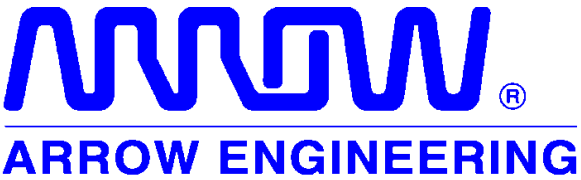
Atmosfire Development Kit
Atmosfire Software Developers Guide
Version 1.0



This page is intentionally left blank.

Revision History		
Version	**Date**	**Updates**
1.0	070515	Document public release.

Arrow Engineering Sweden, Malmö, Sweden

1 Overview

Most of the software development for Atmosfire is identical to development for Bitfire. All tools are the same, please read the Bitfire documentation before you proceed.

2 Atmosfire Source Library

The Atmosfire source library contains C and assembler BSP drivers and examples described later in this guide. The files are located in the **bitfire_atmosfire_sw_src.zip** file. Please refer to the bitfire software developer's guide.

2.1 The Atmosfire examples

There are a few examples 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/atmosfire` directory.

- **71xboot-r10**
  This is the str7xx linux bootloader from ST. Ported to GNU tools by Martin Trojer.

- **adc12/adc_int**
  This example demonstrates how to use the ADC12 in single channel conversion mode and switch on a led connected to Port 0 according to the conversion result using the ADC12 interrupt service routine.

- **adc12/polling**
  This example shows how to use the ADC12 in single channel conversion mode.

- **apb**
  This example is just a theoretical program which demonstrates how to use the APB software library and shows some features of the APB peripheral.

- **bspi**
  In this example BSPI0 and BSPI1 are connected together on the same bus. BSPI0 is configured as Master and BSPI1 as Slave.

- **bsp_pixeltest**
  This example shows how to communicate with the Bitfire FPGA over a SPI channel and set a pixel.

- **can**
  This example demonstrate one of the feature of the CAN. The program runs in loopback mode combined with silent mode (i.e. self-test mode) to be independent

from any true CAN network.

- **emi**
  This example demonstrates one of the EMI feature which is to copy part of the code from a memory to another one and then execute from it.

- **flash**
  This example shows how to use some of the flash software library routines.

- **gpio**
  This example is a theoretical program which presents some of the features of the GPIO peripheral using the library functions.

- **i2c**
  This example shows how to use the I2C software library.

- **minrtos**
  This is the minimal required code that is needed by an RTOS to function. It consists of a timer interrupt and an interrupt driven UART. It compiles without the libatmosfirebsp.a

- **pcu**
  This example highlights one of the features of the PCU which puts the chip in stop mode and use the wakeup pin to resume program execution.

- **rccu**
  This example shows how to use the RCCU peripheral in order to configure the system clock signals. This is done by setting the clock source and using divider/multiplier ratios.

- **rtc**
  This example demonstrates how to configure the RTC in order to generate an IRQ every second using the RTC second interrupt.

- **spi_flash**
  This examples shows how to communicate with spi flash on the atmosfire board without sending any unwanted data to the bitfire FPGA.

- **spi_flash_driver**
  This example uses a modified m25pxx spi flash driver from ST to control the spi flash device on the atmosfire board.

- **sram_test**
  This example stress testes the external SRAM on the atmosfire board.

- **syscalls**
  This example shows how to get newlibs's printf(), malloc() and free() functions to work on a 'bare-metal' atmosfire with no OS.

- **tim/Music**
  This example highlights one of the features of the TIMER which is to play a PWM wave generation.

- **tim/OCMP**
  This Device is configured to Toggle the TIM3 Output compare A pin from low level to high level after 0xF000 timer period. The GPIO pin is also toggled from low level to high level and may be used as a reference to verify the delay of the Output compare function.

- **tim/OPM**
  The timer is configured to generate a pulse on the output compare A pin with configurable fixed length timer period.The pulse generation is activated by an external rising edge on the input capture A.

- **tim/PWMI**
  This example demonstrates the PWM input feature of the Timer: Timer 3 is configured to measure the frequency and the duty-cycle of an input signal using the timer Input Capture A module.

- **tim/PWMO**
  This example demonstrates the following features of the Timer: PWM output generation.

- **uart/interrupt**
  This example shows how to use the UART to communicate between the STR71x and a PC using RS232 protocol. The main program configures the UART0 to generate an interrupt after a byte reception. The received byte will be read in the UART0 interrupt service routine and echoed back to the Host PC.

- **uart/polling**
  This example shows how to use the UART to communicate between the STR71x and a PC using RS232 protocol. The main program waits for the user to send 4 ASCII characters from the PC, and the UART will echo the same characters back to the PC.

- **usb/Audio Speaker**
  Makes the STR71xx behave as an USB audio speaker.

- **usb/Device Firmware Upgrade**
  Makes the STR71xx behave as an USB DFU device.

- **usb/Joystick Mouse**
  Makes the STR71xx behave as an USB mouse device.

- **usb/Mass Storage/Double Buffer**
  Makes the STR71xx behave as an USB mass storage device with double buffer endpoints.

- **usb/Mass Storage/Simple Buffer**
  Makes the STR71xx behave as an USB mass storage device.

- **usb/Virtual Com Port**
  Makes the STR71xx behave as an USB virtual com port.

- **wdg**
  This example highlights two main features of the Watchdog Timer peripheral: Timer interrupt, Watchdog mode.

- **xti**
  This example demonstrates how to configure the XTI to generate IRQ interrupts. This example highlights one of the features of the XTI which is to generate an IRQ interrupts service routine and use the P0.15 pins to execute the interrupt code.

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.

2.2 FreeRTOS

The Atmosfire port can be found in **`software/rtos/FreeRTOS`** directory. The example is located in **`software/rtos/FreeRTOS/Demo/ARM7_STR71x_GCC`**.

See the Bitfire Software Developers Guide for more details.

3 eCos

eCos is a full blown OS for embedded systems, for more info please visit http://ecos.sourceware.org/ This document deals with building RedBoot and eCos for the atmosfire board.

The Atmosfire eCos port is still in experimental stage. Building eCos/RedBoot will require some editing of files that might seem cumbersome. Please follow the instructions below carefully.

Building eCos apps is however much smoother, you should very rarely have to rebuild RedBoot and the eCos libs. In many cases you can just use the binaries supplied on the CD.

3.1 mtshell eCos application

The mtshell eCos application contains eCos device drivers from various peripherals on the atmosfire CPU, a flash file system MTFFS (Martin Trojer Flash File System), and a port of Lua 5.0.2, Here's a short list of the commands.

- ? / help
  Write some help text
- cat <file>
  Print a files contents
- fsinfo
  Print some information about the filesystem
- kill <handle>
  Kill a thread with specific handle
- luai
  Enter an interactive lua prompt
- lua <file>
  Run the lua <file> in the background
- ls
  List contents of the filesystem
- meminfo
  Show some information regarding the eCos memory heap

- ps
  List info about threads
- rm <file>
  Delete file from filsystem
- xmodem <file> <size>
  Send a binary file to the target via the xmodem protocol. You must supply a filename and the size of the file you are sending. Note! Existing files are not truncated. This means that if the file exists and the new version of the file (the one being sent) is smaller than the current one on the target, this file will contain some junk at the end.

3.2 Installing and setting up eCos

On the Bitfire/Atmosfire CD you will find the following files related to eCos.

- source/bitfire_atmosfire_sw_src.zip
  Contains an ecos directory with eCos applications (such as mtshell).
  The ecos directory also contains a shell script called **ecosenv.sh** for setting up the environment needed to build eCos. You may need to edit this file.
- source/ecos070515_build_tree.zip
  The eCos build trees for atmosfire. Pre compiled eCos libs + RedBoot.
  Please unzip these file to software/ecos/build_tree
- source/ecos070515.zip
  CVS snapshot of eCos source tree from 20070515.
  Needed when building eCos/Redboot.
- source/ecos070515_atmosfire_prepatched.zip
  Prepatched eCos source tree with the atmosfire addons.
  Needed when building eCos/Redboot.
- source/ecos070515_atmosfire.patch.gz
  Patch containing all atmosfire additions to the eCos CVS snapshot.
  ```
  unzip ecos070515.zip; cd ecos; zcat
  ../ecos070515_atmosfire.patch.gz | patch -p1
  ```
  Note that if you use the prepatched zip file, you can skip this step.
- documentation/ecos/
  This directory contains various eCos documentation files.
- tools/ecos_tools.zip
  ecosconfig/configtool Linux and Win32 executables.
  Please unzip these file to software/ecos/tools and add to your path.

3.3 Building eCos applications

When you build eCos applications you need a valid build_tree with the target libraries, se above.

Please note that you might have to edit ecosenv.sh to match your installaion setup.

```
$ cd software/ecos
$ . ecosenv.sh
$ cd apps/mtshell
$ make
```

Now you can either load the elf file mthsell directly into memory and run it or put it in RedBoot's flash area. This application will put its output on the UART-1 on the bitfire board at a baudrate of 38400.

There are other simpler examples in the **ecos/apps** directory. They can serve as a good starting point for creating your own eCos applications.

3.4 Building eCos

Atmosfire comes with pre-build eCos and RedBoot libs/binaries. You ***don't*** have to re-build eCos/Redboot just to make new apps. See "building eCos applications " above for further instructions.

To build eCos you will need a v3 gcc compiler suite. Suggested version is binutils215, gcc343, newlib112, gdb64. In the tools directory of the CD you will find both v4 and v3 of the GCC compiler for Win32 / Linux. Install as you please.

To build eCos you will need Tcl on your system. If you're using Windows you might have to install it. There's a file called **WinTclTk-MinGW-0.5.exe** on the Bitfire CD's tool directory. Install and add the bin directory to your path. There's a caveat with this installation, it doesn't supply a tclsh command (just a tclsh84). This can be solved by copying the tclsh84.exe and renaming the copy tclsh.exe.

Install gcc3, tcl and eCos as described above. You should have a **software/ecos** directory with the following 3 subdirectories:

- ecos_atmosfire (maybe just ecos)
  This is ecos component_repository (source tree), should be treated a read-only.
- build_tree
  Here's where you build eCos target libs and RedBoot executables.
- apps
  All eCos apps goes here.

First edit and run the ecosenv.sh script so setup the environment correctly. Then run it from cygwin/bash.

```
$ cd software/ecos; . ecosenv.sh
```

Create a directory called atmosfire_ecos in build_tree/

```
$ cd build_tree; mkdir atmosfire_ecos; cd atmosfire_ecos
```

Run the ecosconfig command to fill use the atmosfire ecos template.

```
$ ecosconfig new atmosfire
```

Add mtffs, lua and FILEIO packages packages

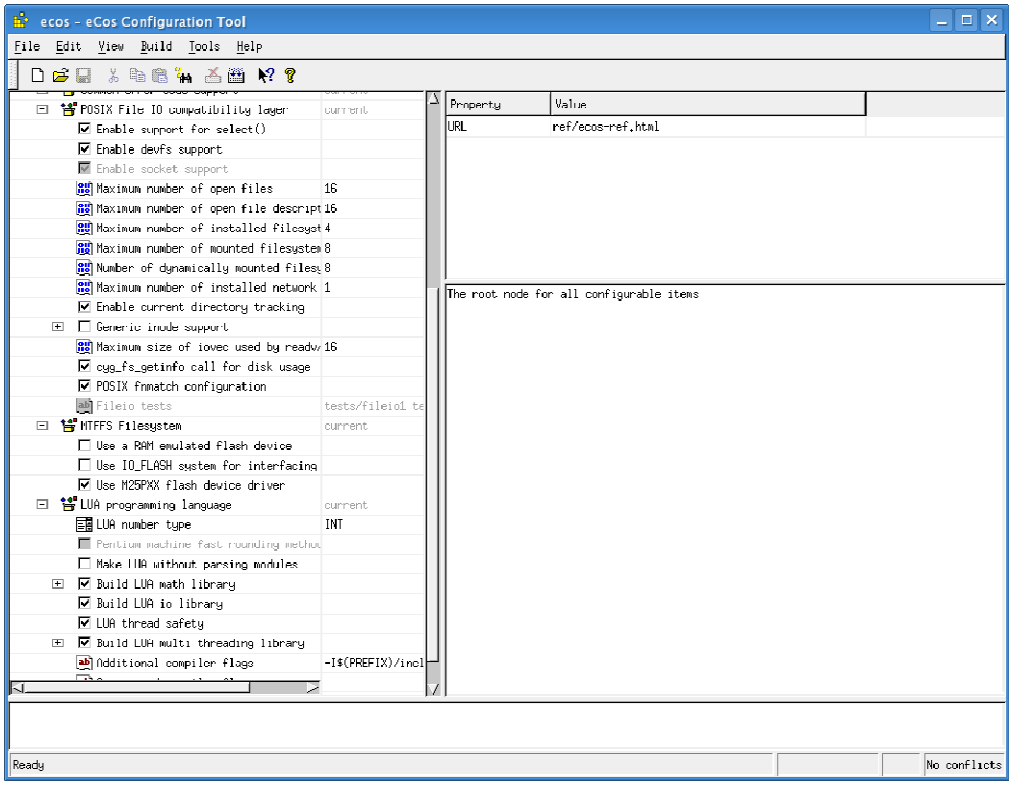
```
$ ecosconfig add CYGPKG_IO_FILEIO mtffs lua
```

Run configtool to edit some settings

```
$ configtool ecos.ecc
```

Change to following settings:

- I/O sub-system – Serial device drivers – Hardware serial device drivers
  Make sure ST STR7XX serial port 1 driver is ticked.
- ISO C library – ISO C library standard input/output – Buffered I/O
  Change Default buffer size to 4096.
- POSIX File IO compability layer
  Make sure cyg_fs_getinfo cade for disk usage is ticked.
- MTFFS Filesystem
  Make sure 'Use M25PXX flash device driver' is ticked, and only nothing else.
- LUA programming langugage
  Make sure “Build LUA multi threading library” is ticked. Click OK on the



'resolve conflicts' window.

Save and close the window.

Run the ecosconfig tree and make command at the prompt.

```
$ ecosconfig tree
$ make
```

Finally you have to edit the file called **`install/lib/target.ld`** The group line should read as follows:

```
GROUP(libtarget.a libgcc.a libsupc++.a libgfx.a libatmosfirebsp.a)
```

3.5 Building RedBoot

Install everything needed to build eCos (se above).

First edit and run the ecosenv.sh script so setup the environment correctly.

```
$ cd software/ecos; . ecosenv.sh
```

Create a directory called atmosfire_redboot in build_tree/

```
$ cd build_tree; mkdir atmosfire_redboot; cd atmosfire_redboot
```

Now we have to edit the .cdl for the atmosfire HAL. The file is called **`software/ecos/component_repository/ecos/packages/hal/arm/str7xx/current/cdl/hal_arm_str7xx.cdl`**. In **`cdl_component CYG_HAL_STARTUP`** change the default_value to "ROM". In **`cdl_option CYGBLD_GLOBAL_COMMAND_PREFIX`** change -O2 to -g in default_value.

Run the ecosconfig command to fill use the atmosfire redboot template.

```
$ ecosconfig new atmosfire redboot
```

Run configtool to edit some settings

```
$ configtool ecos.ecc
```

Change to following settings:

- Redboot ROM monitor
  Make sure Validate RAM addresses during load is not ticked.
- Redboot ROM Monitor – Allow RedBoot to support Flash Programming – Flash Image System default directory contents
  Make sure File to describe RedBoot boot image is not ticked.
  Make sure File to describe RedBoot POST-compatible image is not ticked.

```
$ ecosconfig tree
$ make
```

Now you have to edit install/lib/target.ld file. Under SECTIONS the .data line should read:

```
data ALIGN(0x4) : AT ((LOADADDR (.gcc_except_table) + SIZEOF (.gcc_except_table) + 4 - 1)
& ~ (4 - 1)) { __ram_data_start = ABSOLUTE (.); *(.data*) *(.data1) *(.gnu.linkonce.d.*)
. = ALIGN (4); KEEP(*( SORT (.ecos.table.*))) ; . = ALIGN (4); __CTOR_LIST__ = ABSOLUTE
(.); KEEP (*(SORT (.ctors*))) __CTOR_END__ = ABSOLUTE (.); __DTOR_LIST__ = ABSOLUTE (.);
KEEP (*(SORT (.dtors*))) __DTOR_END__ = ABSOLUTE (.); *(.dynamic) *(.sdata*)
*(.gnu.linkonce.s.*) . = ALIGN (4); *(.2ram.*) } > ram __rom_data_start = LOADADDR
(.data); __ram_data_end = .; PROVIDE (__ram_data_end = .); _edata = .; PROVIDE (edata =
.); PROVIDE (__rom_data_end = LOADADDR (.data) + SIZEOF(.data));
```

I.e the part you have to add is:

`AT ((LOADADDR (.gcc_except_table) + SIZEOF (.gcc_except_table) + 4 - 1) & ~ (4 - 1))` Just after `data ALIGN(0x4) :`

Finally run make again

```
$ make
```

Now you can program the file install/bin/redboot.bin into the flash of the Atmosfire CPU. See 'How to flash the atmosfire board'.

3.6 Downloading an elf info RedBoot's flash area

It's a good idea to minimize the size of the elf you're trying to download, first use the strip command.

```
$ arm-elf-strip -s mtshell
```

First flash and start RedBoot. At the RedBoot prompt do the following:

```
RedBoot> fis init
RedBoot> load -m xmodem
```

Now make your terminal program send the elf with the xmodem protocol.

```
RedBoot> fis create ecos
RedBoot> fis load ecos
RedBoot> go
```

4 Lua

Lua is a powerful, fast, light-weight, embeddable scripting language.

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

You will find a collection of manuals and tutorials on the Bitfire CD in the **documentation/lua** directory.

4.1 'Bare-metal' example

It's possible to run Lua 'bare-metal' without any OS. The example is based on lua-5.1.1. When you unzip the src file you'll find a directory called lua-5.1.1 in the software/ directory.

Enter the **etc** directory and make different examples. They are all build to run from external RAM. The best one to run on target is one.elf.

To rebuild the lua libraries do **make generic** in the lua-5.1.1 directory.

4.2 Lua bitfire library

The eCos Lua distribution on the CD contains a bitfire library for controlling the display. Here follows a short description of the API.

4.2.1 bitfire.init()

Initialize the CPU preripherals and reset the FPGA.

4.2.2 bitfire.flipbuf()

Change display / work buffers. Double buffered graphic rendering.

4.2.3 bitfire.setpixel(x,y,r,g)

Set pixel (x,y) with color (r,g)

4.2.4 bitfire.clrscr()

Clear the current working display.

4.2.5 bitfire.fillrect(left,top,right,bottom,r,g)

Fille the rectangle defined by left(x), top(y), right(x), bottom(y) with color (r,g)

4.2.6 bitfire.line(x1,y1,x2,y2,r,g)

Draw a line from (x1,y1) to (x2,y2) with color (r,g)

4.2.7 bitfire.circle(x,y,r,r,g)

Draw a circle with center (x,y) and radius r with color (r,g)

4.2.8 bitfirte.ellipse(x,y,a,b,r,g)

Draw a ellipse with center (x,y), x radius a and y radius b with color (r,g)

4.2.9 bitfire_renderchar(ch,x,y,r,g)

Draw the char ch at (x,y) with color (r,g)

4.2.10 bitfire.printtext(str,x,y,r,g)

Print (char*) str at (x,y) with color (r,g). No clipping.

4.2.11 bitfire.printhex(val,r,g)

Prints the value in val (hex formatted) with color (r,g)

4.2.12 bitfire.scrolltext(text,x,y,r,g,offset,double)

Scroll (char *) text on instance starting from (x,y) with color (r,g). Offset incremented and returned with status. status==1 means scrolling is done. Double renders the characters at double size.

```
bitfire.init()
s=0
o=0
while s==0 do
        bitfire.clrscr()
        s,o = bitfire.scrolltext("lua",0,0,100,100,o,1)
        bitfire.flipbuf()
        sleep(1)
end
```

4.2.13 bitfire.charsinscroll(text,r,g,offset,double)

List scrolltext but the characters dance in a sinus pattern over the whole display.

4.2.14 bitfire.pixsinscroll(text,r,g,offset,double)

List scrolltext but the characters dance in a alternate sinus pattern over the whole display.

4.2.15 bitfire.pattern1(offset)

Render a simple pattern on the screen. You have to increment the offset.

4.2.16 bitfire.pattern2(offset)

Render a simple pattern on the screen. You have to increment the offset.

4.2.17 bitfire.pattern3(offset)

Render a simple pattern on the screen. You have to increment the offset.

4.2.18 bitfire.pattern4(offset)

Render a simple pattern on the screen. You have to increment the offset.

4.2.19 bitfire.starfield(num)

Render a full screen 3d starfield on the display with num number of stars.

4.2.20 bitifre.fire()

Render a full screen fire effect on the display.

4.2.21 bitfire.fire2()

Render a full screen alternate fire effect on the display.

4.2.22 bitfire.plasma()

Render a fill screen plasma effect on the display.

4.2.23 bitifre.mandel(brightness)

Render a mandelbrot fractal on the display. Brightness sets color offset.

4.2.24 bitfire.julia(brightness)

Render a julia fractal on the display. Brightness sets color offset.

4.2.25 bitfire.vector_load(num)

Loads a 3d object into the rendering engine. The objects are build into the API and num can be either 1 or 2.

4.2.26 bitfire.vector_render(zoom,r,g)

Render the 3d image on the display with color (r,g). Zoom sets zoom factor.

Last Page