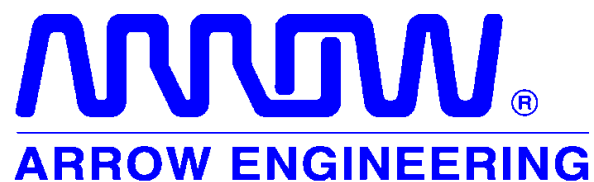


**Bitfire Development Kit
Software Developers Guide
Version 1.2**



This page is intentionally left blank.

Revision History		
Version	Date	Updates
1.2	070514	Updated for the Atmosfire devepment kit
1.1	051021	Minor updates
1.0	050531	Document public release.

Without our written consent in each particular case, this document must not under any circumstances and under penalty of law be reproduced, improperly used, handed over or otherwise communicated to a third part.

Arrow Engineering Sweden, Malmö, Sweden

1 OVERVIEW	8
2 GETTING STARTED.....	8
2.1 INSTALLING AND SETTING UP THE ENVIRONMENT UNDER WINDOWS®.....	8
2.1.1 <i>Cygwin</i>	8
2.1.2 <i>GNU GCC Compiler</i>	10
2.1.3 <i>Eclipse</i>	11
2.1.4 <i>JRE</i>	11
2.1.5 <i>openocd</i>	11
2.1.6 <i>Philips Flash Utility</i>	12
2.1.7 <i>Bitfire Source Library</i>	12
2.2 INSTALLING AND SETTING UP THE ENVIRONMENT UNDER LINUX.....	12
2.2.1 <i>GNU Compiler</i>	12
2.2.2 <i>Eclipse</i>	13
2.2.3 <i>openocd</i>	13
2.2.4 <i>Bitfire Source Library</i>	13
3 BUILDING AN EXAMPLE PROGRAM	14
3.1 BUILDING WITH MAKEFIES	14
3.2 BUILDING WITH ECLIPSE.....	15
4 RUNNING AN EXAMPLE FROM RAM.....	18
4.1 USING GDB	18
5 DEBUGGING AN EXAMPLE FROM RAM	19
5.1 USING INSIGHT	19
5.2 USING ECLIPSE	22
6 RUNNING AN EXAMPLE FROM FLASH.....	25
7 DEBUGGING FROM FLASH.....	26
7.1 USING GDB	26
8 THE BITFIRE BSP.....	28
8.1 BUILDING THE LIBBITFIREBSP.A.....	28
9 THE BITFIRE GRAPHICS LIBRARY	28
9.1 BUILDING THE LIBGFX.A	28
10 THE EXAMPLES	28
10.1 HOW THE EXAMPLE PROGRAMS WORK.....	30
11 PORTED OPERATING SYSTEMS.....	31
11.1 FREERTOS™	31
11.2 UC/OS-II	31
12 APPENDIX A: SETTING UP THE BITFIRE BOARD FOR RUNNING AND DEBUGGING.....	32
13 APPENDIX B: SETTING UP THE BITFIRE BOARD FOR PROGRAMMING THE CPU FLASH.....	33
14 APPENDIX C: BSP REFERENCE	34
14.1 INCLUDE FILES	34
14.1.1 <i>bitfire.h</i>	34
14.1.2 <i>bitfire_fpga.h</i>	34

14.1.3 <i>lpc2129_can.h</i>	34
14.1.4 <i>lpc2129_can_polled.h</i>	34
14.1.5 <i>lpc2129_spi.h</i>	34
14.1.6 <i>lpc2129_spi_polled.h</i>	34
14.1.7 <i>lpc2129_uart.h</i>	34
14.1.8 <i>lpc2129_uart_int.h</i>	34
14.1.9 <i>lpc2129_uart_polled.h</i>	34
14.1.10 <i>lpc2129_vic.h</i>	34
14.1.11 <i>lpc21xx.h</i>	34
14.2 FUNCTION DESCRIPTIONS	35
14.2.1 <i>bitfire_fpga</i>	35
14.2.1.1 void <i>fpga_command</i> (void).....	35
14.2.1.2 void <i>fpga_data</i> (void).....	35
14.2.1.3 void <i>fpga_init</i> (void).....	35
14.2.2 <i>lpc2129_can_polled</i>	35
14.2.2.1 void <i>can_polled_clear_rxbuf</i> (can_control * cc)	35
Parameters:	35
14.2.2.2 void <i>can_polled_init</i> (can_control * cc).....	35
Parameters:.....	35
14.2.2.3 UNS_8 <i>can_polled_send</i> (can_control * cc, UNS_8 dlc, UNS_8 * data)	35
Parameters:	35
14.2.2.4 void <i>can_polled_set_filters</i> (void * filters)	35
Parameters:	36
14.2.2.5 UNS_8 <i>can_polled_receive</i> (can_control * cc, UNS_8 * data).....	36
Parameters:.....	36
14.2.3 <i>lpc2129_spi_polled</i>	36
14.2.3.1 UNS_8 <i>spi_polled_getchar</i> (spi_control * sc)	36
Parameters:	36
14.2.3.2 void <i>spi_polled_init</i> (spi_control * sc).....	36
Parameters:	36
14.2.3.3 UNS_8 <i>spi_polled_putchar</i> (spi_control * sc, UNS_8 ch)	36
Parameters:.....	36
14.2.4 <i>lpc2129_uart_int</i>	36
14.2.4.1 UNS_8 <i>uart_int_getchar</i> (uart_control * uc, UNS_8 * err)	36
Parameters:	36
Returns:	37
14.2.4.2 void <i>uart_int_init</i> (uart_control * uc).....	37
Parameters:	37
14.2.4.3 UNS_8 <i>uart_int_putchar</i> (uart_control * uc, UNS_8 ch).....	37
Parameters:	37
14.2.5 <i>lpc2129_uart_polled</i>	37
14.2.5.1 UNS_8 <i>uart_polled_getchar</i> (uart_control * uc)	37
Parameters:	37
Returns:	37
14.2.5.2 void <i>uart_polled_init</i> (uart_control * uc)	37
Parameters:	37
14.2.5.3 UNS_8 <i>uart_polled_putchar</i> (uart_control * uc, UNS_8 ch).....	38
Parameters:	38
14.2.6 <i>lpc2129_vic</i>	38
14.2.6.1 void <i>vic_disable_int</i> (vic_control * vc).....	38
Parameters:	38
14.2.6.2 void <i>vic_enable_int</i> (vic_control * vc).....	38
Parameters:	38
14.2.6.3 void <i>vic_install_isr</i> (vic_control * vc).....	38
Parameters:	38
14.2.6.4 void <i>vic_remove_isr</i> (vic_control * vc)	38
Parameters:	38
15 APPENDIX D: GRAPHICS LIB REFERENCE	39

15.1 WIN32 EMULATION.....	39
15.2 INCLUDE FILES	39
15.2.1 <i>gfxfx.h</i>	39
15.2.2 <i>gfxlib.h</i>	39
15.2.3 <i>gfmtxt.h</i>	39
15.2.4 <i>gfxvec.h</i>	39
15.3 FUNCTION DESCRIPTIONS	39
15.3.1 <i>gfxfx</i>	39
15.3.1.1 void <i>glfx_3dstar</i> (<i>glfx_star</i> * stars, UNS_8 starnum).....	39
Parameters:	39
15.3.1.2 BOOL_8 <i>glfx_charsinscroll</i> (CHAR * text, gl_col * col, const INT_8 * sintable, UNS_32 * offset).....	39
Parameters:	39
Returns:	40
15.3.1.3 void <i>glfx_fire</i> (UNS_8 * pbuf, UNS_8 YSIZE)	40
Parameters:	40
15.3.1.4 void <i>glfx_fire2</i> (UNS_8 * pbuf, UNS_8 YSIZE)	40
Parameters:	40
15.3.1.5 void <i>glfx_init_greenfire_palette</i> (gl_col * palette)	40
Parameters:	40
15.3.1.6 void <i>glfx_init_radient_palette</i> (gl_col * palette)	40
Parameters:	40
15.3.1.7 void <i>glfx_init_redfire_palette</i> (gl_col * palette)	40
Parameters:	40
15.3.1.8 void <i>glfx_init_yellowfire_palette</i> (gl_col * palette)	40
Parameters:	41
15.3.1.9 void <i>glfx_julia</i> (INT_32 maxiter, UNS_8 co).....	41
Parameters:	41
15.3.1.10 void <i>glfx_mandel</i> (INT_32 maxiter, UNS_8 co).....	41
Parameters:	41
15.3.1.11 void <i>glfx_pattern1</i> (UNS_32 * offset).....	41
Parameters:	41
15.3.1.12 void <i>glfx_pattern2</i> (UNS_32 * offset).....	41
Parameters:	41
15.3.1.13 void <i>glfx_pattern3</i> (UNS_32 * offset, const INT_8 * sintable, UNS_16 sinlen).....	41
Parameters:	41
15.3.1.14 void <i>glfx_pattern4</i> (UNS_32 * offset, const INT_8 * sintable, UNS_16 sinlen).....	41
Parameters:	42
15.3.1.15 void <i>glfx_plasma</i> (UNS_8 * pbuf, const INT_8 * costable, <i>glfx_plasmavars</i> * pvs)	42
Parameters:	42
15.3.1.16 BOOL_8 <i>glfx_scrolltext</i> (CHAR * text, gl_point * pos, gl_col * col, UNS_32 * offset).....	42
Parameters:	42
Returns:	42
15.3.2 <i>gfxlib</i>	42
15.3.2.1 void <i>gl_circle</i> (gl_point * p, UNS_8 r, gl_col * c)	42
Parameters:	42
15.3.2.2 void <i>gl_cliprect</i> (gl_rect * rect).....	42
Parameters:	42
15.3.2.3 void <i>gl_clrscr</i> (gl_col * col)	43
Parameters:	43
15.3.2.4 void <i>gl_ellipse</i> (gl_point * p, UNS_8 a, UNS_8 b, gl_col * c)	43
Parameters:	43
15.3.2.5 void <i>gl_fillrect</i> (gl_rect * rect, gl_col * col)	43
Parameters:	43
15.3.2.6 void <i>gl_flipbuf</i> (void)	43
15.3.2.7 void <i>gl_init</i> ()	43
15.3.2.8 BOOL_8 <i>gl_iswithin</i> (gl_point * p, gl_rect * r)	43
Parameters:	43
15.3.2.9 void <i>gl_line</i> (gl_point * p1, gl_point * p2, gl_col * c).....	44
Parameters:	44

15.3.2.10 void gl_pblit (UNS_8 * source, gl_rect * srect, gl_point * dpos, gl_col * palette)	44
Parameters:	44
15.3.2.11 void gl_printhexdot (INT_32 val, gl_point * p, gl_col * c).....	44
15.3.2.12 UNS_32 gl_random ()	44
15.3.2.13 void gl_setpixel (gl_point * pos, gl_col * col)	44
Parameters:	44
15.3.2.14 void gl_setpixelxy (UNS_16 x, UNS_16 y, gl_col * col).....	44
Parameters:	44
15.3.3 <i>gfxtxt</i>	45
15.3.3.1 void gltxt_printhex (UNS_32 hexval, gl_col * col, BOOL_8 shortval)	45
Parameters:	45
15.3.3.2 void gltxt_printtext (CHAR * str, gl_point * pos, gl_col * col).....	45
Parameters:	45
15.3.3.3 void gltxt_renderchar (CHAR ch, gl_point * pos, gl_col * col).....	45
Parameters:	45
15.3.3.4 void gltxt_setfont (const UNS_16 * font)	45
Parameters:	45
15.3.4 <i>gfxvec</i>	45
15.3.4.1 void InitGVectors (const INT_16 * data, BOOL_8 setuptables)	45
Parameters:	46
15.3.4.2 void UpdateGVectors (INT_16 distance, gl_col * c)	46
Parameters:	46

1 Overview

Bitfire provides the advanced embedded systems developer with an excellent low-cost ARM7 based development kit. The kit also sports a low-cost FPGA, which opens possibilities for a wide variety of applications. This guide explains how to install and setup the required tools for ARM development. Further it explains how to compile the examples, and how to run and debug them from RAM and Flash. Finally you will find a complete reference of the BSP, Graphics Library and example sources.

2 Getting Started

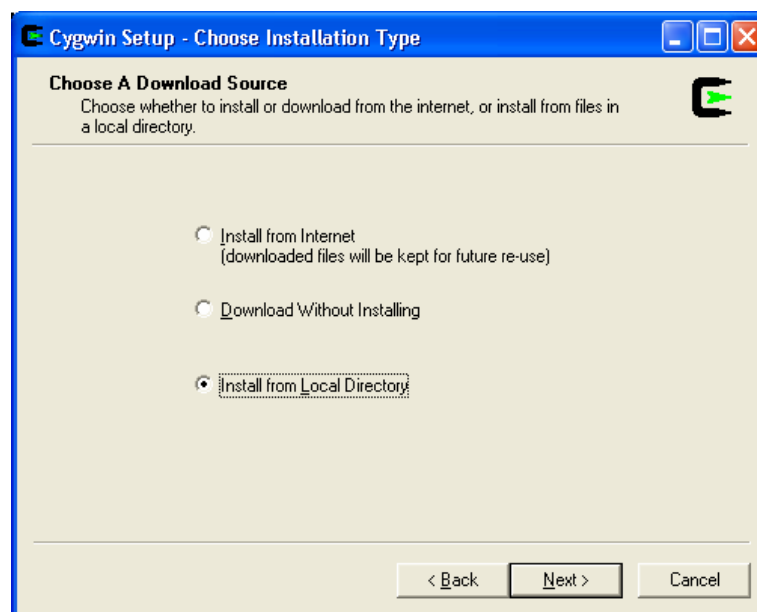
The Bitfire development kit ships with a complete set of development and debugging tools. No additional software or hardware is required. It's all based around the GNU GCC/GDB tool suite, thus it runs all platforms are supported by GCC. The Bitfire build environment has been tested under Windows[®] and Linux operating systems, but should work on all platforms supported by GCC.

2.1 Installing and setting up the environment under Windows[®]

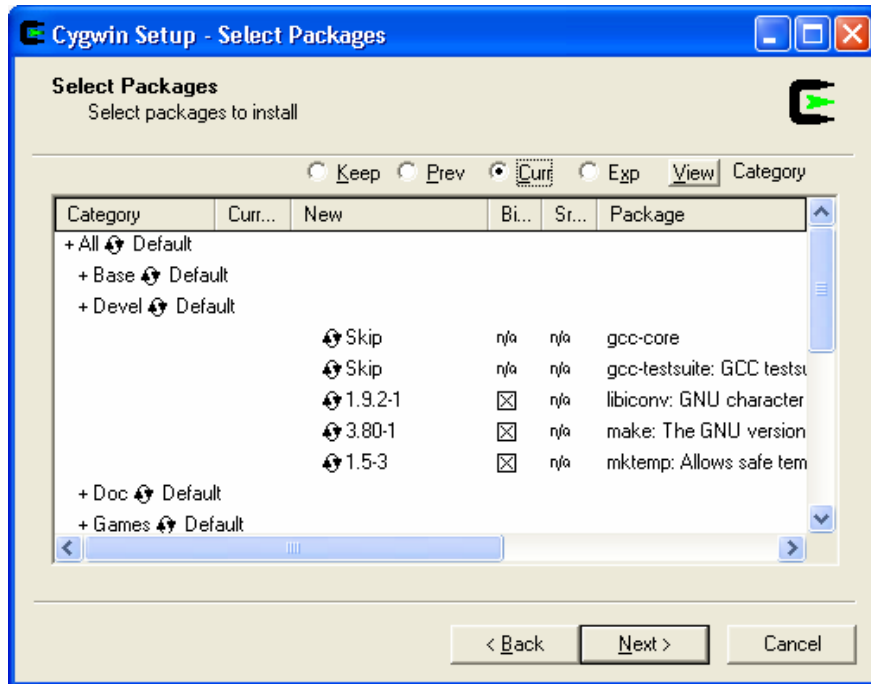
2.1.1 Cygwin

Cygwin is a Linux-like environment for Windows[®]. The GNU GCC compiler and openocd requires the Cygwin DLLs. On the Bitfire CD you will find a pre-downloaded version of the Cygwin applications needed.

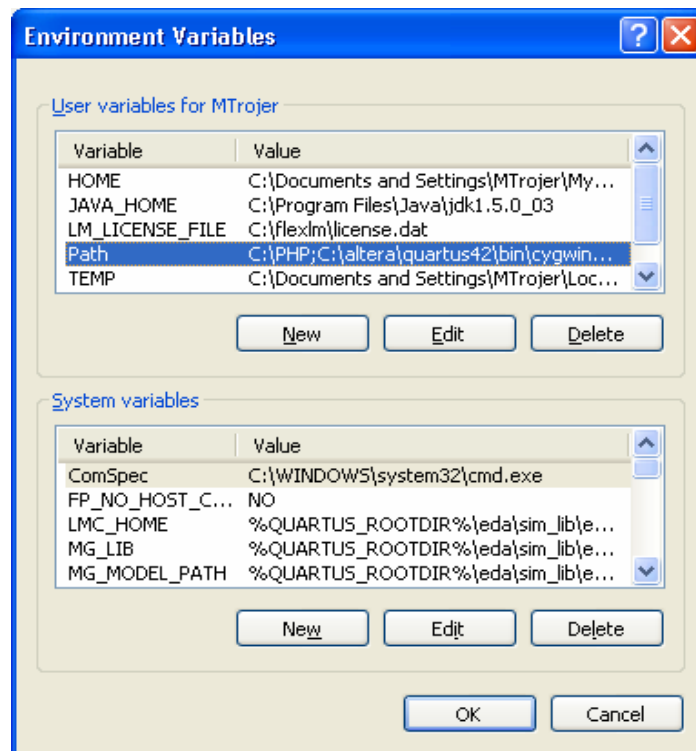
To install Cygwin you run the **setup.exe** in the Bitfire CDs **tools/cygwin** directory. Press next and choose **Install from Local Directory** as show below:



Press Next and choose in which root directory to install, `C:\Bitfire\cygwin` is recommended. On the **Select Local Package Directory** page select the `tools\cygwin` directory on the Bitfire CD. On the **Select Packages** page you should keep all the default selected items, but you need to install the **make** utility. This can be found under the 'Devel' folder. Press next and wait for the download and install to complete.



After the install is complete you have to add the directory `C:\Bitfire\cygwin\bin` to Windows® Path. This is done by right-clicking on **My Computer**, and selecting properties. In the **System Properties** window, go to the Advanced Tab and click **Environment Variables**.



Highlight the **Path** variable in User Variables list (press New if no Path variable exist). Press Edit and enter **C:\Bitfire\Cygwin\bin** in the variable value textbox. If there already are values present in this textbox, put a semicolon ';' as spacing between the variables.

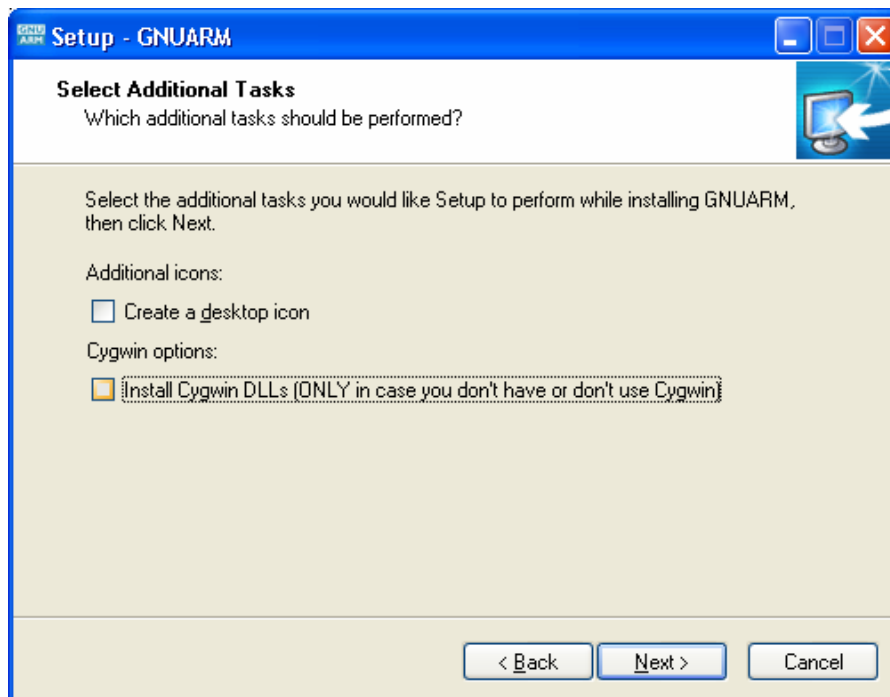
2.1.2 GNU GCC Compiler

Bitfire development kit ships with GCC arm cross compiler (arm-elf-gcc) for Cygwin environment running under Windows®. Bitfire uses the GNUARM (www.gnuarm.org) distribution. The current GNUARM distribution on the Bitfire CD consists of:

- binutils (2.17) assembler and linker
- gcc (4.1.1) C/C++ cross compiler
- newlib (1.14) c and m runtime libs
- insight (6.5) GDB and Insight (GUI based) debugger

To install GNUARM simply run the **bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.exe** in the Bitfire CDs **tools** directory. The suggested install directory is **C:\Bitfire\GNUARM**

When you arrive at the following page you have to make sure the **Install Cygwin DLLs** tickbox is unchecked.



Also tick the last checkbox which asks you update your **PATH** environment variable to include the GNUARM binaries.

2.1.3 Eclipse

Eclipse is an open extensible IDE. It is used for project management, build control and debugging. It sits on-top of the GCC/GDB tools and uses Makefiles for building.

To install eclipse unzip the file **eclipse-SDK-3.1.1-win32_org.eclipse.cdt-3.0.0-win32.x86.zip** located in the Bitfire CDs **tools** directory. The recommended install directory is **C:\Bitfire\eclipse**

2.1.4 JRE

Eclipse requires Java to run. If you don't have a JRE (Java Runtime Environment) on your PC you will need to install one. To install a JRE run the file **jre-6u1-windows-i586-p.exe** located in the Bitfire CDs **tools** directory.

2.1.5 openocd

In order to connect GDB to the Bitfire on-board wiggler you will need this application. openocd works by opening a TCP port which accepts GDB connections; it then translates GDB commands to signals (via the PC's parallel port) on the Bitfire CPU's JTAG pins. Since openocd uses TCP ports, you can connect with GDB either locally on the same computer or over the network.

To install openocd simply run the file **openocd-2006re115-setup-rc01.exe** located in Bitfire CDs **tools** directory. In the installation wizard, untick “Make utils” since we already have them in Cygwin. The recommended install directory is
C:\Bitfire\openocd

Please note that in openocd requires the Parallel port of your PC to be in EPP (or ECP) mode. This is usually controlled in the BIOS setup.

2.1.6 Philips Flash Utility

In order to program the flash on the Bitfire CPU (LPC2129) device you will need to run this program.

To install this utility unzip the file **lpc2000_flash_utility.zip** located in the Bitfire CDs **tools** directory. Then run the unzipped **setup.exe** file and follow the instructions.

2.1.7 Bitfire Source Library

The Bitfire source library contains C and assembler BSP drivers, a C graphics library and examples described later in this guide.

To install the Bitfire Source Library unzip the **bitfire_atmosfire_sw_src.zip** file located in the Bitfire CDs **source** directory. The suggested install location is
C:\Bitfire\software

2.2 Installing and setting up the environment under Linux

When using a Linux based PC for development you will not be able to run the Philips Flash Utility. However, a working flasher has been supplied in the Bitfire CDs **source** directory. It's located in the **bitfire_atmosfire_sw_src.zip** file and called **lpc21isp.c**. Just compile and run it.

2.2.1 GNU Compiler

See 2.1.2 for details, the same applies to the Linux version; however the version of the tools can change.

In the Bitfire CDs **tools** directory you will find the following tarball: **bu-2.16.1_gcc-4.0.1-c-c++-java_nl-1.13.0_gi-6.1_x86-64.tar.bz2**

Un-tar and install at your desired location.

2.2.2 Eclipse

There are several versions of Eclipse available for Linux. Please go to www.eclipse.org and download the version that best suits your system. You will also need the CDT plug-in for C/C++ support.

Don't forget that Eclipse requires an JRE on your system.

2.2.3 openocd

See 2.1.3 for details, the same applies to the Linux version.

Openocd installs with source, check the readme and compile.

2.2.4 Bitfire Source Library

See 2.1.7 for details.

The sources are located in the Bitfire CDs **source** directory, the file is called **bitfire_atmosfire_sw_src.zip**

Unzip and install at your desired location.

3 Building an example program

After you have installed the applications and source library described above you can start building the example programs. In this guide we assume you want to build the **gfxlib_setpixel** example. For more information on other examples, please see section 10.

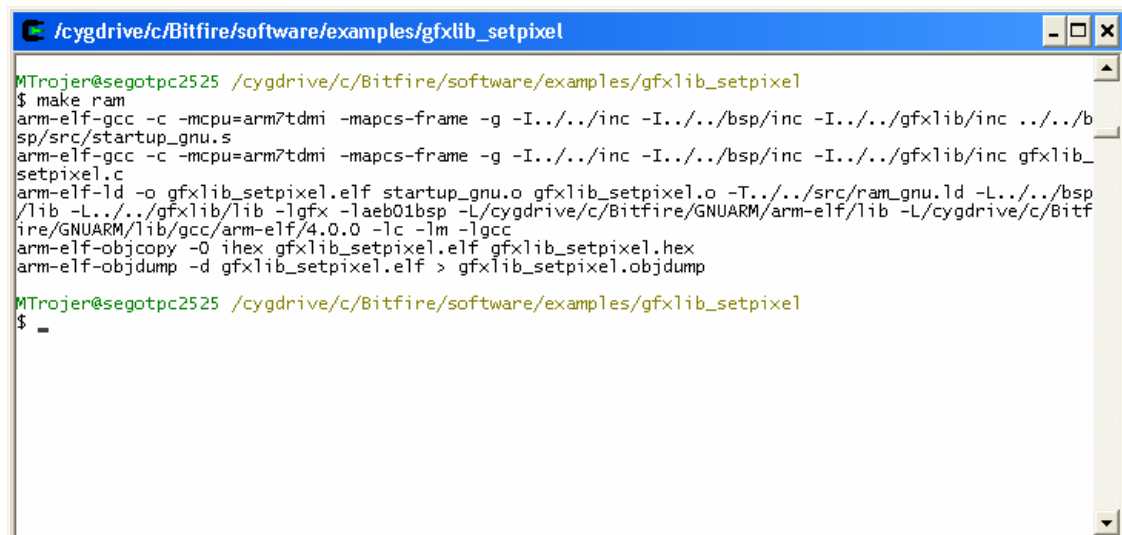
The Makefiles depend on some environment variables, these can be set with a shell script located in the software directory. Please do this before compiling any examples.

```
$ cd software
$ . setenv_bash.sh
```

3.1 Building with Makefiles

Start a Cygwin shell and use the change directory command **cd** to enter the **software/examples/gfxlib_setpixel** directory.

Run the **make bitfire-ram** command. You should get an output like the one below:



```
MTrojer@segotpc2525 /cygdrive/c/Bitfire/software/examples/gfxlib_setpixel
$ make ram
arm-elf-gcc -c -mcpu=arm7tdmi -mapcs-frame -g -I../inc -I../bsp/inc -I../gfxlib/inc ../b
sp/src/startup_gnu.s
arm-elf-gcc -c -mcpu=arm7tdmi -mapcs-frame -g -I../inc -I../bsp/inc -I../gfxlib/inc gfxlib_
setpixel.c
arm-elf-ld -o gfxlib_setpixel.elf startup_gnu.o gfxlib_setpixel.o -T../src/ram_gnu.ld -L../bsp
/lib -L../gfxlib/lib -lgfx -laeb01bsp -L/cygdrive/c/Bitfire/GNUARM/arm-elf/lib -L/cygdrive/c/Bitf
ire/GNUARM/lib/gcc/arm-elf/4.0.0 -lc -lm -lgcc
arm-elf-objcopy -O ihex gfxlib_setpixel.elf gfxlib_setpixel.hex
arm-elf-objdump -d gfxlib_setpixel.elf > gfxlib_setpixel.objdump
MTrojer@segotpc2525 /cygdrive/c/Bitfire/software/examples/gfxlib_setpixel
$
```

You have now successfully built your first Bitfire program! The Makefile compiled the **startup_gnu.s** and **gfxlib_setpixel.c** source files and linked them with the *bitfirebsp* and *gfx* library. It created the **gfxlib_bitfire_setpixel.elf** output (the executable program); it also created a hexfile (**gfxlib_bitfire_setpixel.hex**) from the elf and a disassembly **gfxlib_bitfire_setpixel.objdump** file.

See the section 10.1 for more details on how the example works.

3.2 Building with Eclipse

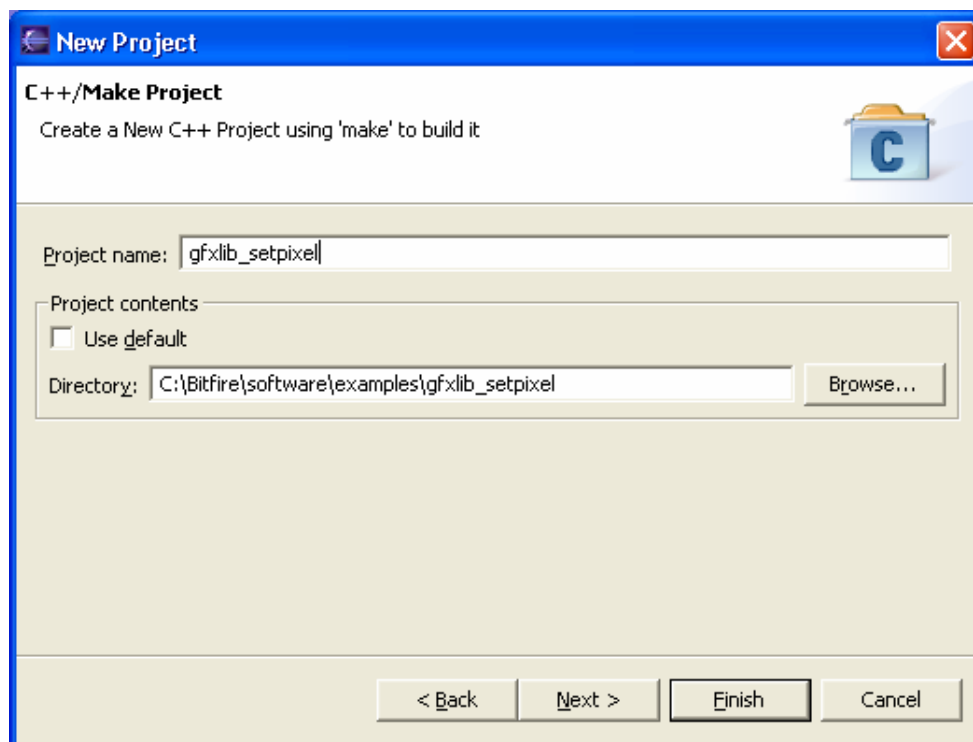
The Makefiles of the examples depends on 2 environment variables called `TOOLDIR` and `GCC_VER`. They are set in the `setenv_bash.sh` file when you compile by hand. In order for eclipse to see them you have to apply them for your system. It's the same method as when you added the cygwin bin directory to your path. See above and add the `TOOLDIR` and `GCC_VER` to your environment variables. The values you want are probably:

- `TOOLDIR` `C:\Bitfire\GNUARM`
- `GCC_VER` `4.1.1`

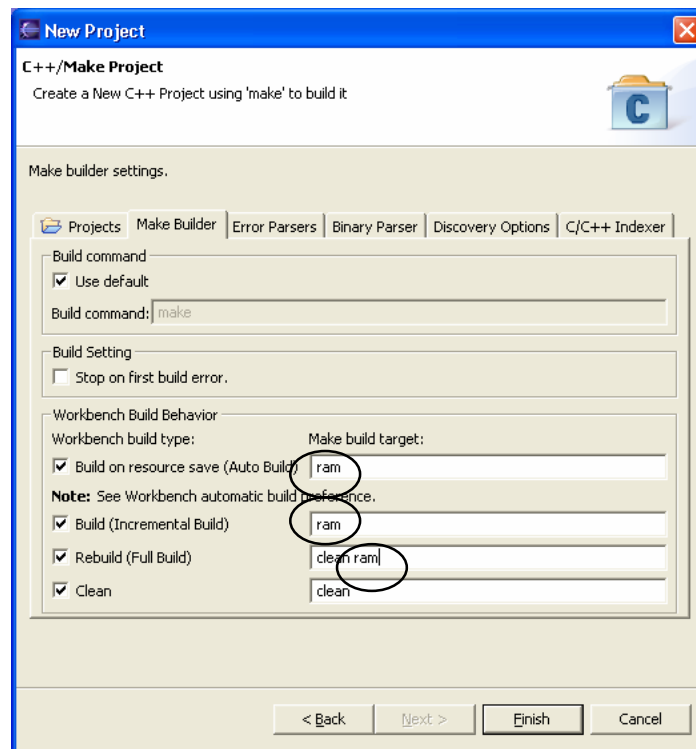
Run the file `C:\Bitfire\eclipse\eclipse.exe` to start Eclipse (it's a good idea to make a shortcut to this file and place it in the start-menu or desktop).

Eclipse is a full IDE with project support, so you will need to create a project before compiling your projects. Use the **Menu File->New->Project** and chose **C/Standard Make C project** in the list.

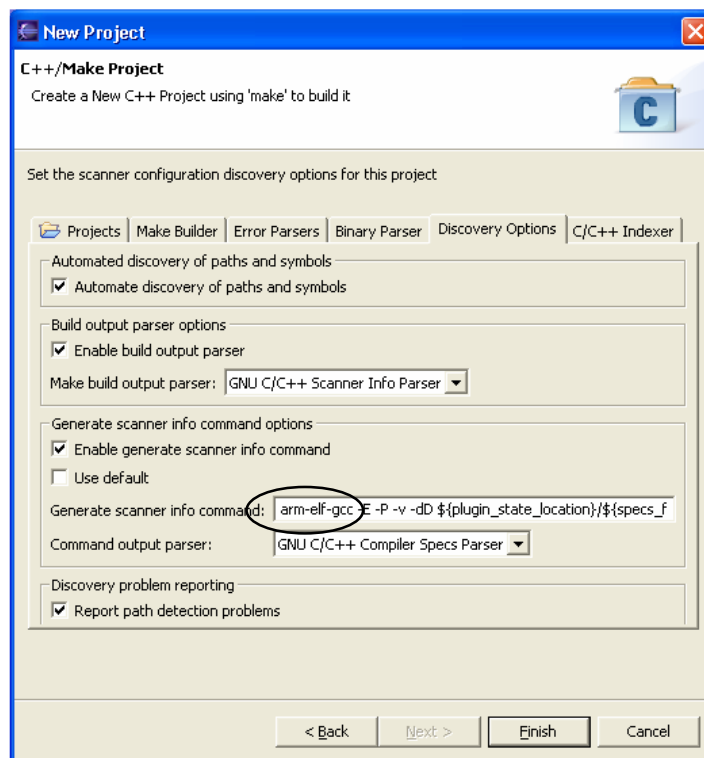
In the next page setup the window like below and press next.



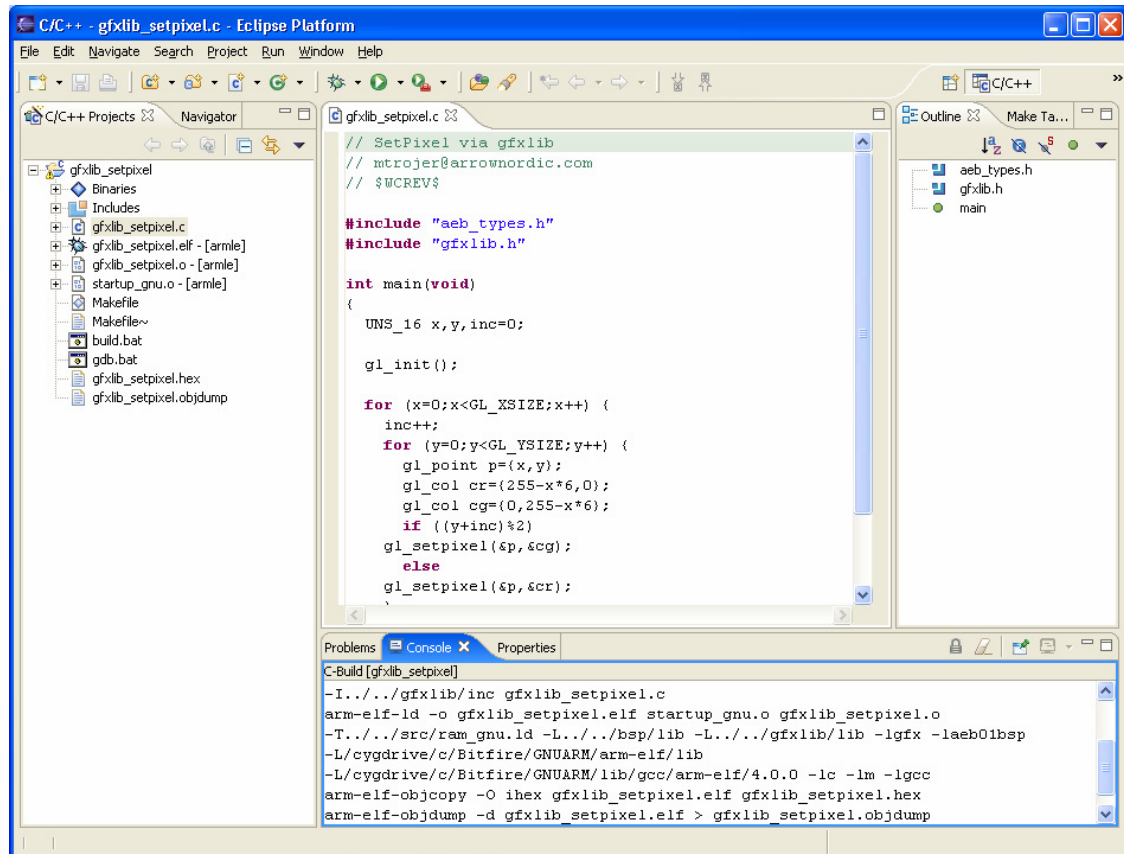
On the following page you need to make some changes. Firstly, select the **Make Builder** tab and change **all** to **bitfire-ram** like below:



Go to the Discovery Options tab and change gcc to arm-elf-gcc like this:



Press Finish and you should get a GUI like this:



As you can see in the Console window at the bottom the example get automatically built.

See the section 10.1 for more details on how the example works.

4 Running an example from RAM


4.1 Using GDB

1. Make sure the Bitfire board is setup correctly, see Appendix A: Setting up the Bitfire board for running and debugging.
2. Open a Cygwin shell and use `cd` to enter the `software/examples/gfxlib_setpixel` directory.
3. Make sure the example is correctly built for ram, see section 3.
4. Run the `arm-elf-gdb gfxlib_bitfire_setpixel.elf` command. Now you should have entered GDB and have a (gdb) prompt.
5. Type `target remote 127.0.0.1:8888`
This will force GDB to make a TCP connection with the openocd tool.

Note: Always use the IP-address '127.0.0.1' when running the openocd and GDB on the same computer.

6. Type `load`
This will force GDB to load the executable into the CPU's onchip RAM.

At this point you should have an output like this:



```
/cygdrive/c/Bitfire/software/examples/gfxlib_setpixel
MTrojer@segotpc2525 /cygdrive/c/Bitfire/software/examples/gfxlib_setpixel
$ arm-elf-gdb gfxlib_setpixel.elf
GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=arm-elf"...
(gdb) target remote 192.168.123.161:8888
Remote debugging using 192.168.123.161:8888
0x00008080 in ?? ()
(gdb) load
Loading section .text, size 0x8b8 lma 0x40000000
Loading section .rodata, size 0x14 lma 0x400008b8
Loading section .rodata.str1.4, size 0x2 lma 0x400008cc
Loading section .data, size 0x40c lma 0x400008d0
Start address 0x40000000, load size 3290
Transfer rate: 3290 bits/sec, 235 bytes/write.
(gdb) c
Continuing.
```

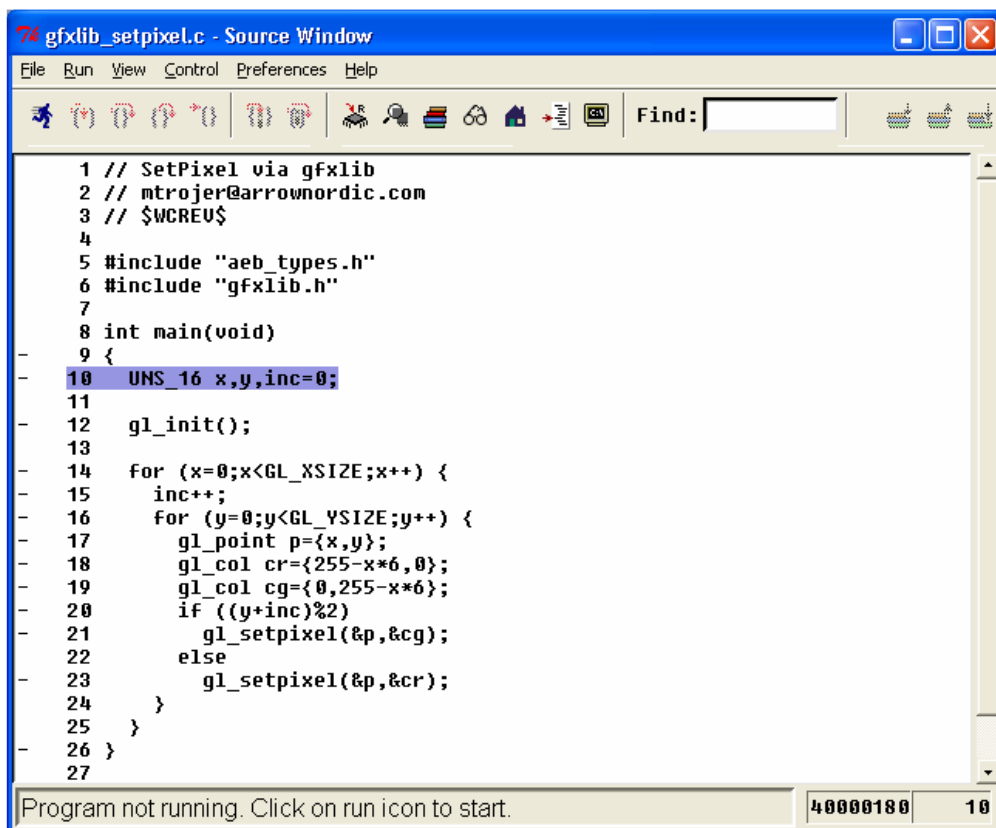
7. Finally, type `c` to continue (run) the program.
At this point the Bitfire's LED matrix should light up and display a nice test pattern.

5 Debugging an example from RAM

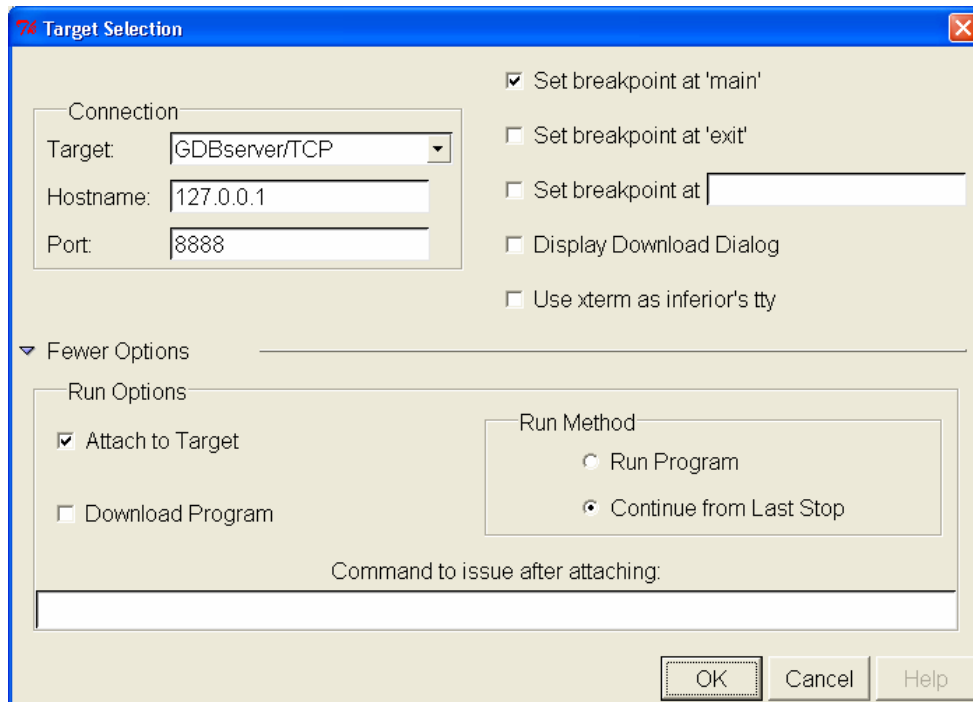
5.1 Using Insight

Insight is a GUI built on-top of GDB to make it easier to step in the code and to manipulate registers, memory and so on. It works in very much the same way as GDB, but it has a nice GUI.

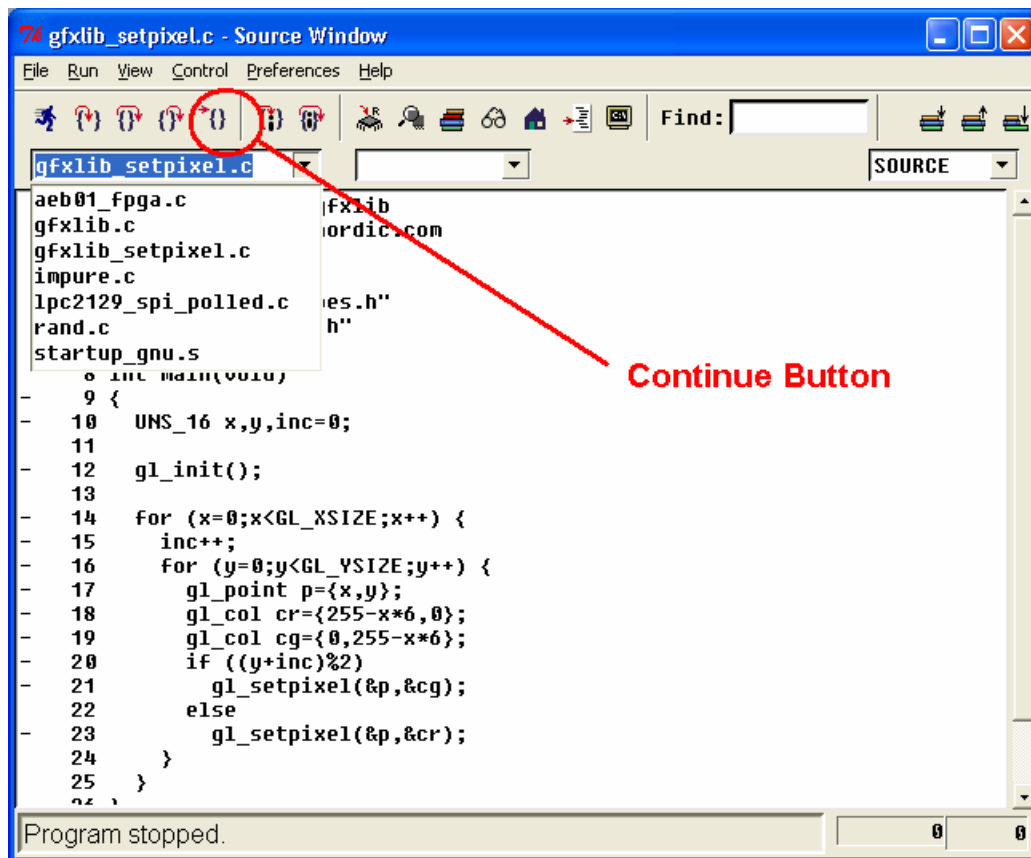
1. Follow steps 1 through 3 in section 4.1.
2. Run the `arm-elf-insight gfxlib_bitfire_setpixel.elf` command. Now you should see the following GUI:



3. Select the **File->Target Settings** menu. Make the following choices:



4. Press OK and go to the **Run->Connect** to target menu. You should get a “Successfully connected” window appearing.
5. Select the **Run->Download** menu and wait for it to finish.
6. Select the **gfxlib_setpixel.c** from the sourcefile list like this:



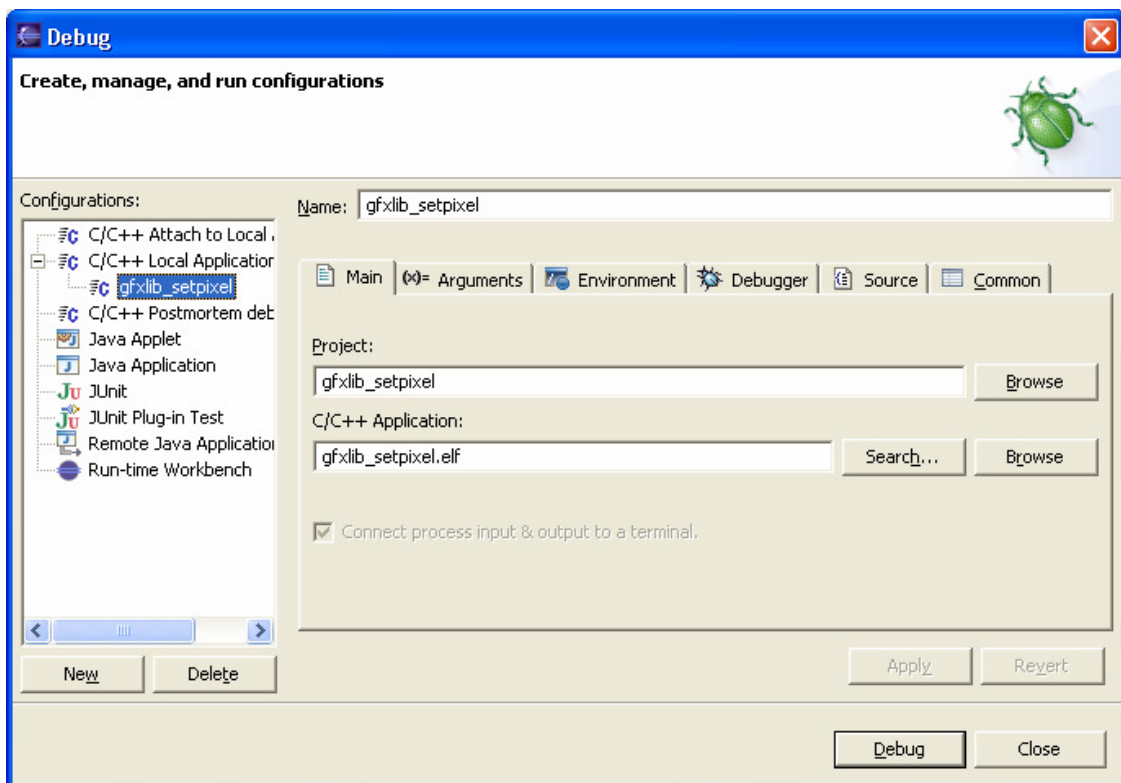
7. Click on the leftmost dash on a specific line to add a breakpoint. In this case do it on line 9. The dash should change to a red square to indicate the breakpoint.
8. Press the Continue button. You should now have stopped at the breakpoint which you have just set.
9. Explore the **View** menu options to look at local variables, memory, stack and more. There are also various step buttons next to the continue button. For more information please read the Insight documentation at <http://sources.redhat.com/insight/>

5.2 Using Eclipse

1. Make sure the Bitfire board is setup correctly, see Appendix A: Setting up the Bitfire board for running and debugging.
2. Open Eclipse with a setup project like described in section 3.2.
3. Edit the file `.gdbinit` located in your project directory, it should contain minimum 2 lines:
 - target remote 127.0.0.1:8888 (if you are running from an local machine, another IP if you are running openocd on a computer on the network).
 - load

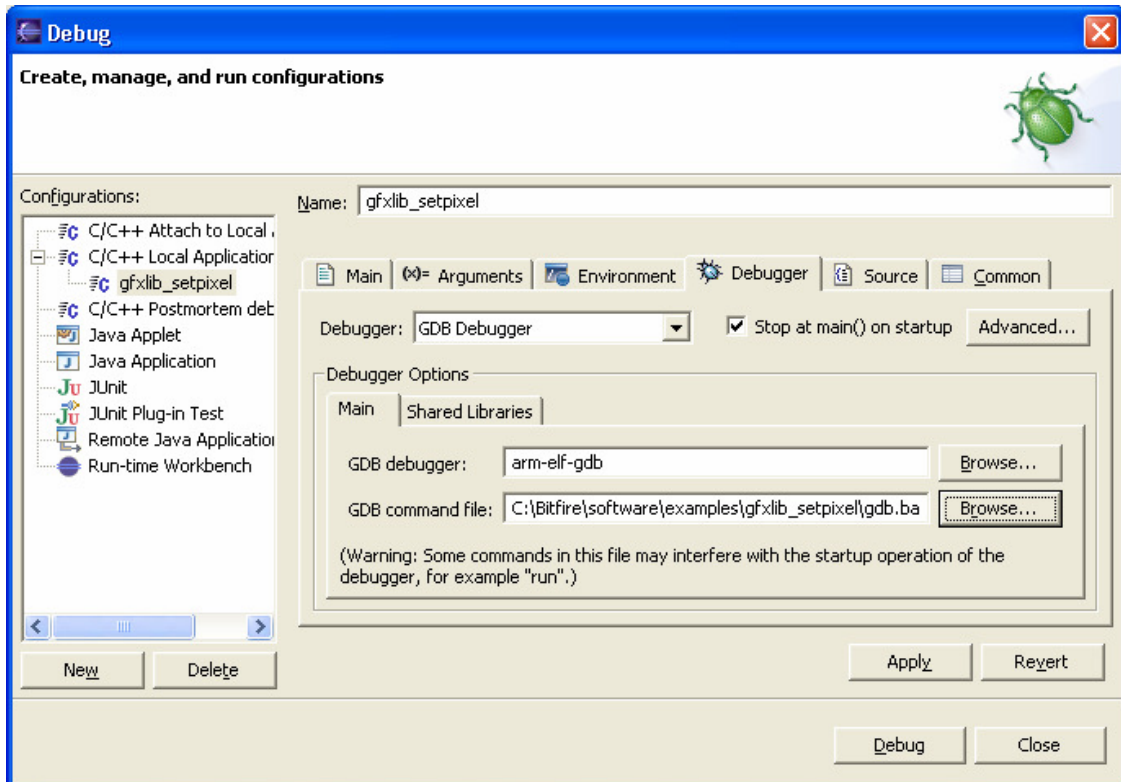
The file `.gdbinit` contains gdb command to be sent to gdb before Eclipse takes over.

4. Go to the menu **Run->Debug...**
5. Make sure **C/C++ Local Application** is selected in the list to the left and press New button.
6. Press Browse next to the C/C++ Application textbox and select the `gfxlib_bitfire_setpixel.elf` file.

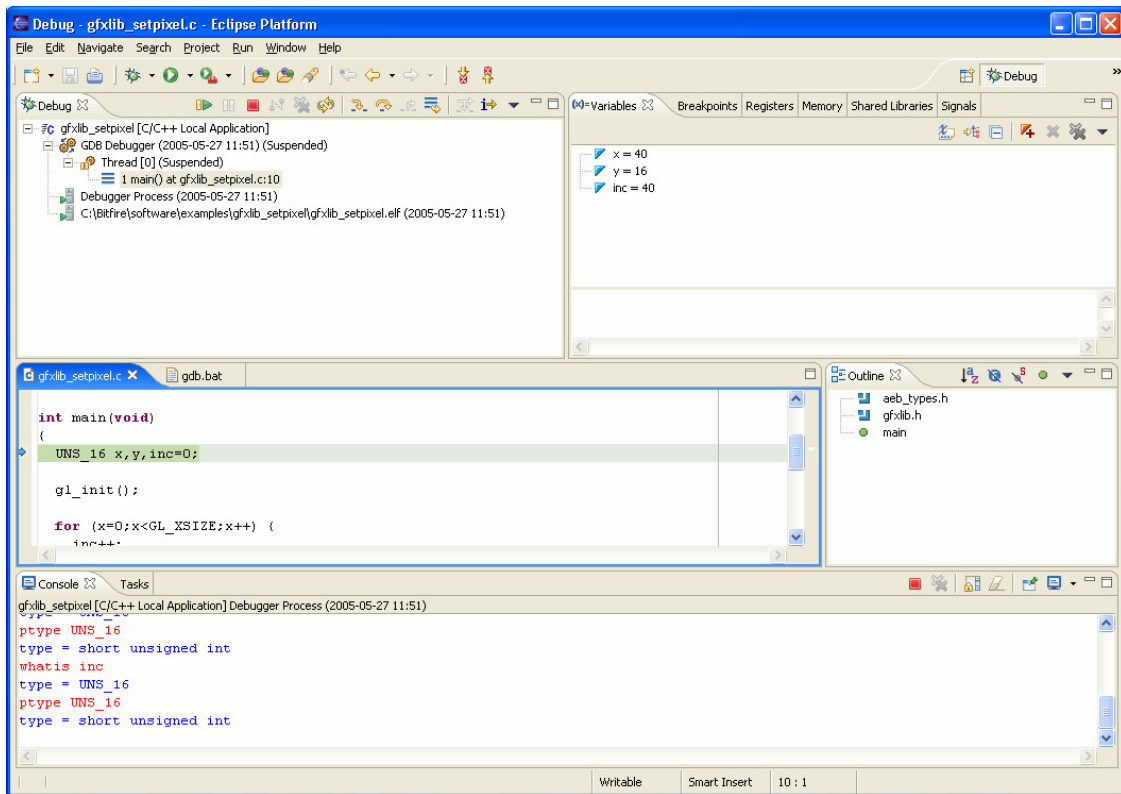


7. Go to the Debugger Tab and select Embedded GDB in the list at the top.
8. Change gdb to arm-elf-gdb in the GDB Debugger textbox.
9. Press the Browse button next to the GDB command file and select the file `.gdbinit` described above.

10. Please note that gdb command **load** has to be run in this script. It might be commented out in some of the supplied .gdbinit files. If that's the case, please edit the file.



11. Press Debug and wait for Eclipse to switch for Debug perspective.
12. You should now have the following screen and have stopped at main().

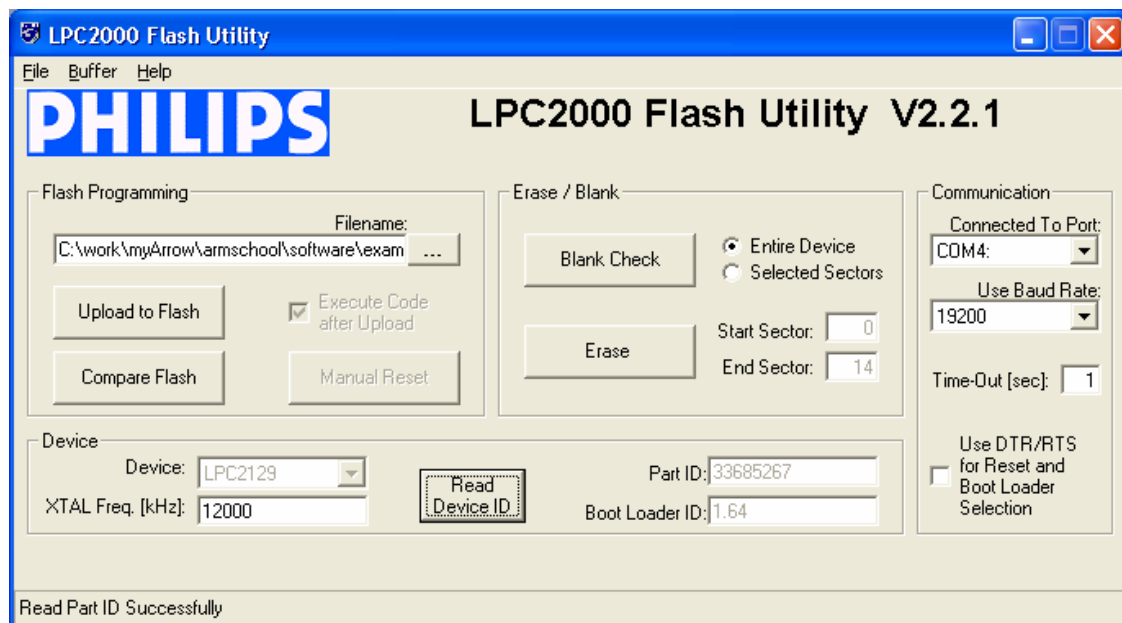


13. At this point you can step, set breakpoints, look at registers/memory like you are used to. For more eclipse documentation, please go to www.eclipse.org

6 Running an example from Flash

The Bitfire ARM CPU (LPC2129) has only 16KB RAM, but 256KB of Flash. If you write more than simple test programs, you will quite soon not be able to use the RAM for both code and data. In this case you have to build your programs to execute from flash.

1. Make sure the Bitfire board is setup correctly; see section 'Appendix B: Setting up the Bitfire board for programming the CPU flash'.
2. Open a Cygwin shell and use `cd` to enter the `software/examples/gfxlib_setpixel` directory
3. Make sure the example is correctly built for **flash**, see section 3. The difference is that you should run the **make** (not 'make ram') command.
4. Start the **LPC2100 Flash Utility**; it should be located in your start menu. Set it up to look like this.



Change the COM: port to the one you use on you PC. Press the **Read Device ID** button and the PartID and Boot Loader ID should fill up like above.

5. Select the current .hex file in the Filname textbox. For this example the correct hexfile is: `software/examples/gfxlib_setpixel/gfxlib_setpixel.hex`
6. Press the **Upload to Flash** button and wait for it to finish.
7. Now you can reset to board, see the Bitfire Hardware Reference Guide for details. The example should automatically run (and the LED matrix will light up).

7 Debugging from Flash

You can debug a program that is executed from flash. This is very similar to debugging a program running from RAM, with one major difference. Since you are running from

Flash, you can't set breakpoints in the same manner. You have to use the hardware breakpoint support in the ARM core, and there are only 2 of them in the Bitfire's CPU. This also has implications on the step feature, since breakpoints are used to for this function. The only safe step function when running from flash is `stepi` (step instruction). This will make stepping in "C Source Mode" impossible since you are stepping over an unknown number of instructions per line of C code. Note that some debuggers support stepping in flash with a technique of allocating one hardware breakpoint solely for stepping.

UPDATE openocd have a feature in which it forces all breakpoints (including temporary one used by step) to be hardware breakpoints. Send to command `mon mon arm7_9 force_hw_bkpts` enable at the gdb prompt. This makes stepping possible in flash!

7.1 Using GDB

1. Compile and program the corresponding hex file to CPU's flash as described in section 6.
2. Setup the Bitfire board for debugging as described in 'Appendix A: Setting up the Bitfire board for running and debugging'.
3. Open a Cygwin shell and use `cd` to enter the `software/examples/gfxlib_setpixel` directory.
4. Run the command `arm-elf-gdb gfxlib_setpixel.elf` command. Now you should have entered GDB and have a (gdb) prompt.
5. Type `target remote 127.0.0.1:8888`
This will force GDB to make a TCP connection with the openocd tool.
Note: Always use the IP-address '127.0.0.1' when running the openocd and GDB on the same computer.
6. At this point we don't need to download the code since it's already in the flash! However, you will need to reset the PC register. Type `set $pc=0`
7. The ARM7 core has only 2 hardware breakpoints as described above, you can tell GDB to check the number of breakpoints you have set, this is done with the command `set remote hardware-breakpoint-limit 2`
8. To set a hardware breakpoint write `hb main`.
9. To start to CPU type `c` (for continue).
10. You should now have stopped at the main breakpoint.

Here are some handy GDB commands that you can use when debugging:

- `i br` (info breakpoints) - Lists your breakpoints.
- `del x` (delete x) - Delete breakpoint X
- `i reg` (info registers) - Show registers
- `i s` (info stack) - Show call stack
- `i loc` (info local) - Show local variables
- `x 0x123456789` (examine) - Look at memory at location 0x12345678
- `p i` (print expression) - Print result of generic expression, can be anything from function calls to variables etc.

8 The Bitfire BSP

The BSP that ships with Bitfire is simple and only consists of a few basic drivers. Not all peripherals are supported at this time, but Arrow Engineering is committed to continue to create drivers and update Bitfire users. See ‘Appendix C: BSP Reference’ for a detailed list of what’s available.

8.1 Building the libbitfirebsp.a

Building the BSP library is very simple. Just open a Cygwin shell and go to `software/bsp/bitfire/lib`. Here you simply execute the `make` command.

9 The Bitfire Graphics Library

The Graphics library was created to make it easy for developers to use the LED matrix and to show off the power and possibilities of the Bitfire platform. See the Bitfire Hardware Reference Guide for details on the LED matrix, and the FPGA Developers Guide on how the matrix is controlled.

The library consists of basic draw functions, functions for writing text, a small collection of graphical effects and finally functions for real-time 3D vector graphics rendering☺. See the ‘Appendix D: Graphics Lib Reference’ section for a more detailed list of what’s available.

9.1 Building the libgfx.a

Building the gfx lib is very simple. Just open a Cygwin shell and go to `software/gfxlib/lib`. Here you simply execute the `make` command.

10 The Examples

There are a couple of example C projects that show the usage of the BSP and graphics library. Instead of describing how to use the BSP and Graphics library in this guide, you will find many examples of how to do it in source-code. The examples can be found in the `software/examples/bitfire` directory.

- **can_polled**
An example of the polled CAN driver.
- **demoreel**
A demo reel with effects and scrolling text.
- **gpio**
An example of how to control the LPC2129 gpio pins.
- **spi_pixelttest**
An example of how to set pixels without using the graphics library.

- **spi_polled**
An example of how to use the spi peripheral.
- **testsuite**
A complete testsuite for all the different peripherals on the Bitfire board, including the CAN interface.
- **uart_int**
An example of the interrupt-driven uart driver.
- **uart_polled**
An example of the polled uart driver.

The bitfire/atmosfire platform independent examples can be found in the **software/examples/** directory.

- **gfxlib_effects**
An example of how to setup and run the different effects.
- **gfxlib_setpixel**
An example of how to set a pixel with the graphics library.
- **gfxlib_text**
An example of how to write text to the screen with the graphics library.

10.1 How the example programs work

The entry point of all examples is called `_startup` and is defined in the `software/bsp/src/startup_gnu.s` file. The `startup_gnu.s` file defines the vector table, beginning with the reset vector. This vector table will be placed correctly at address 0x0 when written to flash and at address 0x40000000 (beginning of RAM) when run from RAM. In the case of running from RAM, the original exception vector table in flash will still be used. The LPC2129 supports remapping a part of it's RAM to address 0x0 in order for programs to change the vectors without writing to flash, however this is not done in the standard examples.

At the reset vector the code defined by `startup_gnu.s` sets up the hardware; VPBBUS dividers, PLL, MAM. Then the stacks of the different modes in the ARM core is setup, the data region relocated to RAM (if needed) and BSS section initialized to zero.

The symbol `main` is entered in ARM USR mode with IRQs and FIQs enabled.

When the examples are built one of two linker scripts is used, either `software/src/ram_gnu.ld` or `software/src/flash_gnu.ld`. This is the files that dictates whether the code should be linked to run from flash or RAM.

A very useful tool to investigate what-went-were in an .elf file is the command `arm-elf-nm`

11 Ported Operating Systems

With the Bitfire development kit you get 2 pre-ported operating systems; FreeRTOS™ and uC/OS-II. There is a demo included with an example of how to write code for them. Note that both examples need to be run from flash.

The demos are the same for the different operating systems. It consists of multiple tasks and 2 semaphores, one semaphore for the top and bottom 8 pixels on the LED matrix. All tasks try to take one of the semaphores and scroll their taskname on that part of the LED matrix. After scrolling is done the task releases the semaphore and goes to sleep for a random amount of clock ticks.

11.1 FreeRTOS™

FreeRTOS™ is a portable, open source, mini Real Time Kernel - a free to download and royalty free RTOS created by Richard Berry. All info about this OS can be found on the www.freertos.org webpage.

The Bitfire port can be found in `software/rtos/FreeRTOS` directory. The example is located in `software/rtos/FreeRTOS/Demo/ARM7_LPC2129_GCC`

11.2 uC/OS-II

uC/OS-II, The Real-Time Kernel is a highly portable, ROMable, very scalable, preemptive real-time, multitasking kernel (RTOS) for microprocessors and microcontrollers. All info about this OS can be found on the www.micrium.com webpage.

The Bitfire port can be found in `software/rtos/MICRIUM` directory. The example I located in `software/rtos/MICRIUM/SOFTWARE/EvalBoards/Philips/Bitfire/GCC/app`

Note: The source files for this OS are not shipped with the Bitfire board. Instead a precompiled lib which includes the kernel and LPC2129 port is provided. The specifications on the setup of the kernel can be found in the `software/rtos/MICRIUM/SOFTWARE/EvalBoards/Philips/Bitfire/GCC/lib/os_cfg.h` file.

If you want to use this OS in your application you need to contact Micrium and arrange a license agreement with them.

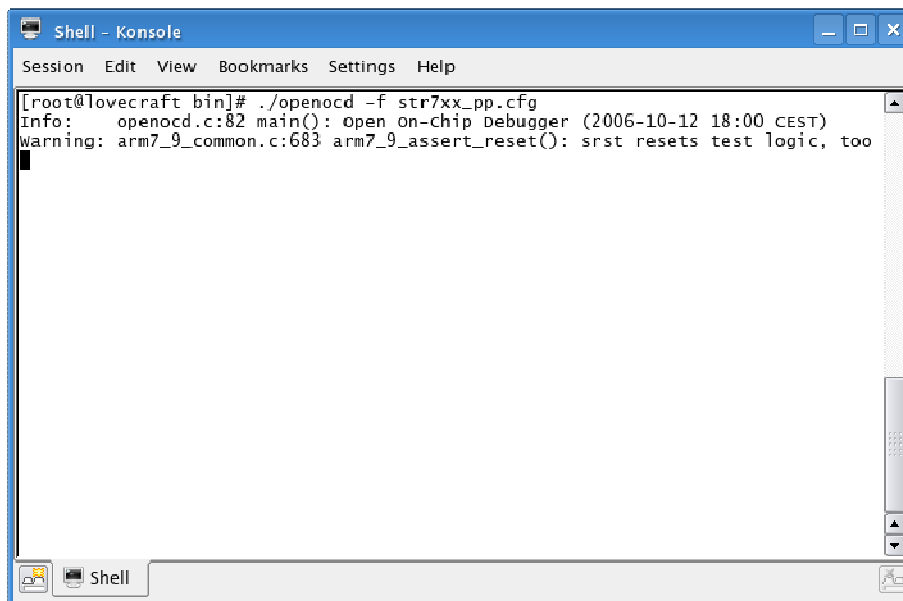
12 Appendix A: Setting up the Bitfire board for running and debugging

Every time you want to load code for running or debugging you'll have to make sure that the following is setup correctly.

1. Plug a parallel cable from your PC to the Bitfire board. See the Bitfire Hardware Reference Guide for details.
2. Power up the Bitfire board, see the Bitfire Hardware Reference Guide for details.
3. Put the CPU in J-TAG mode (not the External J-TAG mode). See the Bitfire Hardware Reference Guide.
4. After this you need to run the openocd tool. Open a Windows[®] command prompt and enter the directory where you installed the openocd (suggested `C:\Bitifre\openocd\bin`). Run the command `openocd-pp.exe -f configs\lpc2xx_pp.cfg`

Please note that in openocd requires the Parallel port of your PC to be in EPP (or ECP) mode. This is usually controlled in the BIOS setup.

You should get the following output:



```
Shell - Konsole
Session Edit View Bookmarks Settings Help
[root@lovecraft bin]# ./openocd -f str7xx_pp.cfg
Info: openocd.c:82 main(): open on-chip debugger (2006-10-12 18:00 CEST)
Warning: arm7_9_common.c:683 arm7_9_assert_reset(): srst resets test logic, too
```

13 Appendix B: Setting up the Bitfire board for programming the CPU flash

When you want to program the Bitfire CPU's flash you have to make sure that the following is setup correctly.

1. Plug a serial cable from your PC to the Bitfire Uart-0 connector, see the Bitfire Hardware Reference Guide for details.
2. Power up the Bitfire board, see the Bitfire Board Guide Bitfire Hardware Reference Guide for details.
3. Force the CPU into Bootloader mode, see the Bitfire Board Guide Bitfire Hardware Reference Guide.

14 Appendix C: BSP Reference

14.1 Include files

14.1.1 **bitfire.h**

Defines the speed of the CPU and VPB clock.

14.1.2 **bitfire_fpga.h**

Functions for sending commands and data to the Bitfire FPGA. This is controlled by GPIO P0.12 and P0.13. Defines FGPACMD_* constants.

14.1.3 **lpc2129_can.h**

Common include file for CAN drivers. Defines CAN_* constants. Defines struct can_control

14.1.4 **lpc2129_can_polled.h**

Polled CAN driver

14.1.5 **lpc2129_spi.h**

Common include file for SPI drivers. Defines SPCR_* and SPI_* constants. Defines struct spi_control

14.1.6 **lpc2129_spi_polled.h**

Polled SPI driver

14.1.7 **lpc2129_uart.h**

Common include file for UART drivers. Defines LCR_*, UART_* constants. Defines struct uart_control

14.1.8 **lpc2129_uart_int.h**

Interrupt-driven UART driver

14.1.9 **lpc2129_uart_polled.h**

Polled UART driver

14.1.10 **lpc2129_vic.h**

Vectored Interrupt Controller (VIC) Driver. Defines VIC_* constants. Defines struct vic_control

14.1.11 **lpc21xx.h**

All register addresses for LPC2129

14.2 Function Descriptions

14.2.1 bitfire_fpga

14.2.1.1 void fpga_command (void)

Setup the FPGA for command communication. Must be called once before each command.

14.2.1.2 void fpga_data (void)

Setup the FPGA for data communication.

14.2.1.3 void fpga_init (void)

Initializes the GPIO ports that are needed to control the FPGA.

14.2.2 lpc2129_can_polled

14.2.2.1 void can_polled_clear_rxbuf (can_control * cc)

Clears the RX buffer in the CAN peripheral

Parameters:

*cc Pointer to initialized can_control struct. You must call **can_polled_init()** first.

14.2.2.2 void can_polled_init (can_control * cc)

Initializes the CAN peripheral

Parameters:

*cc Pointer to initialized can_control struct. Will update the status field.

Example usage:

```
can_control tx = {1, 0x55, 20000000, 0};  
can_polled_init(&tx);
```

14.2.2.3 UNS_8 can_polled_send (can_control * cc, UNS_8 dlc, UNS_8 * data)

Writes 1-8 8bit words to the CAN peripheral

Parameters:

*cc Pointer to initialized can_control struct. You must call **can_polled_init()** first.

dlc Counter-value of how many 8bit words to send.

*data Pointer to 8bit databuffer with data to send.

14.2.2.4 void can_polled_set_filters (void * filters)

Set acceptance filters in the CAN peripheral

Parameters:

*filters Pointer to a filter struct. Please note that this function is not yet implemented! If sets the CAN periferal in bypass mode (no filter mode).

14.2.2.5 UNS_8 can_polled_receive (can_control * cc, UNS_8 * data)

Reads 1-8 8bit words from the CAN peripheral. Blocks until data is ready.

Parameters:

*cc Pointer to initialized can_control struct. You must call **can_polled_init()** first.

*data Pointer to 8bit databuffer of max length 8.

14.2.3 lpc2129_spi_polled

14.2.3.1 UNS_8 spi_polled_getchar (spi_control * sc)

Reads 8 bits from the SPI peripheral. NOT IMPLEMENTED!

Parameters:

*sc Pointer to initialized spi_control struct.

14.2.3.2 void spi_polled_init (spi_control * sc)

Initializes the SPI peripheral

Parameters:

*sc Pointer to initialized spi_control struct. Will update the status field.

Example usage:

```
spi_control sc = {0, 0,  
                 SPCR_MSTR_MASTER | SPCR_LSBF_LSB,  
                 SPCCR_FASTEST, 0};  
spi_polled_init(&sc);
```

14.2.3.3 UNS_8 spi_polled_putchar (spi_control * sc, UNS_8 ch)

Writes 8 bits to the SPI peripheral

Parameters:

*sc Pointer to initialized spi_control struct. You must call **spi_polled_init()** first.

ch Containing the 8 bits to be sent.

14.2.4 lpc2129_uart_int

14.2.4.1 UNS_8 uart_int_getchar (uart_control * uc, UNS_8 * err)

Reads 8 bits from the UART peripheral. This is non-blocking call.

Parameters:

*uc Pointer to initialized uart_control struct.

*err Pointer to a 8bit character that will hold the error code. It will be one of the following:
UART_ERR_NO_ERROR, UART_ERR_NOT_INIT, UART_ERR_NO_CHAR_READY

Returns:

Read character

14.2.4.2 void uart_int_init (uart_control * uc)

Initializes the UART peripheral

Parameters:

*uc Pointer to initialized uart_control struct. Will update the status field.

Example usage:

```
uart_control uart0;  
uart0.port=0;  
uart0.LCR = LCR_WLS_8BIT | LCR_SBS_1BIT | LCR_PE_0 | LCR_PS_ODD  
| LCR_BK_0 | LCR_DLAB_1;  
uart0.baudrate = 115200;  
uart0.vecaddr=2;  
  
uart_int_init(&uart0);
```

14.2.4.3 UNS_8 uart_int_putchar (uart_control * uc, UNS_8 ch)

Writes 8 bits to the UART peripheral

Parameters:

*uc Pointer to initialized uart_control struct. You must call **uart_int_init()** first.

ch Containing the 8 bits to be sent.

14.2.5 lpc2129_uart_polled

14.2.5.1 UNS_8 uart_polled_getchar (uart_control * uc)

Reads 8 bits from the UART periferal. Blocks until data is ready.

Parameters:

*uc Pointer to initialized uart_control struct.

Returns:

Read character

14.2.5.2 void uart_polled_init (uart_control * uc)

Initializes the UART peripheral

Parameters:

*uc Pointer to initialized uart_control struct. Will update the status field.

Example usage:

```
uart_control uart0;  
uart0.port=0;  
uart0.LCR = LCR_WLS_8BIT | LCR_SBS_1BIT | LCR_PE_0 | LCR_PS_ODD | LCR_BK_0 |  
LCR_DLAB_1;  
uart0.baudrate = 115200;  
  
uart_int_init(&uart0);
```

14.2.5.3 UNS_8 uart_polled_putchar (uart_control * uc, UNS_8 ch)

Writes 8 bits to the UART peripheral

Parameters:

**uc* Pointer to initialized `uart_control` struct. You must call `uart_polled_init()` first.
ch Containing the 8 bits to be sent.

14.2.6 `lpc2129_vic`

14.2.6.1 `void vic_disable_int (vic_control * vc)`

Disable the interrupt for an installed ISR.

Parameters:

**vc* Pointer to initialized `vic_control` struct.

14.2.6.2 `void vic_enable_int (vic_control * vc)`

Enable the interrupt for an installed ISR.

Parameters:

**vc* Pointer to initialized `vic_control` struct.

14.2.6.3 `void vic_install_isr (vic_control * vc)`

Installs an ISR at a specific Vector Address.

Parameters:

**vc* Pointer to initialized `vic_control` struct.

Example usage:

```
vic_control vc;  
vc.vecaddr = 0;  
vc.vecchannel = VIC_CH_UART0;  
vc.func = (void*)uart0_int_isr;  
vic_install_isr(&vc);
```

14.2.6.4 `void vic_remove_isr (vic_control * vc)`

Uninstall a previously installed ISR

Parameters:

**vc* Pointer to initialized `vic_control` struct.

15 Appendix D: Graphics Lib Reference

15.1 Win32 emulation

In the directory `software/win32_emulation` you will find a Visual Studio© project containing an emulation environment for the LED matrix. The purposed of this environment is to make it easier to create graphical functions for the Bitfire board. This guide will not go into further details of how the emulation works, but it's pretty straight forward.

15.2 Include files

15.2.1 `gfxfx.h`

The Graphics Library effects part.

15.2.2 `gfxlib.h`

The Graphics Library main functions and structs.

15.2.3 `gfmtxt.h`

The Graphics Library text part.

15.2.4 `gfxvec.h`

The Graphics Library 3D vector part.

15.3 Function Descriptions

15.3.1 `gfxfx`

15.3.1.1 `void glfx_3dstar (glfx_star * stars, UNS_8 starnum)`

Animated 3D star field, uses floats.

Parameters:

**stars* pointer to array of `glfx_star` structs (init to 0)
starnum number of stars

15.3.1.2 `BOOL_8 glfx_charsinscroll (CHAR * text, gl_col * col, const INT_8 * sintable, UNS_32 * offset)`

Scrolltext where the chars follows sinvalues. This takes up whole screen. Writes to working buffer.

Parameters:

**text* String to scroll
**col* Pointer to color(r,g) struct.
**sintable* should have XSIZE (40) elements and amplitude of YSIZE/2 (8).
**offset* internal counter, is updated, start at 0.

Returns:

Returns True when scroll is done, otherwise 0.

15.3.1.3 void glfx_fire (UNS_8 * *pbuf*, UNS_8 *YSIZE*)

Fire effect You need to call gl_pblit (with a valid palette) after each glfx_fire() call to update the working buffer.

Parameters:

**pbuf* local screen buffer that consists of UNS_8 palette-colored indexes.
YSIZE number of XSIZE-ed lines in the pbuf buffer.

15.3.1.4 void glfx_fire2 (UNS_8 * *pbuf*, UNS_8 *YSIZE*)

Fire effect with 'real' flames You need to call gl_pblit (with a valid palette) after each glfx_fire() call to update the working buffer.

Parameters:

**pbuf* local screen buffer, that consists of UNS_8 palette-colored indexes.
YSIZE number of XSIZE-ed lines in the pbuf buffer.

15.3.1.5 void glfx_init_greenfire_palette (gl_col * *palette*)

Generate 256 color palette, biased on green.

Parameters:

**palette* pointer to palette buffer, 256 gl_cols big.

15.3.1.6 void glfx_init_radient_palette (gl_col * *palette*)

Generate 256 color 'radient' palette, biased on red.

Parameters:

**palette* pointer to palette buffer, 256 gl_cols big.

15.3.1.7 void glfx_init_redfire_palette (gl_col * *palette*)

Generate 256 color palette, biased on red.

Parameters:

**palette* pointer to palette buffer, 256 gl_cols big.

15.3.1.8 void glfx_init_yellowfire_palette (gl_col * *palette*)

Generate 256 color palette, biased on yellow.

Parameters:

**palette* pointer to palette buffer, 256 gl_cols big.

15.3.1.9 void glfx_julia (INT_32 *maxiter*, UNS_8 *co*)

Draw a Juila fractal on the whole workbuf. (Uses floats).

Parameters:

maxiter maximum number of iterations in fractal loop.
co color offset

15.3.1.10 void glfx_mandel (INT_32 *maxiter*, UNS_8 *co*)

Draw a Mandelbrot fractal on the whole workbuf. (Uses floats).

Parameters:

maxiter maximum number of iterations in fractal loop.
co color offset

15.3.1.11 void glfx_pattern1 (UNS_32 * *offset*)

Write a simple fullscreen cyclic pattern ver 1 to working buffer.

Parameters:

**offset* pointer to offset counter that controls the pattern cycle. Needs to be incremented by calling function.

15.3.1.12 void glfx_pattern2 (UNS_32 * *offset*)

Write a simple fullscreen cyclic pattern ver 2 to working buffer.

Parameters:

**offset* pointer to offset counter that controls the pattern cycle. Needs to be incremented by calling function.

15.3.1.13 void glfx_pattern3 (UNS_32 * *offset*, const INT_8 * *sintable*, UNS_16 *sinlen*)

Write a vertical fullscreen cyclic bars to working buffer.

Parameters:

**offset* pointer to offset counter that controls the pattern cycle. Needs to be incremented by calling function.
**sintable* pointer to sintable that defines the bars
sinlen length of the sintable.

15.3.1.14 void glfx_pattern4 (UNS_32 * *offset*, const INT_8 * *sintable*, UNS_16 *sinlen*)

Write a horizontal fullscreen cyclic bars to working buffer.

Parameters:

**offset* pointer to offset counter that controls the pattern cycle. Needs to be incremented by calling function.
**sintable* pointer to sintable that defines the bars
sinlen length of the sintable.

15.3.1.15 void glfx_plasma (UNS_8 * *pbuf*, const INT_8 * *costable*, glfx_plasmavars * *pvs*)

Plasma effect You need to call gl_pblit (with a valid palette) after each glfx_plasma() call to update the working buffer.

Parameters:

**pbuf* local screen buffer, that consists of UNS_8 palette-colored indexes.
**costable* should have 256 elements.
**pvs* struct plasmavars, will be overwritten, init all to 0.

15.3.1.16 BOOL_8 glfx_scrolltext (CHAR * *text*, gl_point * *pos*, gl_col * *col*, UNS_32 * *offset*)

Normal scrolltext in working buffer.

Parameters:

**text* String to scroll
**pos* y top scrolled line
**col* Pointer to color(r,g) struct.
**offset* internal counter, is updated, start at 0.

Returns:

Returns True when scroll is done, otherwise 0.

15.3.2 gfxlib

15.3.2.1 void gl_circle (gl_point * *p*, UNS_8 *r*, gl_col * *c*)

Draw a non-filled circle (Bresenahm) of radius r. (Uses gl_setpixel).

Parameters:

**p* center of the circle.
r radius of the circle.
**c* color of the circle.

15.3.2.2 void gl_cliprect (gl_rect * *rect*)

Modify a rect so it fits within the screen.

Parameters:

**rect* Pointer to rect struct to be modified.

15.3.2.3 void gl_clrscr (gl_col * *col*)

Clear the current work buffer. (Uses SPI0)

Parameters:

**col* Pointer to color(r,g) struct. Example usage:

```
gl_clrscr(&glcol_black);
```

15.3.2.4 void gl_ellipse (gl_point * *p*, UNS_8 *a*, UNS_8 *b*, gl_col * *c*)

Draw a non-filled ellipse (Bresenahm). (Uses gl_setpixel).

Parameters:

**p* center of the ellipse.
a x radius.
b y radius.
**c* color of the ellipse

15.3.2.5 void gl_fillrect (gl_rect * *rect*, gl_col * *col*)

Fill a gl_rect in work buffer with a given color. (Uses SPI0)

Parameters:

**rect* Pointer to rect(left,top,right,bottom) struct
**col* Pointer to color(r,g) struct. Example usage:

```
gl_rect r = {7,7,20,15};  
gl_fillrect(&r,&glcol_yellow);
```

15.3.2.6 void gl_flipbuf (void)

To control double buffering. This function flips working and show buffer. The buffer flip is done at next vertsync by the FPGA, so you don't have to bother about waiting for sync before calling this function. Must be called once before first set_pixel(). If you don't want to use double buffering you can ignore this function, by default work buffer=show buffer=0 (Uses SPI0) Example usage:

```
gl_init();  
gl_flipbuf();           // Initialize dubble buffering  
..  
gl_setpixel(&p,&c);  
gl_flipbuf();           // Drawing done -- flip buffer!  
..
```

15.3.2.7 void gl_init ()

Initialize the gfxlib (Will setup and use SPI0!)

15.3.2.8 BOOL_8 gl_iswithin (gl_point * *p*, gl_rect * *r*)

Check whether a point is within a rect.

Parameters:

**p* Pointer to point struct
**r* Pointer to rect struct.

15.3.2.9 void gl_line (gl_point * *p1*, gl_point * *p2*, gl_col * *c*)

Draw a line (Bresenham) from p1 to p2 into work buffer. (Uses gl_setpixel).

Parameters:

**p1* position 1 of gl_point struct.
**p2* position 2 of gl_point struct.
**c* color of the line of gl_col struct.

15.3.2.10 void gl_pblit (UNS_8 * *source*, gl_rect * *srect*, gl_point * *dpos*, gl_col * *palette*)

Blit from a palette bitmap into the work buffer (with palette translations). (Uses SPI0)

Parameters:

**source* Pointer to 8 bit palette bitmap buffer. Note this buffer has to have the same XSIZE as the work buffer!

**srect* Source rect within the bitmap buffer to blit.

**dpos* Start position in work buffer to start blitting to.

**palette* Pointer to the palette.

15.3.2.11 void gl_printhexdot (INT_32 *val*, gl_point * *p*, gl_col * *c*)

Print a 32bit interger as 32 dots on the LED matrix. This is useful for debug purposes. (USES gl_setpixel).

15.3.2.12 UNS_32 gl_random ()

Generate a random number (uses the c runtime lib).

15.3.2.13 void gl_setpixel (gl_point * *pos*, gl_col * *col*)

Set a pixel in current working buffer. (Uses SPI0)

Parameters:

**pos* Pointer to position(x,y) struct.

**col* Pointer to color(r,g) struct. Example usage:

```
gl_point p = {5,10};  
gl_col c = {128,64};  
gl_setpixel(&p,&c);
```

15.3.2.14 void gl_setpixelxy (UNS_16 *x*, UNS_16 *y*, gl_col * *col*)

Set a pixel in current working buffer. Doesn't use the gl_point struct. (Uses SPI0)

Parameters:

x x position.

y y position.

**col* Pointer to color(r,g) struct.

15.3.3 gftxt

15.3.3.1 void gltxt_printhex (UNS_32 *hexval*, gl_col * *col*, BOOL_8 *shortval*)

Print a hexadecimal value to the LED screen. (uses gltxt_renderchar()) The value can either be 32 or 16 of length.

Parameters:

hexval Value to print

**col* Pointer to color(r,g) struct.

shortval control to write 16 or 32 bit value.

15.3.3.2 void gltxt_printtext (CHAR * *str*, gl_point * *pos*, gl_col * *col*)

Print a string to the LCD screen. No automatic line-feed is done. (uses `gltxt_renderchar()`)

Parameters:

**str* Pointer to string to render.

**pos* Pointer to position(x,y) struct.

**col* Pointer to color(r,g) struct.

```
gltxt_setfont(font_rom8x8_bits);  
gl_point p = {3,3};  
gltxt_printtext("Hello!", &p, &glcol_green);
```

15.3.3.3 void gltxt_renderchar (CHAR *ch*, gl_point * *pos*, gl_col * *col*)

Render a character to the LCD screen. (uses `gl_setpixel()`)

Parameters:

ch Charartcher to print.

**pos* Pointer to position(x,y) struct.

**col* Pointer to color(r,g) struct.

15.3.3.4 void gltxt_setfont (const UNS_16 * *font*)

Set the font to use in upcoming text functions.

Parameters:

**font* Pointer to font.

15.3.4 gfxvec

15.3.4.1 void InitGVectors (const INT_16 * *data*, BOOL_8 *setuptables*)

Initialize a 3D object and the rendering engine.

Parameters:

**data* pointer to vertex and polygon data.

setuptables whether of not to reset the position and angels of the object

15.3.4.2 void UpdateGVectors (INT_16 *distance*, gl_col * *c*)

Render next screen of 3D object. Angels are automatically incremented.

Parameters:

distance drawing distance (zoom level)

**c* color of the lines

