Tool Chain: In this case it is YARGTO.
   2. Target Debugger: In this case it is YAGARTO GDB
   3. Target firmware: In this case it's the Micro Builder LPC1343 Reference Design Firmware
   4. Appropriate connection to the target hardware.
3. The JTAG debugger. In this case it is the Segger J-Link.
4. The target board itself. In this case it is the Micro Builder LPC1343 Reference Design.
5. USB I/O PC. This PC powers the target board. It also runs the target CLI via USB. This PC is optional.



Wednesday, November 09, 2011

JTAG Debugger: Physical Configuration

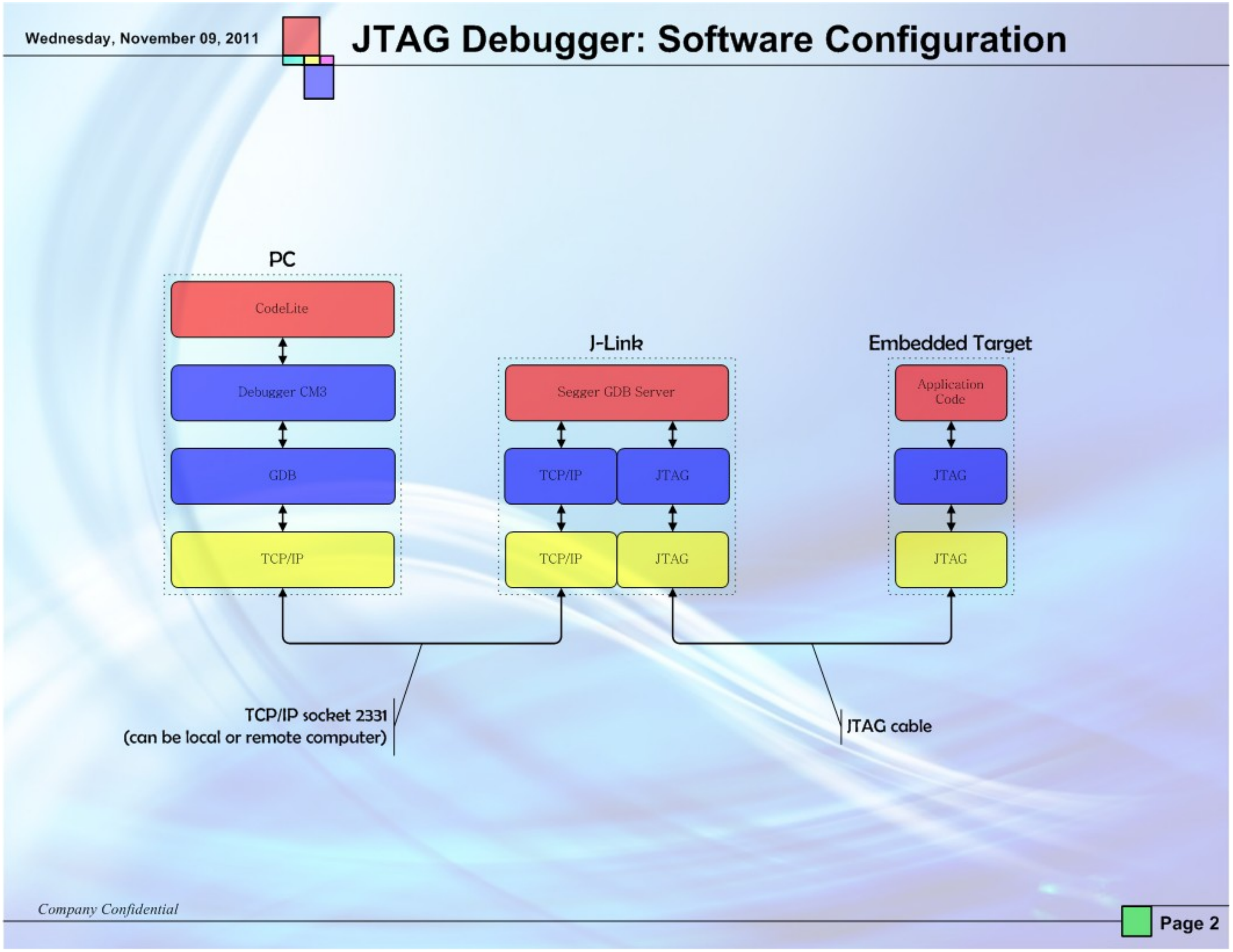
Company Confidential

Page 1

Software Configuration

There are three main software stacks involved in debugging:

1. On the PC
    1. CodeLite
    2. DebuggerGDB CM3. This has been customized to work with the the Segger GDB server.
    3. GDB. In this case it's the YAGARTO GDB
    4. GDB uses TCP/IP sockets to communicate to the GDB server.
2. J-Link
    1. Segger J-Link GDB Server
    2. To communicate to GDB, the Segger GDB server uses TCP/IP sockets.
    3. Segger GDB server communicates to the target platform via a USB link to a JTAG device
3. On the Target
    1. The target has JTAG hardware to interface with the Segger GDB server.
    2. Target firmware. In this case it's the Micro Builder LPC1343 Reference Design Firmware



Wednesday, November 09, 2011

JTAG Debugger: Software Configuration

Company Confidential

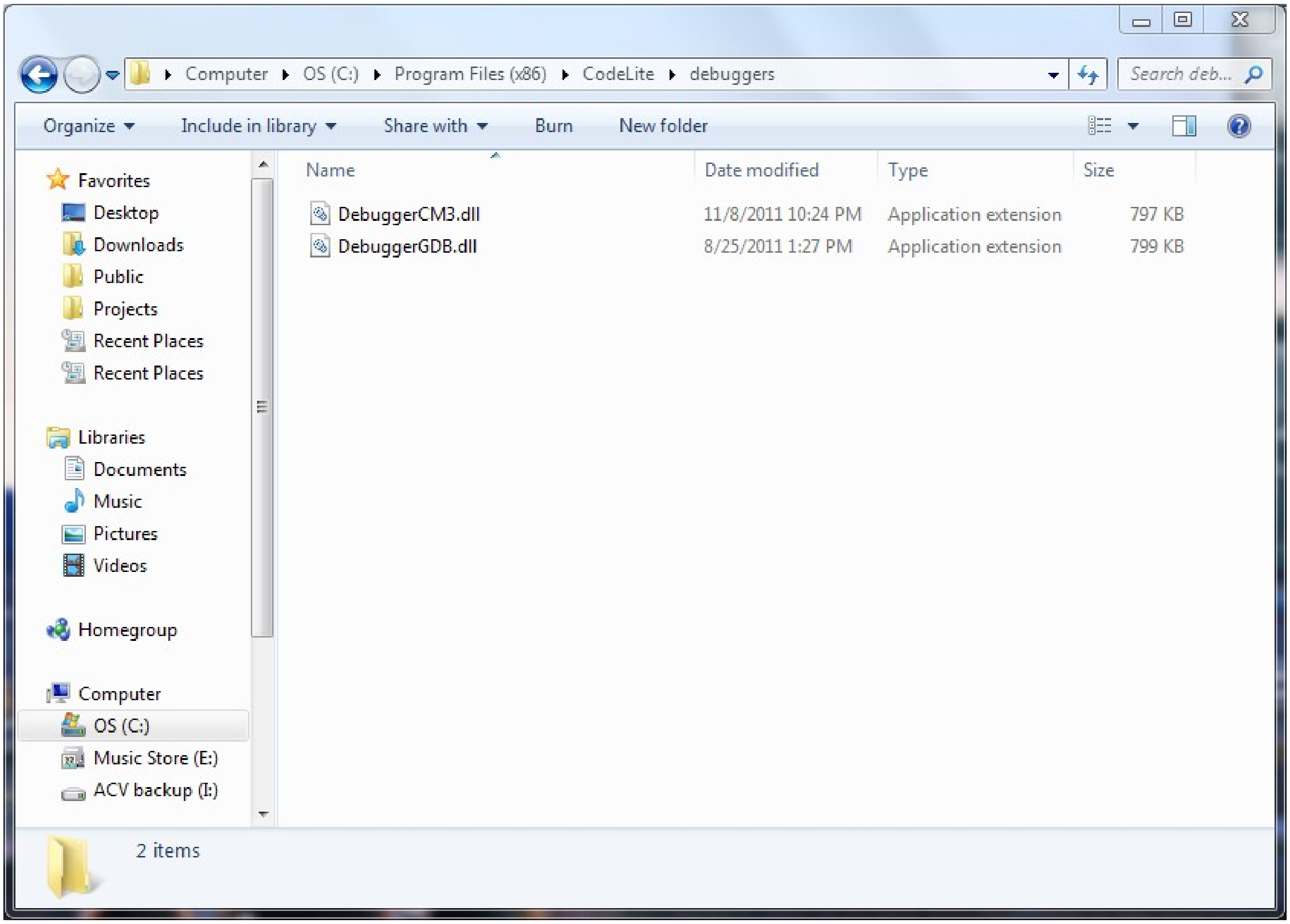
Page 2

Debugger DLL Installation

The debugger DLL installation is straightforward:

4. Make sure CodeLite is not running.
5. Open an explorer window.
6. Change to $(PROGRAMFILES)/CodeLite/debuggers.
   (e.g. C:\Program Files (x86)\CodeLite\debuggers)
7. Copy in the new debugger DLL: DebuggerCM3.dll

On my system it looks like this:



Name	Date modified	Type	Size
DebuggerCM3.dll	11/8/2011 10:24 PM	Application extension	797 KB
DebuggerGDB.dll	8/25/2011 1:27 PM	Application extension	799 KB

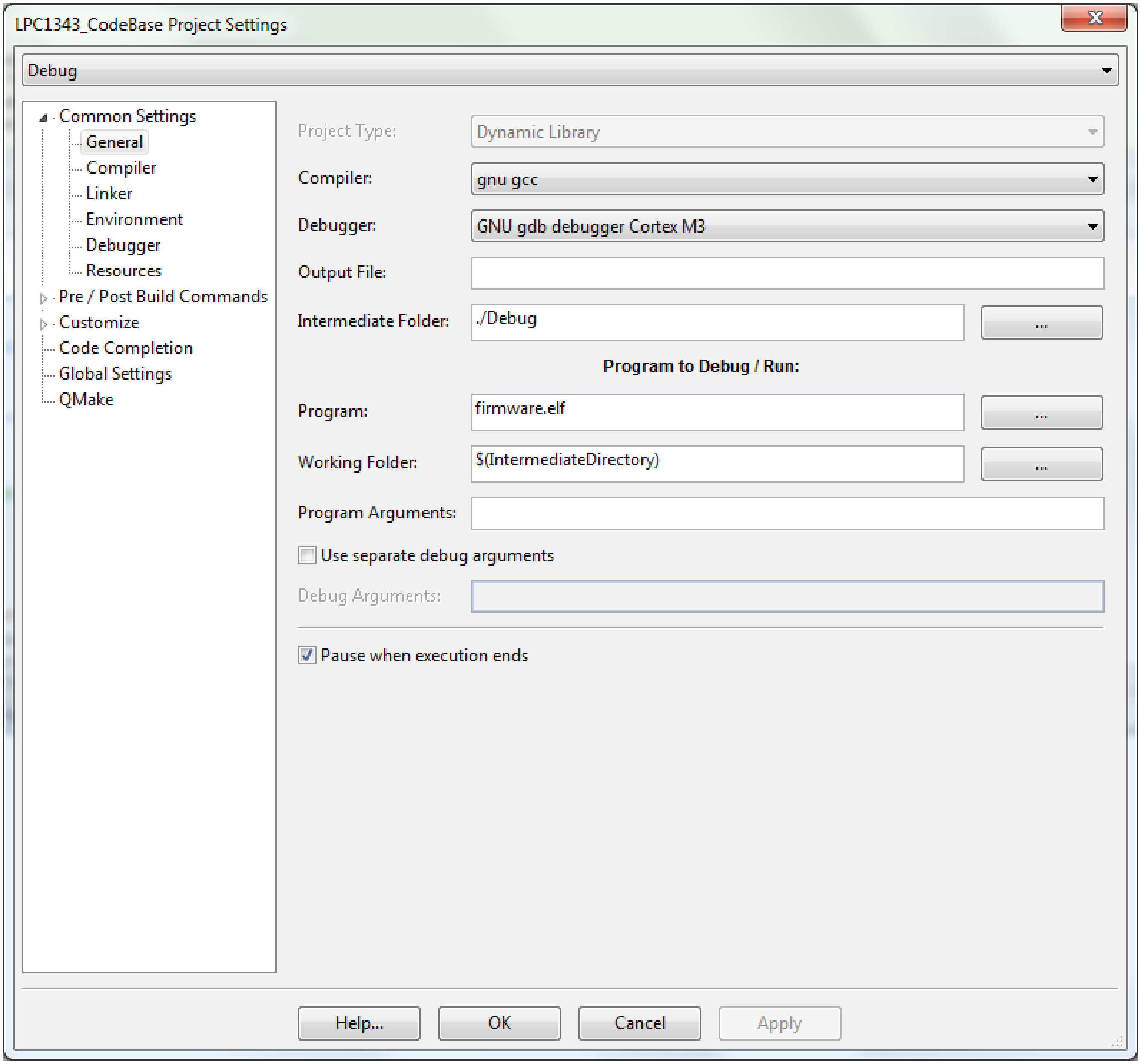
Initial CodeLite Configuration

In this section, we will do the following:

1. Enable the debugger DLL in the Project Settings
2. Configure the interface to GDB to work with the Cortex M3.
3. Modify the target code Makefile to place the elf in a CodeLite compatible location.
4. Rebuild the target with debugging enabled.
5. Start the debugger and begin debugging.

Here's how to configure CodeLite once DebuggerCM3.dll (Cortex M3 dll) is installed:

1. Start the J-Link DGDB Server.
2. Connect the target board: Power / USB and JTAG
3. Start CodeLite.
4. Load the target embedded project work space.
5. Open the project settings dialog.
6. Go to the General Dialog Page.



Project Type: Dynamic Library

Compiler: gnu gcc

Debugger: GNU gdb debugger Cortex M3

Output File:

Intermediate Folder: ./Debug

Program to Debug / Run:

Program: firmware.elf

Working Folder: $(IntermediateDirectory)

Program Arguments:

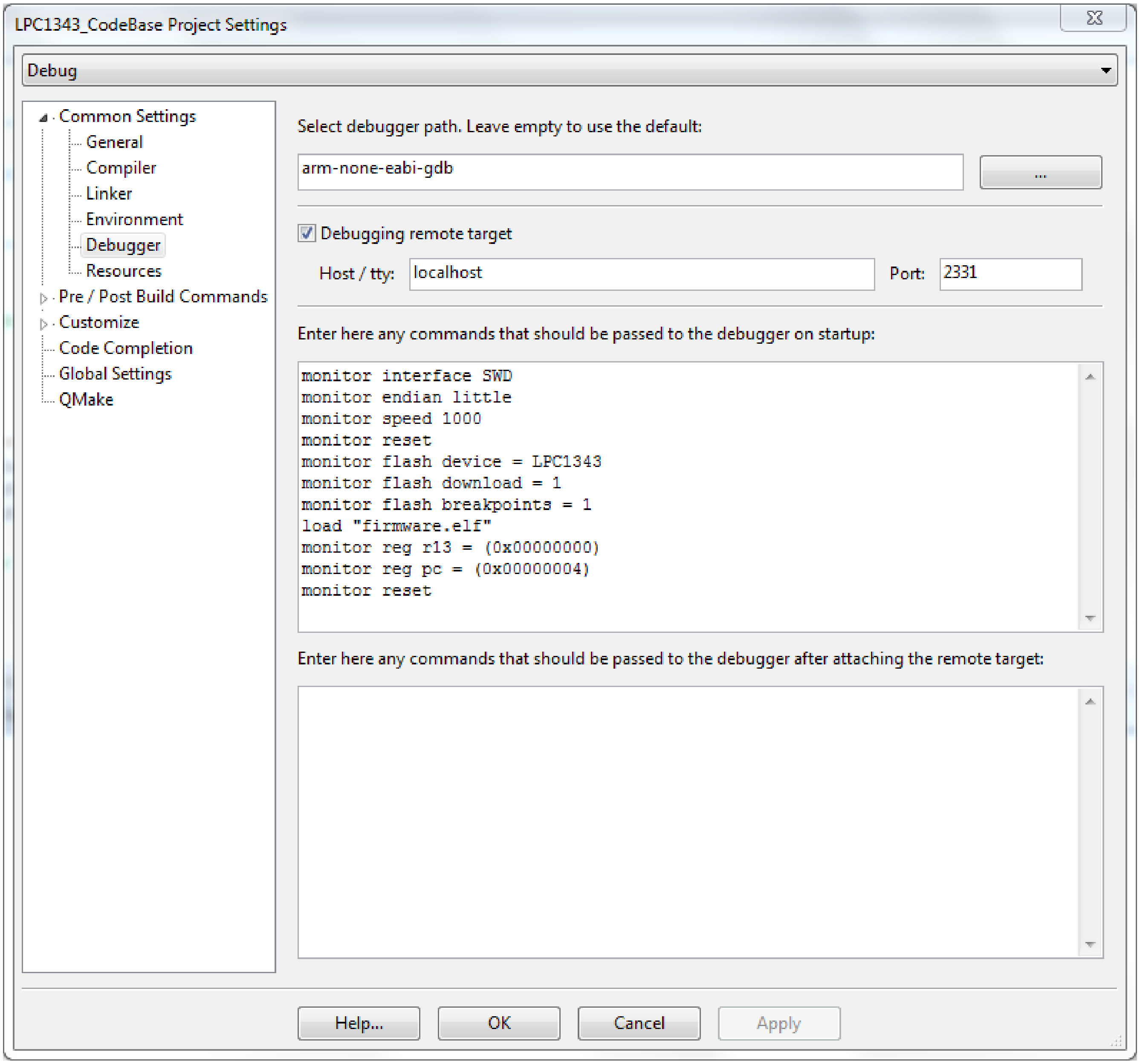
☐ Use separate debug arguments

Debug Arguments:

☑ Pause when execution ends

Configure the dialog to match. Specifically, make sure that the correct debugger is selected, the Intermediate Folder and Working Folders are set. And last, set the program file.

Go to the Debugger Page and set the following commands.



Select debugger path. Leave empty to use the default:

arm-none-eabi-gdb

☑ Debugging remote target

Host / tty: localhost

Port: 2331

Enter here any commands that should be passed to the debugger on startup:

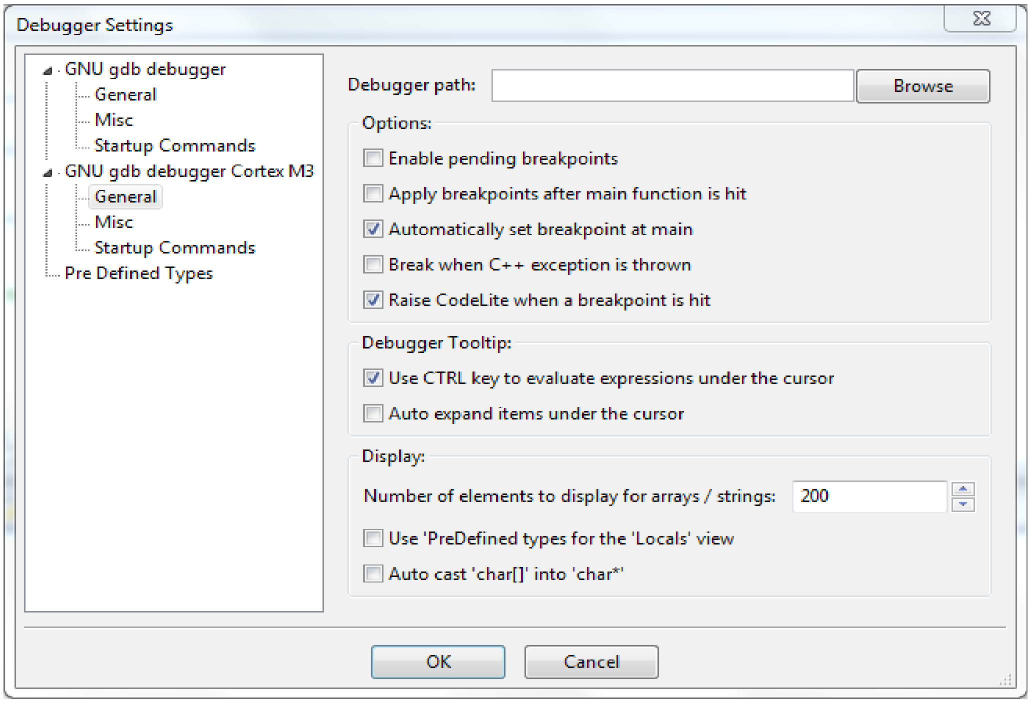
```
monitor interface SWD
monitor endian little
monitor speed 1000
monitor reset
monitor flash device = LPC1343
monitor flash download = 1
monitor flash breakpoints = 1
load "firmware.elf"
monitor reg r13 = (0x00000000)
monitor reg pc = (0x00000004)
monitor reset
```

Enter here any commands that should be passed to the debugger after attaching the remote target:

These commands initialize the connection to the target and load the firmware (symbols are loaded when gdb is started).

Apply changes and exit the dialog.

Go to Settings → Debugger Settings dialog from the main menu and set the following switches:



Debugger path:

Options:

☐ Enable pending breakpoints

☐ Apply breakpoints after main function is hit

☑ Automatically set breakpoint at main

☐ Break when C++ exception is thrown

☑ Raise CodeLite when a breakpoint is hit

Debugger Tooltip:

☑ Use CTRL key to evaluate expressions under the cursor

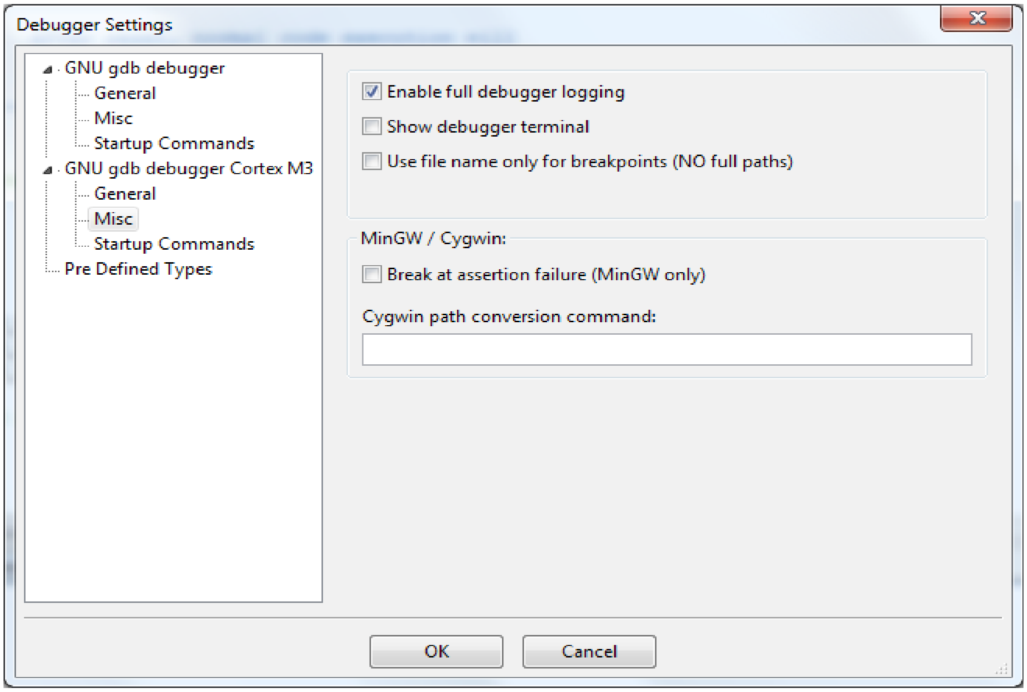
☐ Auto expand items under the cursor

Display:

Number of elements to display for arrays / strings: 200

☐ Use 'PreDefined types for the 'Locals' view

☐ Auto cast 'char[]' into 'char*'



☑ Enable full debugger logging

☐ Show debugger terminal

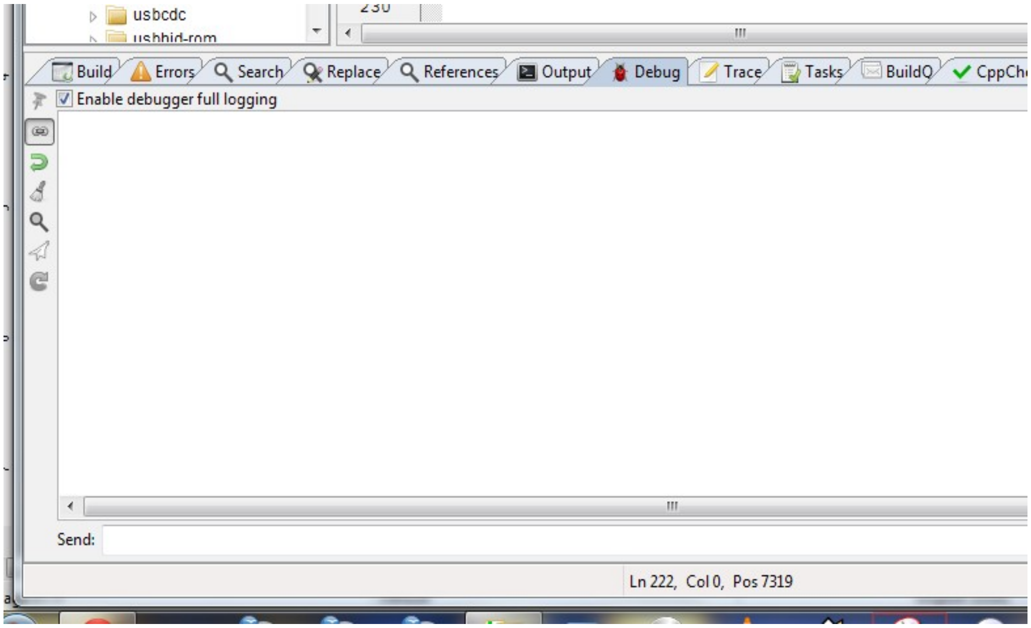
☐ Use file name only for breakpoints (NO full paths)

MinGW / Cygwin:

☐ Break at assertion failure (MinGW only)

Cygwin path conversion command:

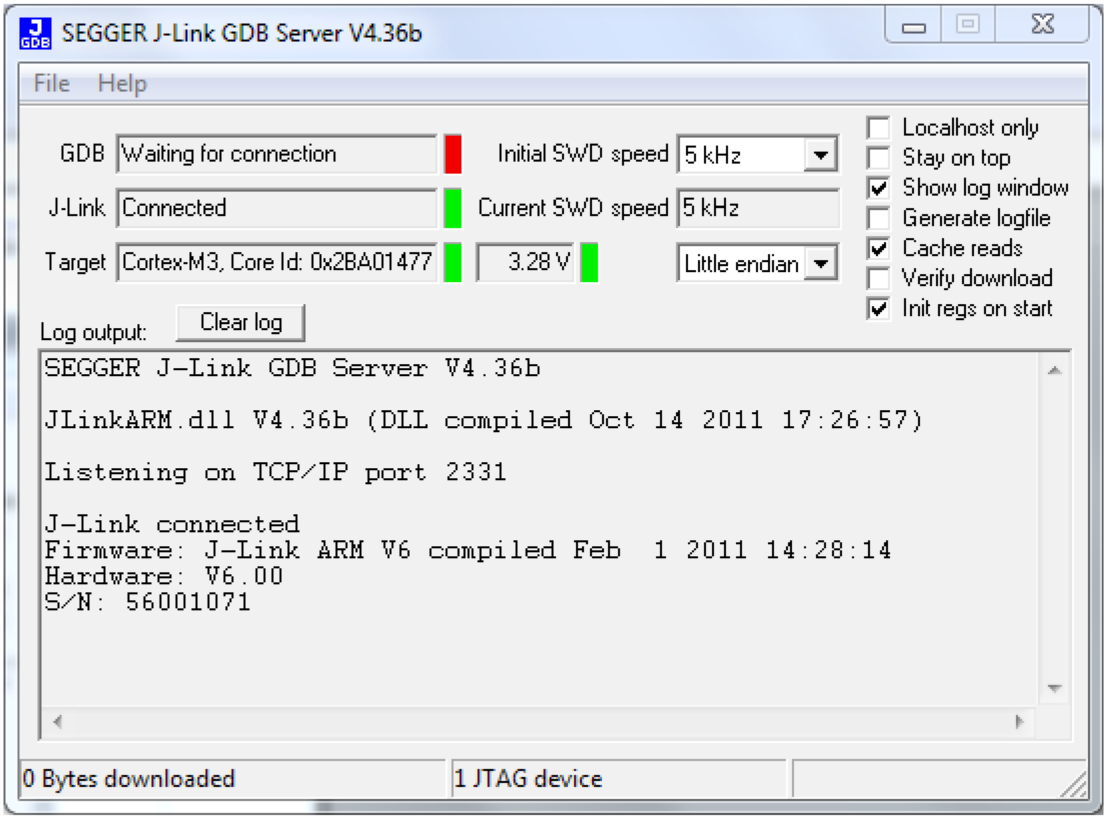
Last, set the “Enable debugger full logging switch”.



☑ Enable debugger full logging

Ln 222, Col 0, Pos 7319

Last, start the Segger GDB server and make sure that is configured to match GDB. The default is to use port 2331. See the Segger J-Link ARM GDB Server PDF file for more information.



GDB: Waiting for connection 🟥
J-Link: Connected 🟩
Target: Cortex-M3, Core Id: 0x2BA01477 🟩
3.28 V 🟩
Initial SWD speed: 5 kHz
Current SWD speed: 5 kHz
Little endian

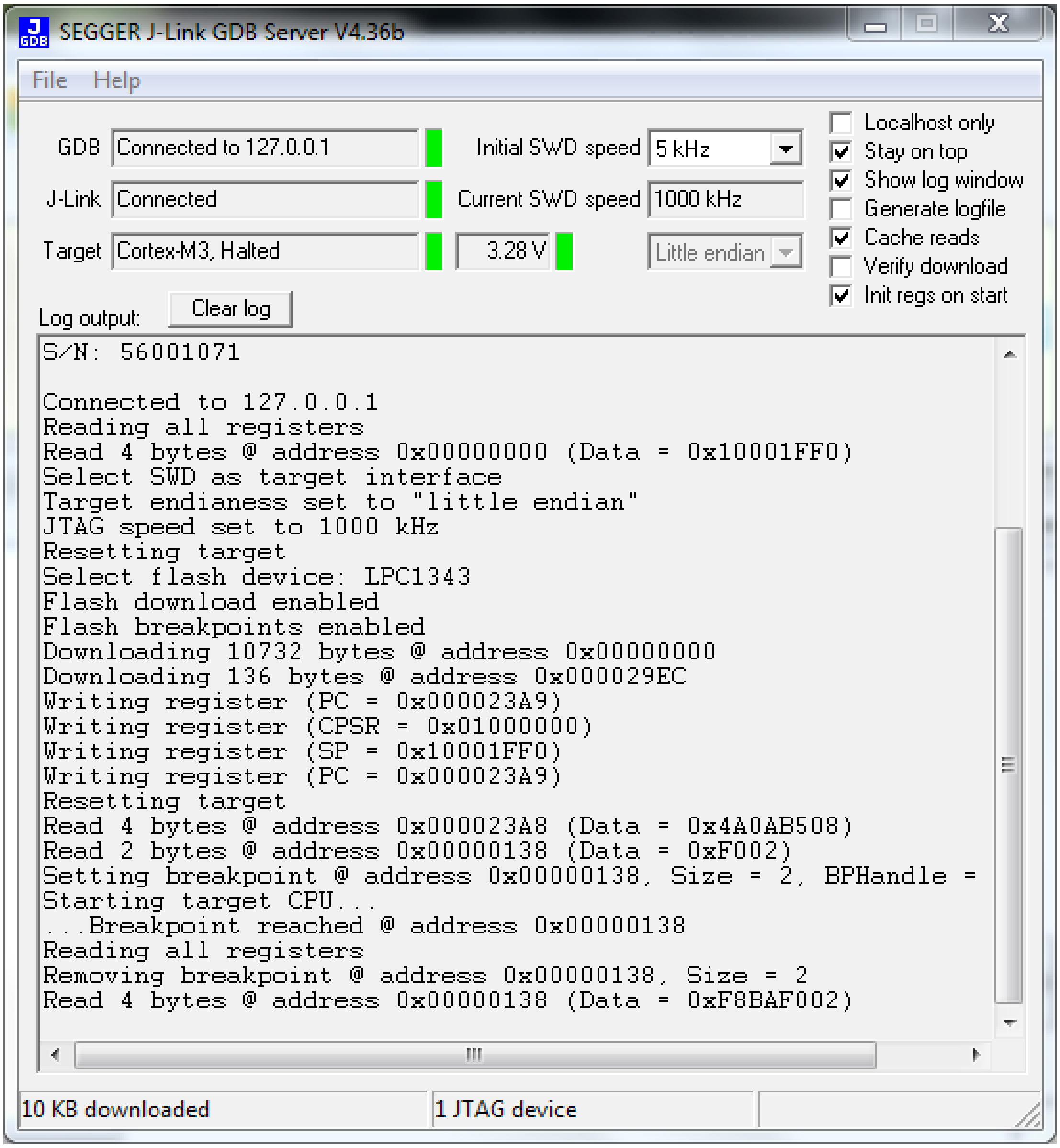
☐ Localhost only
☐ Stay on top
☑ Show log window
☐ Generate logfile
☑ Cache reads
☐ Verify download
☑ Init regs on start

Log output:

```
SEGGER J-Link GDB Server V4.36b

JLinkARM.dll V4.36b (DLL compiled Oct 14 2011 17:26:57)

Listening on TCP/IP port 2331

J-Link connected
Firmware: J-Link ARM V6 compiled Feb  1 2011 14:28:14
Hardware: V6.00
S/N: 56001071
```

0 Bytes downloaded
1 JTAG device

Here is another view of the Segger GDB server in action.



GDB: Connected to 127.0.0.1 🟩
J-Link: Connected 🟩
Target: Cortex-M3, Halted 🟩
3.28 V 🟩
Initial SWD speed: 5 kHz
Current SWD speed: 1000 kHz
Little endian

☐ Localhost only
☑ Stay on top
☑ Show log window
☐ Generate logfile
☑ Cache reads
☐ Verify download
☑ Init regs on start

```
S/N: 56001071

Connected to 127.0.0.1
Reading all registers
Read 4 bytes @ address 0x00000000 (Data = 0x10001FF0)
Select SWD as target interface
Target endianess set to "little endian"
JTAG speed set to 1000 kHz
Resetting target
Select flash device: LPC1343
Flash download enabled
Flash breakpoints enabled
Downloading 10732 bytes @ address 0x00000000
Downloading 136 bytes @ address 0x000029EC
Writing register (PC = 0x000023A9)
Writing register (CPSR = 0x01000000)
Writing register (SP = 0x10001FF0)
Writing register (PC = 0x000023A9)
Resetting target
Read 4 bytes @ address 0x000023A8 (Data = 0x4A0AB508)
Read 2 bytes @ address 0x00000138 (Data = 0xF002)
Setting breakpoint @ address 0x00000138, Size = 2, BPHandle =
Starting target CPU...
...Breakpoint reached @ address 0x00000138
Reading all registers
Removing breakpoint @ address 0x00000138, Size = 2
Read 4 bytes @ address 0x00000138 (Data = 0xF8BAF002)
```

10 KB downloaded
1 JTAG device

Once these changes have been made, set the configuration to debug, rebuild, start the Segger GDB server and start the debugger. If it works properly, you will see the commands run and the code will download, execute to main and break. Press the green arrow again and the target will begin execution again. Press the break/pause button. The debugger will stop with a SIGINT and the current line will be displayed. All the stack frames will be displayed and the local variables will be updated.

If you float the cursor over a function or a variable, then a tool-tip like window will be displayed that has information about the object.

Supported Devices

The way ARM envisioned the Cortex world, it should be easy to change CPUs (e.g. go from a Cortex M0 to a Cortex M3 or vice-versa), change CPU vendors; and at the same time, continue to use the vast majority of the code developed for the initial CPU. With this in mind, I believe that this DLL should be compatible with nearly every Cortex M3 supported by SEGGER and the J-Link. Here is SEGGER webpage that has all the devices supported by the J-Link.

http://www.segger.com/jlink_supported_devices.html

It should be a matter of changing one line in the Project Settings → Debugger → Debugger Startup Commands: Change the "monitor flash device " from the LPC1343, to your target device. If it supported by SEGGER, it should work.

Cortex M3 Debugging Issues

Here is a list of issues that were addressed by the changes I made:

- The first thing that must be done is to put GDB into "async mode". This is to support async calls to run / break. This must be done before a connection is made to the debugger.
- The connection to the debug service was done far too late in the process. It must be the second thing done.
- Loading symbols and the code once initialization was complete.
- Not core but still useful was to remove the PC/Linux/OSX cruft commands to the Segger GDB server.

I modified four functions in DebuggerGDB.cpp:

- *DoLocateGdbExecutable()* -- this was mostly to remove codelite_gdbinit.txt from the gdb execution line. I wanted to re-enable the use of .gdbinit.
- *DoInitializeGdb()* -- this was modified to initialize gdb for embedded development.
- *Run()* -- this was modified to properly start the embedded target. The "target remote" command was removed because it was sent far too late: It reset the connection to the GDB server causing other issues.
- *Break()* -- this was modified to properly break the embedded target.

One caveat: While the updated DLL does support breakpoints in FLASH; the DLL does not (currently) support the Cortex M3 hardware breakpoints. I'm not sure if there is a way to do this in gdb right now.

Command Line Debugging using GDB

Debugging from the command line is surprisingly easy and much more powerful than the GUI. The GUI is nice for most things (setting breakpoints, looking at locals, etc.). However gdb and the command line have an amazing flexibility.

To debug with GDB command line, do the following:

1. Connect up the LPC1343 development board to the Segger J-Link and power everything up.
2. Copy .gdbinit to the same directory as "firmware.elf".
3. Start up the Segger "GDB server"
4. Start up "arm-none-eabi-gdb" from the command line. The .gdbinit file will be called at gdb startup. It will properly initialize gdb and download the firmware.
5. If you aren't familar with the power of gdb, download the book "Debugging with GDB" from the FSF. Almost all the commands in this book will work from the command line.