SCIOPTA

High Performance
Real-Time Operating Systems

ARM
Tiny Flash File System for Atmel DataFlash™

User's Guide

Copyright

Disclaimer

Trademark

Headquarters

SCIOPTA Systems AG
Fiechthagstrasse 19
4103 Bottmingen
Switzerland
Tel. +41 61 423 10 62
Fax +41 61 423 10 63
email: sales@sciopta.com
www.sciopta.com

Document No. S08108RL1

Table of Contents

1 Introduction

1.1 SCIOPTA ARM Real-Time Operating System

SCIOPTA ARM is a high performance real-time operating system for microcontrollers using the ARM cores ARM7TDMI, ARM7TDMIS, ARM966E-S, ARM940T, ARM946E-S, ARM720T, ARM920T, ARM922T, ARM926E-JS and other derivatives of the ARM 7/9 family. Including:

Atmel AT91SAM
AT91SAM7S, AT91SAM7SE, AT91SAM7X, AT91SAM9 and all other ARM7/9 based microcontrollers.

NXP LPC2000
LPC21xx, LPC22xx, LPC212x, LPC23xx, LPC24xx, LPC28xx, LPC3180, and all other ARM7/9 based microcontrollers.

Sharp ARM7 and ARM9 MCU and SoC
LH754xx, LH795xx, LH7A4xx and all other ARM7/9 based microcontrollers.

STMicroelectronics STR7 and STR9
STR71x, STR72x, STR75x, STR91x and all other ARM7/9 based microcontrollers.

Including all microcontrollers from other suppliers which have ARM7/9 cores.

The full featured operating system environment includes:

- KRN - Pre-emptive Multi-Tasking Real-Time Kernel
- BSP - Board Support Packages
- IPS - Internet Protocols (TCP/IP)
- IPS Applications - Internet Protocols Applications (Web Server, TFTP, DNS, DHCP, Telnet, SMTP etc.)
- SFATFS - FAT File System
- SFFS - FLASH File System, NOR
- SFFSN - FLASH File System, NAND support
- USBD - Universal Serial Bus, Device
- USBH - Universal Serial Bus, Host
- DRUID - System Level Debugger
- SCIOPTA PEG - Embedded GUI
- CONNECTOR - support for distributed multi-CPU systems
- SMMS - Support for MMU
- SCAPI - SCIOPTA API on Windows or LINUX host
- SCSIM - SCIOPTA Simulator

1.2 About This Manual

This guide is intended for use by embedded software engineers who have knowledge of the C programming language, standard file API's who wish to implement a file system in Atmel DataFlash™.

Although every attempt has been made to make the system as simple to use as possible the developer must understand fully the requirements of the system they are designing to get the best practical benefit from the system.

In the reference part all specific file system messages and functions are listed.

Please consult also the SCIOPTA ARM - Kernel, User’s Guide.

1.3 SCIOPTA Tiny Flash File System for Atmel DataFlash™ Overview

There are many benefits to be gained by designing file systems specifically for the devices which are to be used for storage – these include large improvements in efficiency resulting in both resource (code, ram, CPU cycles) and power savings.

This file system is designed specifically for use with flash devices with small erase sectors. It has been highly crafted to ensure the reliability required by embedded systems together with their tight code and RAM requirements.

Some of the features include:

- 100% Failsafe
- Small ROM footprint (typically <6K for full read/write file system)
- Extremely small RAM usage (<100 bytes is possible)
- Highly scalable
- Supports most Atmel DataFlash™ devices
- Handles 10k writes/sector problem
- Standard file API
- Variable/Definable length file names
- Multiple simultaneous open files

1.4 Devices Supported

S-ARM-SFFSTDF is suitable for use with the following devices:

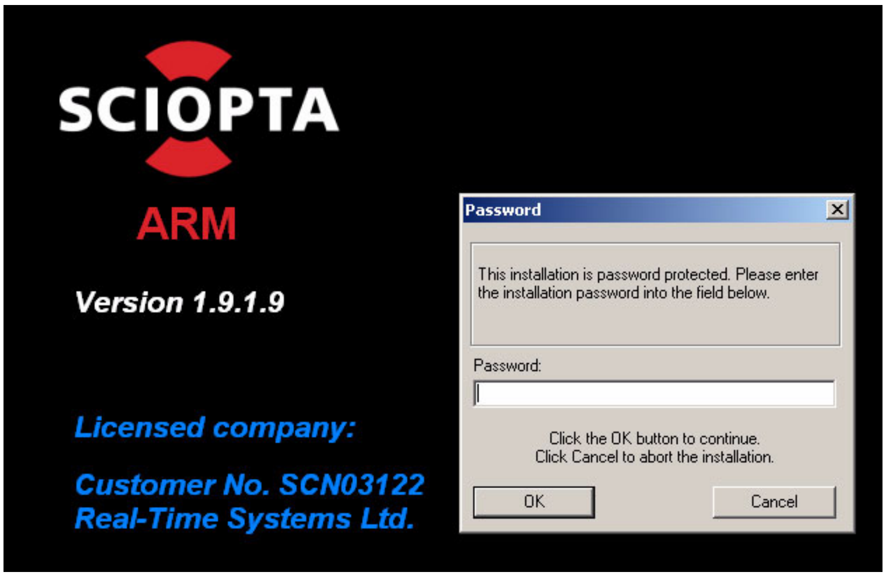
- AT45DB11B
- AT45DB21B
- AT45DB41B
- AT45DB81B
- AT45DB161B
- AT45DB321B
- AT45DB642B

Please contact SCIOPTA for any devices not in this list.

2 Installation

2.1 Introduction

Please consult the chapter "Installation" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description and guidelines of the SCIOPTA installation.



This installation is password protected. Please enter the installation password into the field below.

Password:

Click the OK button to continue.
Click Cancel to abort the installation.

Figure 2-1: Main Installation Window

3 Getting Started

3.1 Introduction

These are small tutorial examples which gives you a good introduction into typical SCIOPTA systems and products. They can be used as a starting point for more complex applications and your real projects.

Please Note:

The getting-started examples are using specific integrated development environments, build utilities, compilers, cpus and boards. If you are using another environment you may need to include other files from the delivery or modify the used files. Usually the following files are concerned: project file (system.c), linker script (<board_name.ld for GNU), BSP assembler files (led.S, resethook.S), BSP C files (druid_uart.c, fec.c, serial.c, simple_uart.c, systick.c) and C startup file (cstartup.S).

3.2 Getting Started

3.2.1 Description

In this getting started example we will create a SCIOPTA Tiny Flash File System for an Atmel DataFlash.

The example application consists of one file.

- The file tinyeffstest.c contains some user code to show file system functionality included in the process "SCP_effsTinyTest".

The user process is doing different file system calls to show the usage of the SCIOPTA Tiny Flash File System.

3.2.2 Tiny Flash File System for Atmel DataFlash™ Example Block Diagram

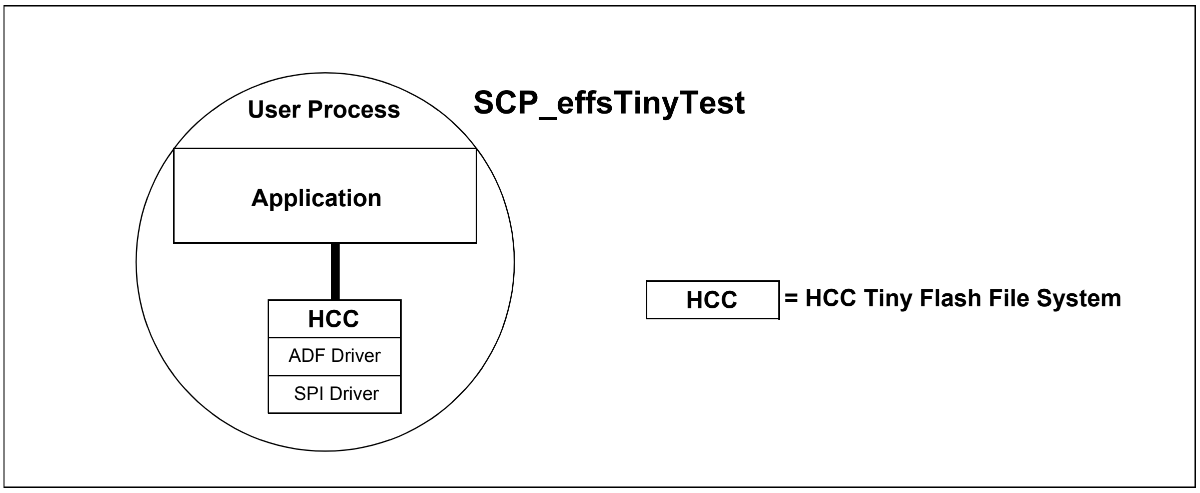


Figure 3-1: SCIOPTA Tiny Flash File System for Atmel DataFlash™ Block Diagram

3.2.3 Makefile and GNU GCC

3.2.3.1 Equipment

The following equipment is used to run this getting started example:

- Personal Computer or Workstation with: Intel® Pentium® processor, Microsoft® Windows XP, 256 MB of RAM and 200 MB of available hard disk space.
- GCC version 3.4.4, GNU Binutils version 2.16.1 and MSys Build Shell version 1.0.10. These products can be found on the separate SCIOPTA ARM tools CD delivery.
- A debugger/emulator for ARM which supports GNU GCC such as the iSYSTEM winIDEA Emulator/Debugger for ARM or the Lauterbach TRACE32 debugger for ARM.
- Target board which is included in the SCIOPTA examples. For each board there is a directory in the example folder: <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\. Actually supported are:
    - **AT91SAM7X-EK**
- SCIOPTA ARM - Kernel.
- SCIOPTA ARM - Tiny Flash File System for Atmel DataFlash™
- This getting started example is sending some specific log messages to a selected UART of the board. To display these messages on your host PC you can optionally connect a serial line from a COM port of your host PC to an UART port of your selected target board.
    - For the **AT91SAM7X-E** board COM1 is used.

  Please consult files system.c, simple_uart.c, simple_uart.h and config.h for selecting another UART or for another board.

3.2.3.2 Step-By-Step Tutorial

1. Check that the environment variable SCIOPTA_HOME is defined as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide.
2. Be sure that the GNU GCC compiler bin directory is included in the PATH environment variable as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide.
3. Be sure that the %SCIOPTA_HOME%\bin\win32 directory is included in the PATH environment variable as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide. This will give access to the GNU make utility.
4. Create an example working directory at a suitable place.
5. Copy the script **copy_files.bat** from the example directory for your selected target boards:

   <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\

   to your project folder.
6. Open a Windows Explorer or a Command Prompt window.
7. Go to the working directory of your project.
8. Type **copy_files.bat** to execute the batch file and the file copy process.
9. Launch the SCIOPTA configuration utility **SCONF** from the desktop or the Star menu.

10. Load the SCIOPTA example project file hello.xml from your project folder into **SCONF** (menu: **File** > **Open**).
11. Click on the **Build All** button or press **Ctrl-B** to build the kernel configuration files.

    The following files will be created in your project folder:

    - sciopta.cnf
    - sconf.c
    - sconf.h.
12. Execute the makefile for your selected target board:

    - For the **AT91SAM7X-EK** board: **gnu-make BOARD_SEL=9**

    The executable example file sciopta.elf is created.
13. Launch your ARM source-level emulator/debugger and load the resulting sciopta.elf.
14. If you have connected a serial line from the COM port of your host PC to the UART of your target board, open a terminal window on your PC and connect it to your selected PC COM port. The speed must be set to 115200 baud.
15. For some emulators/debuggers specific project and board initialization files can be found in the created project folder or in other example directories.
16. Now you can start the system and check the log messages on your host terminal window.
17. You can also set breakpoints anywhere in the example system and watch the behaviour.

3.2.4 Eclipse IDE and GNU GCC

3.2.4.1 Equipment

The following equipment is used to run this getting started example:

- Personal Computer or Workstation with: Intel® Pentium® processor, Microsoft® Windows XP, 256 MB of RAM and 200 MB of available hard disk space.
- GCC version 3.4.4, GNU Binutils version 2.16.1 and MSys Build Shell version 1.0.10. These products can be found on the separate SCIOPTA ARM tools CD delivery.
- The MSys make utility which is included in the \bin\win32 directory of the SCIOPTA installation.
- A debugger/emulator for ARM which supports GNU GCC such as the iSYSTEM winIDEA Emulator/Debugger for ARM or the Lauterbach TRACE32 debugger for ARM.
- Target board which is included in the SCIOPTA examples. For each board there is a directory in the example folder: <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\. Actually supported are:
    - **AT91SAM7X-EK**
- SCIOPTA ARM - Kernel.
- SCIOPTA ARM - Tiny Flash File System for Atmel DataFlash™
- Eclipse Version 3.3.1.1 (special distribution including SCIOPTA plug-ins), included on the separate SCIOPTA ARM tools CD.
- In order to run the Eclipse Platform you also need the Sun Java 2 SDK, Standard Edition for Microsoft Windows.
- This getting started example is sending some specific log messages to a selected UART of the board. To display these messages on your host PC you can optionally connect a serial line from a COM port of your host PC to an UART port of your selected target board.
    - For the **AT91SAM7X-E** board COM1 is used.

  Please consult files system.c, simple_uart.c, simple_uart.h and config.h for selecting another UART or for another board.

3.2.4.2 Step-By-Step Tutorial

1. Check that the environment variable SCIOPTA_HOME is defined as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide.
2. Be sure that the GNU GCC compiler bin directory is included in the PATH environment variable as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide.
3. Be sure that the %SCIOPTA_HOME%\bin\win32 directory is included in the PATH environment variable as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide. This will give access to the MSys make utility.
4. Create a project folder to hold all project files (e.g. d:\myprojects\sciopta) if you have not already done it for other getting-started projects.
5. Launch Eclipse. The Workspace Launcher window opens.
6. Select your created project folder (e.g. c:\myproject\sciopta) as your workspace (by using the Browse button).
7. Click the **OK** button. The workbench windows opens. Close the Welcome Tab.

8. Maximize the workbench and deselect **“Build Automatically”** in the **Project** menu.
9. Open the **New Project** window (menu: **File -> New -> C Project**).
10. Select arm-gcc ELF (Gnu)from the menu in the left window.
11. The New Project window opens. Enter the new project name: **sfs_tinyeffs** and click on the **Finish** button.
12. Select the new created project in the browser window and close the project (menu: **Project -> Close Project**).
13. The next steps we will execute outside Eclipse.
14. Copy the script **copy_files.bat** from the example directory for your selected target boards:

    <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\

    to your project folder.
15. Open a Command Prompt window.
16. Go to the example working directory of your project.
17. Type **copy_files.bat** to execute the batch file and the file copy process.
18. Launch the SCIOPTA configuration utility **SCONF** from the desktop or the Star menu.
19. Load the SCIOPTA example project file hello.xml from your project folder into **SCONF**.
    File > Open
20. Click on the **Build All** button or press **CTRL-B** to build the kernel configuration files.

    The following files will be created in your project folder:

    - sciopta.cnf
    - sconf.c
    - sconf.h
21. Swap back to the Eclipse workbench. Make sure that the kernel hello project (**sfs_tinyeffs**) is highlighted and open the project (menu: **Project > Open Project**).
22. Expand the project by selecting the **[+]** button and make sure that the **sfs_tinyeffs** project is highlighted.
23. Type the F5 key (or menu: **File > Refresh**).
24. Now you can see all files in the Eclipse Navigator window.
25. Select the Console window at the bottom of the Eclipse workbench to see the project building output.
26. Be sure that the project (**sfs_tinyeffs**) is high-lighted and build the project (menu: **Project > Build Project**).
27. The executable (sciopta.elf) will be created in the debug folder of the project.
28. Launch your ARM source-level emulator/debugger and load the resulting sciopta.elf.
29. If you have connected a serial line from the COM port of your host PC to the UART of your target board, open a terminal window on your PC and connect it to your selected PC COM port. The speed must be set to 115200 baud.
30. For some emulators/debuggers specific project and board initialization files can be found in the created project folder or in other example directories.
31. Now you can start the system and check the log messages on your host terminal window.
32. You can also set breakpoints anywhere in the example system and watch the behaviour.

3.2.5 iSYSTEM winIDEA and GNU GCC

3.2.5.1 Equipment

The following equipment is used to run this getting started example:

- Personal Computer or Workstation with: Intel® Pentium® processor, Microsoft® Windows XP, 256 MB of RAM and 200 MB of available hard disk space.
- GCC version 3.4.4, GNU Binutils version 2.16.1 and MSys Build Shell version 1.0.10. These products can be found on the separate SCIOPTA ARM tools CD delivery.
- Target board which is included in the SCIOPTA examples. For each board there is a directory in the example folder: <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\. Actually supported are:
    - **AT91SAM7X-EK**
- SCIOPTA ARM - Kernel.
- SCIOPTA ARM - Tiny Flash File System for Atmel DataFlash™
- iSYSTEM iC3000 - winIDEA Emulator/Debugger for ARM.
- This getting started example is sending some specific log messages to a selected UART of the board. To display these messages on your host PC you can optionally connect a serial line from a COM port of your host PC to an UART port of your selected target board.
    - For the **AT91SAM7X-EK** board COM1 is used.

  Please consult files system.c, simple_uart.c, simple_uart.h and config.h for selecting another UART or for another board.

3.2.5.2 Step-By-Step Tutorial

1. Check that the environment variable SCIOPTA_HOME is defined as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide.
2. Be sure that the GNU GCC compiler bin directory is included in the PATH environment variable as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide.
3. Create an example working directory at a suitable place.
4. Copy the script **copy_files.bat** from the example directory for your selected target boards:

   <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\
5. Open a Windows Explorer or a Command Prompt window.
6. Go to the working directory of your project.
7. Type **copy_files.bat** to execute the batch file and the file copy process.

8. Launch the SCIOPTA configuration utility **SCONF** from the desktop or the Star menu.
9. Load the SCIOPTA example project file hello.xml from your project folder into **SCONF**.
   File > **Open**
10. Click on the **Build All** button or press **CTRL-B** to build the kernel configuration files.

    The following files will be created in your project folder:

    - sciopta.cnf
    - sconf.c
    - sconf.h.
11. Launch the iSYSTEM iC3000 - winIDEA Emulator/Debugger for ARM.
12. Open the example workspace (menu: **File > Workspace > Open Workspace...**). Browse to your example project directory and select the workspace file iC3000.jrf.
13. Make the project (menu: **Project > Make**) or type the **F7** button.
14. The executable (sciopta.elf) will be created in the debug folder of the project.
15. Download the sciopta.elf file into the target system (menu: **Debug > Download**) or type the **Ctrl+F3** button.
16. If you have connected a serial line from the COM port of your host PC to the UART of your target board, open a terminal window on your PC and connect it to your selected PC COM port. The speed must be set to 115200 baud.
17. Run the system (menu: **Debug > Run**) or type the **F5** button.
18. Now you can check the log messages on your host terminal window.
19. You can also set breakpoints anywhere in the example system and watch the behaviour.

3.2.6 IAR Embedded Workbench

3.2.6.1 Equipment

The following equipment is used to run this getting started example:

- Personal Computer or Workstation with: Intel® Pentium® processor, Microsoft® Windows XP, 256 MB of RAM and 200 MB of available hard disk space.
- IAR Embedded Workbench for ARM including the following main components:
  - IAR Assembler for ARM 4.42A
  - IAR C/C++ Compiler for ARM 4.42A
  - IAR Embedded Workbench IDE 4.8
  - IAR XLINK 4.60I
  - IAR C-SPY including suitable target connection
- Target board which is included in the SCIOPTA examples. For each board there is a directory in the example folder: <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\. Actually supported are:
  - **AT91SAM7X-EK**
- SCIOPTA ARM - Kernel.
- SCIOPTA ARM - Tiny Flash File System for Atmel DataFlash™
- This getting started example is sending some specific log messages to a selected UART of the board. To display these messages on your host PC you can optionally connect a serial line from a COM port of your host PC to an UART port of your selected target board.
  - For the **AT91SAM7X-EK** board COM1 is used.

  Please consult files system.c, simple_uart.c, simple_uart.h and config.h for selecting another UART or for another board.

3.2.6.2 Step-By-Step Tutorial

1. Check that the environment variable SCIOPTA_HOME is defined as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide.
2. Be sure that the %SCIOPTA_HOME%\bin\win32 directory is included in the PATH environment variable as described in the chapter “Installation” of the SCIOPTA ARM - Kernel, User’s Guide. This will give access to the sconf.exe utility. Some IAREW examples might call sconf.exe directly from the workbench to do the SCIOPTA configuration.
3. Be sure that the IAR Embedded Workbench is installed as described in the IAREW user manuals.
4. Create an example working directory at a suitable place.
5. Copy the script **copy_files_iar.bat** from the example directory for your selected target boards:

   <install_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\
6. Open a Command Prompt window.
7. Go to the working directory of your project.
8. Type **copy_files_iar.bat** to execute the batch file and the file copy process.

9. Launch the SCIOPTA configuration utility **SCONF** from the desktop or the Star menu.
10. Load the SCIOPTA example project file hello.xml from your project folder into **SCONF**.
    File > **Open**
11. Click on the **Build All** button or press **CTRL-B** to build the kernel configuration files.

    The following files will be created in your project folder:

    - sciopta.cnf
    - sconf.c
    - sconf.h.
12. Launch the IAR Embedded Workbench.
13. Select the **Open existing workbench** button int the Embedded Workbench Startup window.
14. Browse to your example project directory and select the IAR Embedded Workbench file for your selected board: **<board>.eww**.
15. Select the project in the Workspace and Make the project (menu: **Project** > **Make**) or type the **F7** button.
16. The executable (sciopta.elf) will be created in the Output folder of the project.
17. Download and debug the sciopta.elf file into the target system (menu: **Project** > **Debug**) or type the **Ctrl+D** button.
18. If you have connected a serial line from the COM port of your host PC to the UART of your target board, open a terminal window on your PC and connect it to your selected PC COM port. The speed must be set to 115200 baud.
19. Run the system (menu: **Debug** > **Go**) or type the **Go** button.
20. Now you can check the log messages on your host terminal window.
21. You can also set breakpoints anywhere in the example system and watch the behaviour.

4 SCIOPTA Tiny Flash File System Project Overview

4.1 Introduction

This is an introduction into a typical SCIOPTA ARM Tiny Flash File System for Atmel DataFlash™ application. It follows a chronological order and for each step lists and describes the files needed from the SCIOPTA ARM delivery.

Only SCIOPTA Tiny Flash File System applications are covered. If you are using other SCIOPTA real-time products such as kernel, networking software, file systems, USB software, CONNECTOR packages for supporting distributed systems, memory management unit support software etc., please read the user manuals which are included in the delivery of these products for information how to use it.

4.2 SCIOPTA Techniques and Concepts

Before you can start writing any embedded applications using SCIOPTA you need become familiar with the concepts and techniques used by SCIOPTA.

Please read the chapter "SCIOPTA Technology and Methods" of the ARM - Kernel, User's Guide to get an introduction into the SCIOPTA technology.

In a new project you have first to determine the specification of the system. As you are designing a real-time system, speed requirements needs to be considered carefully including worst case scenarios. Defining function blocks, environment and interface modules will be another important part for system specification.

Systems design includes defining the modules, processes and messages. SCIOPTA is a message based real-time operating system therefore specific care needs to be taken to follow the design rules for such systems. Data should always be maintained in SCIOPTA messages and shared resources should be encapsulated within SCIOPTA processes.

4.3 SCIOPTA Tiny Flash File System Techniques and Concepts

Please consult chapter 5 "Tiny Flash File System Technology and Methods" on page 5-1 to get an introduction into the SCIOPTA Tiny Flash File System technology.

A SCIOPTA Tiny Flash File System application consists of specific user and system processes and file system specific messages and functions. Please consult chapter 5.3 "Tiny Flash File System Atmel DataFlash™ Block Diagram" on page 5-2 of an overview block diagram of a typical SCIOPTA Tiny Flash File System application.

4.4 SCIOPTA Kernel System Calls

To design SCIOPTA systems, modules and processes, to handle interprocess communication and to understand the included software of the SCIOPTA delivery you need to have detailed knowledge of the SCIOPTA application programming interface (API). The SCIOPTA API consist of a number of system calls to the SCIOPTA kernel to let the SCIOPTA kernel execute the needed functions.

Please consult the chapter "Application Programming Interface" of the ARM - Kernel User's Guide for a detailed description of all SCIOPTA system calls.

4.5 SCIOPTA Tiny Flash File System API

The SCIOPTA Tiny Flash File System for Atmel DataFlash™ uses and integrates the Embedded Flash File System Tiny For Atmel DataFlash™ from HCC Embedded. The HCC Tiny Flash File System functions are directly called from a SCIOPTA application process.

Please consult chapter 6 "HCC Tiny Flash File System API" on page 6-1 for a detailed description of the HCC API.

4.6 SCIOPTA Tiny Flash File System Files

In order to use the SCIOPTA Tiny Flash File System you need to include some HCC Tiny Flash File System source files in your project.

Please consult chapter 7.4.1 "HCC Tiny Flash File System" on page 7-3 for more information about the HCC Tiny File System files.

Files

dev.c	Serial number functions.
f_dir.c	Directory entry handling.
f_ext.c	File and directory changed detection.
f_file.c	File handling functions.
f_volume.c	Volume handling functions.
port.c	Time and date function.

File location: <installation_folder>\sciopta\<version>\sfs\hcc\effs-tiny\

4.7 Writing your Tiny File System Application

Now you are ready to write your SCIOPTA Tiny Flash File System application. You need to design your application and system which is divided into modules, processes and the interprocess communication and coordination.

When you are analysing a real-time system you need to partition a large and complex real-time system into smaller components and you need to design system structures and interfaces carefully. This will not only help in the analysing and design phase, it will also help you maintaining and upgrading the system.

In chapter 7 "Application Programming" on page 7-1 you will find information how to design a SCIOPTA Tiny Flash File System application.

For the "tinyeffs" SCIOPTA Tiny Flash File System for Atmel DataFlash™ getting started example the following file contains the HCC Mode Tiny Flash File System user process:

Files

tiny_effstest.c	Example tiny file system process

File location<installation_folder>\sciopta\<version>\exp\sfs\common\tinyeffs\

4.8 Error Handling

SCIOPTA uses a centralized mechanism for error reporting called Error Hooks. Error Hooks must be registered and written by the user. Please consult chapter 7.6 "Error Hook" on page 7-4 for more information about error hooks.

Files

error.c	Error hook example.

File location: <installation_folder>\sciopta\<version>\exp\common\

The error hook can for example be registered in the SCIOPTA start hook which contains early startup code. The start hook could for instance be placed in an example configuration file. In the SCIOPTA getting started examples the file **system.c** contains some system initialization code including the start hook and the error hook registration.

Please consult chapter 8.2.4 "Start Hook" on page 8-3 for more information.

4.9 System Configuration and Initialization

System and application configuration consists basically to write the start hook (see chapter 8.2.4 "Start Hook" on page 8-3), the system module hook (see chapter 8.2.6.1 "System Module Hook" on page 8-4) and other system initialization functions. This is usually done in a specific file called system.c which can be found in the SCIOPTA examples deliveries.

For the "tinyeffs" SCIOPTA Tiny Flash File System for Atmel DataFlash™ getting started example the file **system.c** contains the system and application configuration for the example system module:

Files

system.c	System configuration file including hooks and other setup code.

File location: <installation_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\

System and application configuration functions for other modules (user module hooks) is usually done in specific files having the same name as the module (**dev.c**, **ips.c** etc.) which can also be found in the SCIOPTA examples deliveries. Please consult also chapter 8.2.6.2 "User Modules Hooks" on page 8-4.

4.10 General System Functions and Drivers

System functions and drivers which do not depend on a specific board and a specific CPU and are common for SCIOPTA systems. Files for external systems, chips and controllers.

Source Files

f_atmel.c	Atmel DataFlash™ driver.

File location: <install_folder>\sciopta\<version>\sfs\hcc\effs-tiny\driver\

4.11 ARM System Functions and Drivers

For a basic system some general ARM CPU family functions and drivers which are not board and CPU-derivative dependent are needed.

Please consult chapter 9.3 “ARM System Functions and Drivers” on page 9-1 for more information about ARM functions and drivers.

Files

exception.S	Exception handler for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\gnu\

exception.s	Exception handler for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\arm\

exception.s79	Exception handler for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\iar\

4.12 ARM CPUs System Functions and Drivers

For a basic system some general CPU ARM Derivative CPU Family functions and drivers which are not board dependent are needed.

Please consult chapters

9.4 “AT91SAM7 System Functions and Drivers” on page 9-2
9.5 “AT91SAM9 System Functions and Drivers” on page 9-5
9.6 “LPC21xx System Functions and Drivers” on page 9-7
9.7 “LPC23xx and LPC24xx System Functions and Drivers” on page 9-9
9.8 “STR7 System Functions and Drivers” on page 9-11
9.9 “STR9 System Functions and Drivers” on page 9-13

Files

simple_uart.c	Simple polling uart function for printf debugging or logging.
spi_hcc.c	SPI driver for HCC Tiny Flash File System.
systick.c	System timer setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\src\

4.13 Board System Functions and Drivers

For your board you need to implement board functions and drivers. Usually at least a reset hook containing early board setup code is needed. Reset hooks are compiler manufacturer and board specific. Reset hook examples can be found in the SCIOPTA Board Support Package deliveries.

Please consult chapters

9.10 “Atmel AT91SAM7A3-EK Board” on page 9-15
9.11 “Atmel AT91SAM7SE-EK Board” on page 9-17
9.12 “Atmel AT91SAM7S-EK Board” on page 9-19
9.13 “Atmel AT91SAM7X-EK Board” on page 9-21
9.14 “Atmel AT91SAM9260-EK Board” on page 9-23
9.16 “Phytec phyCORE-LPC2294 Board” on page 9-27
9.18 “STR711-SK Board” on page 9-30
9.19 “STR912-SK Board” on page 9-32

Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\<board>\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\<board>\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\<board>\src\iar\

config.h	Board configuration.

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\<board>\include\

4.14 C/C++ Language Setup

To setup and initialize the C-environment you need to include C startup functions. C startup functions are compiler specific.

Please consult chapter 8.2.3 "C Startup" on page 8-3 for more information about C-startup.

For IAR Embedded Workbench there is no cstartup file needed as we are using the IAR C initialization library function.

Source File

cstartup.S	C startup assembler source for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\gnu\

cstartup.s	C startup assembler source for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\arm\

4.15 Kernel Configuration

To build your system properly you need to configure the target system, modules, processes and pools. This is done with the SCIOPTA configuration utility **SCONF**.

Please consult chapter 10 "Kernel Configuration" on page 10-1 for more information about kernel configuration.

Example SCIOPTA configuration files hello.xml which can be loaded in the **SCONF** configuration program are included in the SCIOPTA delivery.

For the "tinyeffs" SCIOPTA Tiny Flash File System getting started example the configuration file is delivered in the example projects delivery:

SCONF Configuration File

hello.xml	SCIOPTA kernel configuration file.

File location: <installation_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\

After your configuration is correct the **SCONF** utility will generate three files which need to be included into your SCIOPTA project.

Generated Kernel Configuration Files

sciopta.cnf	This is the configured part of the kernel which will be included when the SCIOPTA kernel is assembled.
sconf.h	This is a header file which contains some configuration settings.
sconf.c	This is a C source file which contains the system initialization code.

4.16 Linker Scripts and Memory Map

Please consult the chapter "Linker Script and Memory Map" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of linker scripts.

A link is usually controlled by a linker script or linker control file. Linker scripts are compiler and board specific. Linker script examples can be found in the SCIOPTA Board Support Package and/or example deliveries. Please consult also chapter 9 "Board Support Packages" on page 9-1 for more information.

Files

<board>.ld	Linker script for GNU GCC.
<board>.sct	Linker script for ARM RealView.
<board>.xcl	Linker script for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\<board>\include\
or: <installation_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\

module.ld	Linker script: Module sections (common to all boards) for GNU GCC

File location: <install_folder>\sciopta\<version>\bsp\arm\include\

For IAR you need to define the free RAM of the modules in a separate file. In this area there are no initialized data. Module Control Block (ModuleCB), Process Control Blocks (PCBs), Stacks and Message Pools are placed in this free RAM.

Source File

map.c	Module mapping definitions for IAR.

File location: <installation_folder>\sciopta\<version>\exp\krn\<arm>\<cpu>\<board>\

4.17 Kernel

The SCIOPTA kernel is provided in assembler source file and therefore compiler manufacturer specific. The kernels can be found in the library directory of the SCIOPTA delivery.

Source File

sciopta.S	Kernel source file for GNU GCC.
sciopta.s	Kernel source file for ARM RealView.
sciopta.s79	Kernel source file for IAR EW.

File location: <install_folder>\sciopta\<version>\lib\arm\krn\

4.18 Libraries and Include Files

Make sure that you have included the kernel libraries as described in chapter 11.2 “Kernel Libraries” on page 11-1 and the SCIOPTA file system library as described in chapter 11.3 “SCIOPTA Tiny File System Library” on page 11-3.

Make sure that the include search directories are defined as described in chapter 11.4 “Include Files” on page 11-4. Please consult this chapter for more information about SCIOPTA includes.

4.19 Building the Project

4.19.1 Makefiles

The most flexible and direct way to build a SCIOPTA system is to use makefiles. Makefiles for the getting started projects are included in the delivery.

Please consult a getting started example (e.g. chapter 3.2.3 “Makefile and GNU GCC” on page 3-2) for a description how to build a system with makefiles.

For the “tinyeffs” SCIOPTA Tiny Flash File System getting started example the makefiles are delivered in the example projects delivery:

Files

Makefile	Example makefile.

File location: <installation_folder>\sciopta\<version>\exp\krn\arm\tinyeffs\

board.mk	Board dependent makefiles.

File location: <installation_folder>\sciopta\<version>\exp\krn\arm\tinyeffs\<board>\

4.19.2 Eclipse

SCIOPTA supplies Eclipse project files if you want to use Eclipse as an IDE for editing and building SCIOPTA applications.

Please consult a getting started example (e.g. chapter 3.2.4 “Eclipse IDE and GNU GCC” on page 3-4) for a description how to build a system with Eclipse.

For the “tinyeffs” SCIOPTA Tiny Flash File System getting started example the Eclipse project file can be found in the example projects delivery:

Files

.cproject	Eclipse project file
.settings	Eclipse preferences file which is needed to have *.S file types supported.

File location: <installation_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\

4.19.3 iSYSTEM winIDEA

If winIDEA and the iSYSTEM emulator/debugger is your preferred build and debug environment you will find winIDEA project files in the SCIOPTA examples deliveries.

Please consult a getting started example (e.g. chapter 3.2.5 "iSYSTEM winIDEA and GNU GCC" on page 3-6) for a description how to build a system with iSYSTEM winIDEA.

For the "tinyeffs" SCIOPTA Tiny Flash File System getting started example the iSYSTEM winIDEA project file can be found in the example projects delivery:

Files

iC3000.xqrf	iSYSTEM winIDEA project file.
iC3000.xjrf	iSYSTEM winIDEA project file.

File location: <installation_folder>\sciopta\<version>\exp\sfs\arm\tineeffs\<board>

winIDEA_gnu.ind	iSYSTEM winIDEA indirection file.

File location: <installation_folder>\sciopta\<version>\bsp\common\include\

<board>.ini	iSYSTEM winIDEA board initialization file.

File location: <installation_folder>\sciopta\<version>\bsp\arm\<cpu>\<board>\include\<board>.ini

4.19.4 IAR Embedded Workbench

If the IAR Embedded Workbench (EW) is your preferred build and debug environment you will find IAR EW project files in the SCIOPTA examples deliveries.

Please consult a getting started example (e.g. chapter 3.2.6 "IAR Embedded Workbench" on page 3-8) for a description how to build a system with IAR EW.

For the "tinyeffs" SCIOPTA Tiny Flash File System getting started example the IAR EW project file can be found in the example projects delivery:

Files

<board>.eww	IAR Embedded Workbench project file.

File location: <installation_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>

<board>.mac	IAR Embedded Workbench board initialization file.

File location: <installation_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>

5 Tiny Flash File System Technology and Methods

5.1 SCIOPTA Tiny Flash File System for Atmel DataFlash™ Overview

There are many benefits to be gained by designing file systems specifically for the devices which are to be used for storage – these include large improvements in efficiency resulting in both resource (code, ram, CPU cycles) and power savings.

This file system is designed specifically for use with flash devices with small erase sectors. It has been highly crafted to ensure the reliability required by embedded systems together with their tight code and RAM requirements.

Some of the features include:

- 100% Failsafe
- Small ROM footprint (typically <6K for full read/write file system)
- Extremely small RAM usage (<100 bytes is possible)
- Highly scalable
- Supports most Atmel DataFlash™ devices
- Handles 10k writes/sector problem
- Standard file API
- Variable/Definable length file names
- Multiple simultaneous open files

5.2 Devices Supported

S-ARM-SFFSTDF is suitable for use with the following devices:

- AT45DB11B
- AT45DB21B
- AT45DB41B
- AT45DB81B
- AT45DB161B
- AT45DB321B
- AT45DB642B

Please contact SCIOPTA for any devices not in this list.

5.3 Tiny Flash File System Atmel DataFlash™ Block Diagram

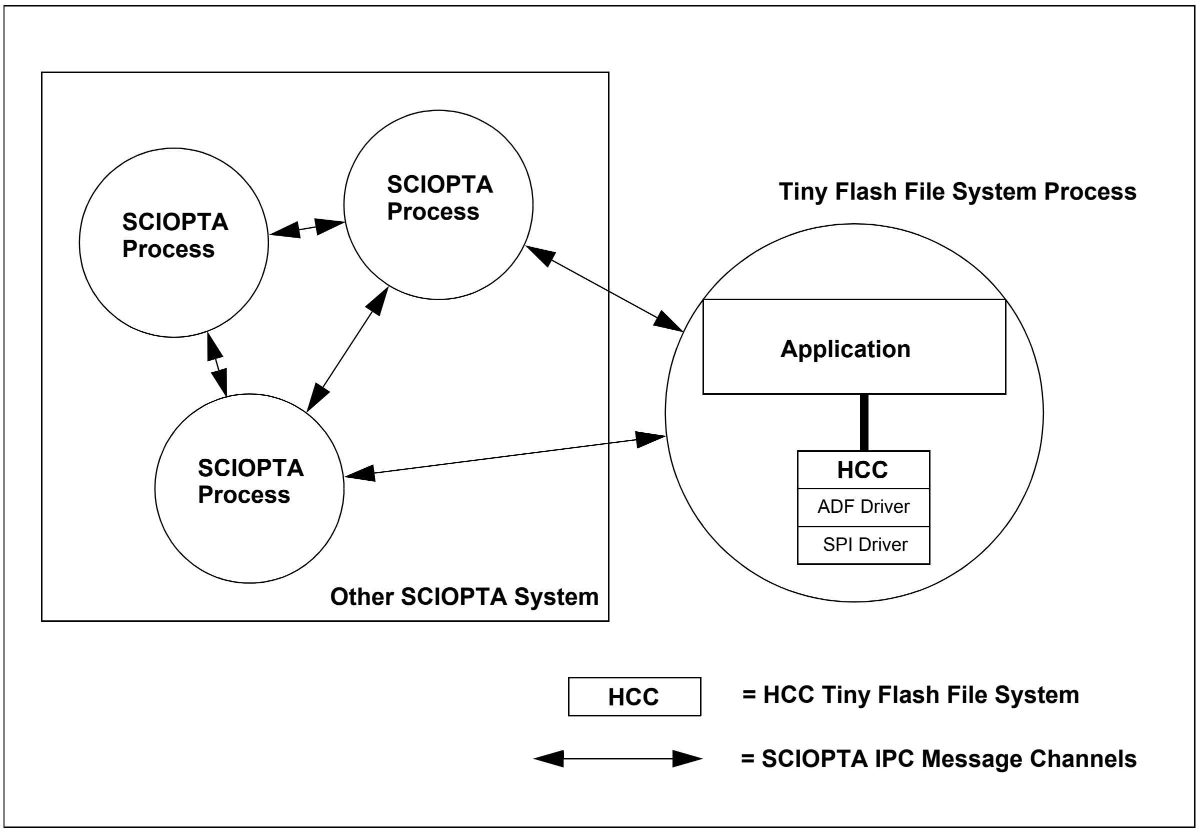


Figure 5-1: SCIOPTA Tiny Flash File System for Atmel DataFlash™ Block Diagram

5.4 HCC Tiny Flash File System

The SCIOPTA Tiny Flash File System for Atmel DataFlash™ uses and integrates the Tiny Flash File System from HCC Embedded.

The following diagram illustrates the structure of the tiny file system software.

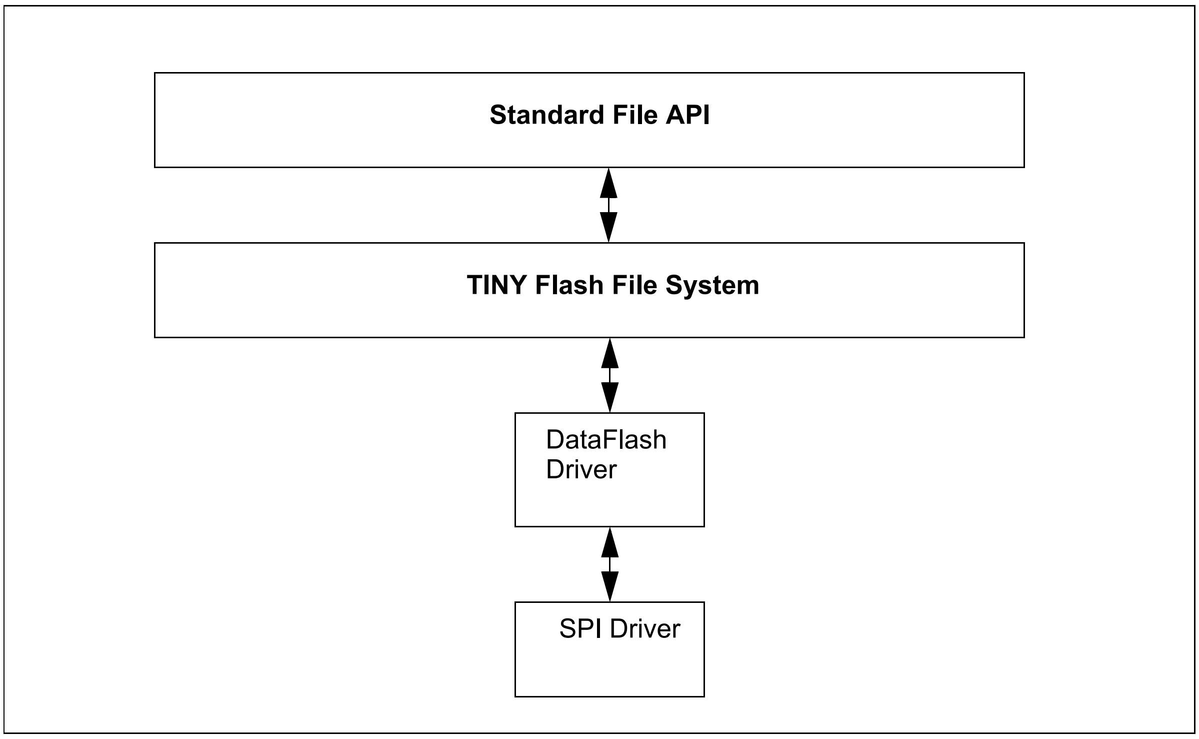


Figure 5-2: HCC EFFS DF Structure

5.5 Processes

5.5.1 Tiny File System User Process

The HCC Tiny Flash File System functions are called directly by the user. The HCC Tiny Flash File System runs in the context of the user process.

The user process contains the user code to access the file system. Contrary to the standard SCIOPTA Tiny Flash File System, there is only this direct way (HCC Mode) of using the SCIOPTA Tiny Flash File System.

5.6 About Atmel DataFlash™

These devices are ideal for data storage but require special handling for reliable handling.

5.6.1 Erase/Writes per Sector

The devices specify that if 10,000 erase/writes are performed on a sector then during this time all pages in that sector must be rewritten to ensure the integrity of the flash. This file system is designed to ensure that this criterion is met with the absolute minimum systems overhead and also ensures that where a page must be rewritten this is done in a completely failsafe way; i.e. such that when the page is erased there is a saved copy of it so that in the event of power loss during this process the system will not lose the data.

5.6.2 RAM Buffers

The devices contain page buffers in internal RAM. When modifying the contents of a page the Ram buffer is modified and when the command to commit this page is given the page is erased and then the data is written. If a powerfail were to occur during this process any or all of the data in the page can be lost regardless of whether it was modified. The Tiny Flash File Systems employs algorithms to ensure that where critical data is being altered it is done in a failsafe way.

5.6.3 Wear

Some of the device datasheets specify a limit of 100,000 erase/write cycles per page on the device. The Tiny Flash File System uses various techniques to ensure that the usage if pages is distributed over the entire device.

It is specifically designed to handle all these issues with a minimum of overhead. In typical usage this overhead is close to zero but in worst case scenarios amounts to about a 15% overhead on writes to the device.

6 HCC Tiny Flash File System API

The SCIOPTA Tiny Flash File System for Atmel DataFlash™ uses and integrates the Embedded Flash File System Tiny For Atmel DataFlash™ from HCC Embedded. The user can directly access the HCC Tiny Flash File System API (HCC-Mode).

The following diagram illustrates the structure of the file system software.

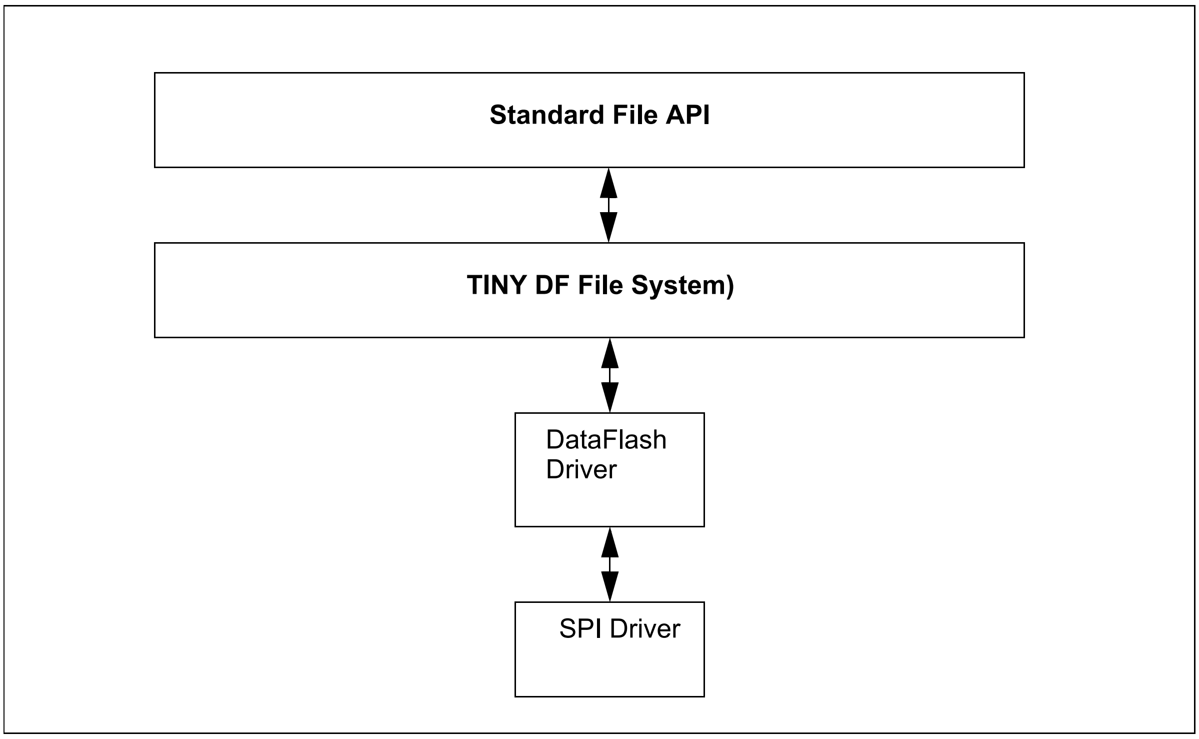


Figure 6-1: HCC EFFS DF Structure

6.1 HCC Tiny File System Functions List

In alphabetical order.

Function	Short Description	Page
f_chdir	Change current working directory.	6-8
f_close	Close a previously opened file.	6-19
f_delete	Delete a file.	6-12
f_eof	Check whether the current position in the opened target file is the end of the file.	6-24
f_filelength	Get the length of a file.	6-13
f_findfirst	Find first file or subdirectory in specified directory.	6-14
f_findnext	Find the next file or subdirectory in a specified directory after a previous call to f_findfirst or f_findnext.	6-15
f_format	Format a drive.	6-5
f_getc	Read a character from the current position in the open target file.	6-27
f_getcwd	Get current working folder on current drive.	6-10
f_getfreespace	Fill a user allocated structure with information about the usage of the volume specified.	6-6
f_gettimedate	Get time and date information from a file or directory.	6-16
f_getversion	Retrieve file system version information.	6-3
f_initvolume	Initialize the file system	6-4
f_mkdir	Make a new directory.	6-7
f_open	Opens a file.	6-17
f_putc	Write a character to the open target file at the current file position.	6-26
f_read	Read data from the current file position.	6-21
f_rename	Rename a file or directory.	6-11
f_rewind	Set the current file position in the open target file to the beginning.	6-25
f_rmdir	Remove directory.	6-9
f_seek	Move read/write position in the file.	6-22
f_tell	Tell the current file position in the target file.	6-23

6.2 Tiny Flash File System Functions

In functional order.

6.2.1 f_getversion

This function is used to retrieve file system version information.

```
char * f_getversion(void)
```

Parameter	Description
None	

Return Value	Condition
Pointer to null terminated ASCII string	

Example

```
void display_fs_version(void)
{
   printf("File System Version: %s",f_getversion());
}
```

6.2.2 f_initvolume

This function is used to initialize the volume. This function should be called every time the file system is started.

```
unsigned char f_initvolume(void)
```

Parameter	Description
None	

Return Value	Condition
F_NO_ERROR	Drive successfully initialized.
Error Code	Error condition.

Example

```
void myinitfs() {
   unsigned char ret;
      /* Initialize Drive */
   ret=f_initvolume();
   if(ret)
      printf("Drive init error %d\n",ret);
         .
         .
}
```

6.2.3 f_format

Format the drive. If the media is not present this routine will fail. If successful all data on the specified volume will be destroyed. Any open files will be closed.

```
unsigned char f_format(void)
```

Parameter	Description
none	

Return Value	Condition
F_NO_ERROR	Drive successfully formatted.
Error Code	Error condition.

Example

```
void myinitfs() {
   unsigned char ret;
   f_initvolume();
   ret=f_format();
   if(ret)
      printf("Format Failed: Error %d",ret);
   else
      printf("Drive formatted Correctly");
         .
         .
}
```

6.2.4 f_getfreespace

This function fills a structure with information about the drive space usage - total space and free space.

```
unsigned char f_getfreespace(
                    F_SPACE    *pSpace
)
```

Parameter	Description
pSpace	Pointer to user’s free space structure

Return Value	Condition
F_NO_ERROR	Success.
Error Code	Error condition.

Example

```
void info() {
   F_SPACE space;
   unsigned char ret;
      /* get free space on current drive */
   ret = f_getfreespace(space);
   if(!ret)
      printf("\nThere are %d bytes total, \
                     %d bytes free.",\
                     space.total, space.free);
   else
      printf("\nError %d reading drive\n", ret);
}
```

6.2.5 f_mkdir

Make a new directory.

```
unsigned char f_mkdir(
              const char *dirname
)
```

Parameter	Description
dirname	New directory name to create.

Return Value	Condition
F_NO_ERROR	New directory name created successfully.
Error Code	Error condition.

Example

```
void myfunc() {
        .
        .
   f_mkdir("subfolder"); /*creating directory*/
   f_mkdir("subfolder/sub1");
   f_mkdir("subfolder/sub2");
   f_mkdir("/subfolder/sub3"
        .
        .
}
```

6.2.6 f_chdir

Change current working directory.

```
unsigned char f_chdir(
              const char    *dirname
)
```

Parameter	Description
dirname	New directory name to change.

Return Value	Condition
F_NO_ERROR	Directory has been changed successfully.
Error Code	Error condition.

Example

```
void myfunc(void)
{
      .
      .
   f_mkdir(“subfolder”);
   f_chdir(“subfolder”);          /* change directory */
   f_mkdir(“sub2”);
   f_chdir(“..”);                 /* go to upward */
   f_chdir(“subfolder/sub2”);     /* goto into sub2 dir */
      .
      .
}
```

6.2.7 f_rmdir

Remove directory. If the target directory not empty or it is read-only then this returns with an error.

```
unsigned char f_rmdir(
              const char    *dirname
)
```

Parameter	Description
dirname	Directory name to remove.

Return Value	Condition
F_NO_ERROR	Directory is removed successfully.
Error Code	Error condition.

Example

```
void myfunc(void)
{
      .
      .
   f_mkdir(“subfolder”);          /* creating directories */
   f_mkdir(“subfolder/sub1”);
      .
      . doing some work
      .
   f_rmdir(“subfolder/sub1”);
   f_rmdir(“subfolder”);          /* removes directory */
      .
      .
}
```

6.2.8 f_getcwd

Get current working folder.

```
unsigned char f_getcwd(
              char    *buffer,
              int     maxlen
)
```

Parameter	Description
buffer	Where to store current working directory string.
maxlen	Length of the buffer.

Return Value	Condition
F_NO_ERROR	Success.
Error Code	Error Condition

Example

```
#define BUFFLEN 256

void myfunc() {
   char buffer[BUFFLEN];
   unsigned char ret;
   ret = f_getcwd(buffer, BUFFLEN);
   if (!ret)
      printf (“current directory is %s”,buffer);
   else
      printf (“Error %d”, ret)
}
```

6.2.9 f_rename

Rename a file or directory.

If a file or directory is read only it cannot be renamed. If a file is open it cannot be renamed.

```
unsigned char f_rename(
              const char   *filename,
              const char   *newname
)
```

Parameter	Description
filename	File or directory name with/without path.
newname	New name of file or directory.

Return Value	Condition
F_NO_ERROR	Success.
Error code	Error condition.

Example

```
void myfunc(void)
{
      .
      .
   f_rename (“oldfile.txt”,”newfile.txt”);
   f_rename (“A:/subdir/oldfile.txt”,”newfile.txt”);
      .
      .
}
```

6.2.10 f_delete

Delete a file. A read-only or open file cannot be deleted.

```
unsigned char f_delete(
             const char    *filename
)
```

Parameter	Description
filename	File name with/without path to be deleted

Return Value	Condition
F_NO_ERROR	Success.
Error code	Error condition.

Example

```
void myfunc(void)
{
      .
      .
   f_delete ("oldfile.txt");
   f_delete ("A:/subdir/oldfile.txt");
      .
      .
}
```

6.2.11 f_filelength

Get the length of a file. If file does not exist this function returns with zero.

```
unsigned long f_filelength(
                char        *filename
)
```

Parameter	Description
filename	File name with/without path

Return Value	Condition
Length of file	

Example

```
int myreadfunc(char *filename, char *buffer, long buffsize)
{
   F_FILE *file=f_open(filename,"r");
   long size=f_filelength(filename);

   if (!file)
   {
      printf ("%s Cannot be opened!",filename);
      return 1;
   }

   if (size>buffsize)
   {
      printf ("Not enough memory!");
      return 2;
   }

   f_read(buffer,size,1,file);
   f_close(file);

   return 0;
}
```

6.2.12 f_findfirst

Find first file or subdirectory in specified directory. First call **f_findfirst** function and if file was found get the next file with **f_findnext** function. Files with the system attribute set will be ignored.

```
unsigned char f_findfirst(
                const char    *filename,
                F_FIND        *find
)
```

Parameter	Description
filename	Name of file to find.
find	Where to store find information

Return Value	Condition
F_NO_ERROR	Success.
Error Code	Error condition.

Example

```
void mydir(void)
{
   F_FIND find;

   if (!f_findfirst("A:/subdir/*.*",&find))
   {
      do
      {
         printf (“filename:%s”,find.filename);
       if (find.attr&F_ATTR_DIR)
         {
            printf (“ directory\n”);
          }
          else
         {
            printf (“ size %d\n”,find.len);
       }
      } while (!f_findnext(&find));
   }
}
```

6.2.13 f_findnext

Find the next file or subdirectory in a specified directory after a previous call to **f_findfirst** or **f_findnext**. First call **f_findfirst**; if file was found get the rest of the matching files by repeated calls to **f_findnext**. Files with the system attribute set will be ignored.

```
unsigned char f_findnext(
              F_FIND        *find
)
```

Parameter	Description
find	Find structure (from f_findfirst).

Return Value	Condition
F_NO_ERROR	Success.
Error Code	Error condition.

Example

```
void mydir(void)
{
   F_FIND find;
   if (!f_findfirst("A:/subdir/*.*",&find))
   {
      do
      {
         printf ("filename:%s",find.filename);
         if (find.attr&F_ATTR_DIR)
         {
            printf (" directory\n");
          }
          else
         {
            printf (" size %d\n",find.len);
      }
      } while (!f_findnext(&find));
   }
}
```

6.2.14 f_gettimedate

Get time and date information from a file or directory.

```
unsigned char f_gettimedate(
             const char      *filename,
             unsigned short  *pctime,
             unsigned short  *pcdate)
```

Parameter	Description
filename	Target file or directory.
pctime	Pointer where to store the time.
pcdate	Pointer where to store the date.

Return Value	Condition
F_NO_ERROR	Success.
Error code	Error condition.

Example

```
void myfunc(void)
{
  unsigned short t,d;
  if (!f_gettimedate("subfolder",&t,&d))
  {
     unsigned short sec=(t & 0x001f) << 1;
     unsigned short minute=((t & 0x07e0) >> 5);
     unsigned short hour=((t & 0x0f800) >> 11);
     unsigned short day= (d & 0x001f);
     unsigned short month= ((d & 0x01e0) >> 5);
     unsigned short year=1980+((d & 0xfe00) >> 9);
     printf ("Time: %d:%d:%d",hour,minute,sec);
     printf ("Date: %d.%d.%d",year,month,day);
  }
  else
  {
     printf ("File time cannot retrieved!"
  }
}
```

6.2.15 f_open

Opens a file.

```
F_FILE *f_open(
             const char     *filename,
             const char     *mode
)
```

Parameter	Description	
filename	File to be opened.	
mode	Pointer to open mode string.	
	Mode	**Description**
	“r”	Open an existing file for reading. The stream is positioned at the beginning of the file.
	“r+”	Open an existing file for reading and writing. The stream is positioned at the beginning of the file.
	“w”	Truncate file to zero length or create file for writing. The stream is positioned at the beginning of the file.
	“w+”	Open for reading and writing. The file is created if it does not exist; otherwise it is truncated. The stream is positioned at the beginning of the file.
	“a”	Open for appending (writing at end of file). The stream is positioned at the end of the file.
	“a+”	Open for reading and appending (writing at end of file). The stream is positioned at the end of the file.

Return Value	Condition
Pointer to the associated opened file handle.	File successfully opened.
0	File could not be opened.

Example

```
void myfunc(void)
{
   F_FILE *file;
   char c;

   file=f_open("myfile.bin","r");
   if (!file)
   {
      printf ("File cannot be opened!");
      return;
   }

   f_read(&c,1,1,file); /* read 1byte */
   printf ("'%c' is read from file",c);
   f_close(file);
}
```

6.2.16 f_close

Closes a previously opened file.

```
unsigned char f_close(
             F_FILE       *filehandle
)
```

Parameter	Description
filehandle	File handle of target.

Return Value	Condition
F_NO_ERROR	Success.
Error Code	Error condition.

Example

```
void myfunc(void)
{
   F_FILE *file;
   char *string="ABC";

   file=f_open("myfile.bin","w");

   if (!file)
   {
      printf ("File cannot be opened!");
      return;
   }

   f_write(string,3,1,file); /* write 3byte */
   if (!f_close(file))
   {
      printf ("File stored");
   }
   else printf ("file close error");
}
```

6.2.17 f_write

Write data into file at current position. File has to be opened with "r+", “w”, “w+”, "a+" or “a”. The file pointer is moved forward by the number of bytes successfully written.

```
long f_write(
             const void   *buf,
             long         size,
             long         size_t,
             F_FILE       *filehandle
)
```

Parameter	Description
buf	Pointer to data buffer.
size	Size of items to be written.
size_t	Number of items to be written.
filehandle	File handle to write to.

Return Value	Condition
Number of items written	

Example

```
void myfunc(void)
{
   F_FILE *file;
   char *string=”ABC”;

   file=f_open(“myfile.bin”,”w”);
   if (!file)
   {
      printf (“File cannot be opened!”);
      return;
   }
   if (f_write(string,1,3,file)!=3)
   {  /* write 3bytes */
      printf (“not all items written”);
   }
   f_close(file);
}
```

6.2.18 f_read

Read data from the current file position. File has to be opened with “r”, "r+", "w+" or "a+". The file pointer is moved forward by the number of bytes read.

```
long f_read(
             void          *buf,
             long          size,
             long          size_t,
             F_FILE        *filehandle
)
```

Parameter	Description
buf	Pointer to data buffer.
size	Size of each of items to be read.
size_t	Number of items to be read.
filehandle	File handle to read from.

Return Value	Condition
Number of bytes read.	

Example

```
int myreadfunc(char *filename, char *buffer, long buffsize)
{
   F_FILE *file=f_open(filename,”r”);

   long size=f_filelength(filename);
   if (!file)
   {
      printf (“%s Cannot be opened!”,filename);
      return 1;
   }
   if (f_read(buffer,1,size,file)!=size)
   {
      printf (“not all items read”);
   }
   f_close(file);
   return 0;
}
```

6.2.19 f_seek

Move read/write position in the file.

```
int f_read(
             F_FILE          *filehandle,
             long            offset,
             unsigned char   whence
)
```

Parameter	Description	
filehandle	File handle to read from.	
offset	Relative byte position according to whence	
whence	Where to calculate offset from. Whence parameter could be one of:	
	F_SEEK_CUR	Current position of file pointer.
	F_SEEK_END	End of file.
	F_SEEK_SET	Beginning of file

Return Value	Condition
F_NO_ERROR	Success.
Error code	Error condition.

Example

```
int myreadfunc(char *filename, char *buffer, long buffsize)
{
   F_FILE *file=f_open(filename,"r");
   f_read(buffer,1,1,file);     /* read 1 byte */
   f_seek(file,0,F_SEEK_SET);
   f_read(buffer,1,1,file);     /*read the same 1 byte */
   f_seek(file,-1,F_SEEK_END);
   f_read(buffer,1,1,file);     /* read last 1 byte */
   f_close(file);
   return 0;
}
```

6.2.20 f_tell

Tell the current file position in the target file.

```
long f_tell(
             F_FILE       *filehandle
)
```

Parameter	Description
filehandle	File handle of target.

Return Value	Condition
Current read or write file position	

Example

```
int myreadfunc(char *filename, char *buffer, long buffsize)
{
   F_FILE *file=f_open(filename,"r");

   printf ("Current position %d",f_tell(file));
   f_read(buffer,1,1,file); /* read 1 byte */
   printf ("Current position %d",f_tell(file));
   f_read(buffer,1,1,file); /* read 1 byte */
   printf ("Current position %d",f_tell(file));

   f_close(file);
   return 0;
}
```

6.2.21 f_eof

Check whether the current position in the opened target file is the end of the file.

```
unsignet char f_eof(
              F_FILE        *filehandle
)
```

Parameter	Description
filehandle	File handle of target.

Return Value	Condition
0	Not at end of file.
!=0	End of file or any error.

Example

```
int myreadfunc(char *filename, char *buffer, long buffsize)
{
   F_FILE *file=f_open(filename,"r");

   while (!f_eof())
   {
      if (!buffsize) break;
      buffsize--;
      f_read(buffer++,1,1,file);
   }

   f_close(file);
   return 0;
}
```

6.2.22 f_rewind

Set the current file position in the open target file to the beginning.

```
unsigned char f_rewind(
             F_FILE       *filehandle
)
```

Parameter	Description
filehandle	File handle of target.

Return Value	Condition
0	Success.
!=0	Error condition.

Example

```
void myfunc(void)
{
   char buffer[4];
   char buffer2[4];

   F_FILE *file=f_open("myfile.bin",”r”);

   if (file)
   {
      f_read(buffer,4,1,file);
      f_rewind(file);            /* rewind file pointer */
      f_read(buffer2,4,1,file);
                                 /* read from beginning */
      f_close(file);
   }
   return 0;
}
```

6.2.23 f_putc

Write a character to the open target file at the current file position. The current file position is incremented.

```
int f_putc(
             int          ch,
             F_FILE       *filehandle
)
```

Parameter	Description
ch	Character to be written
filehandle	File handle of target.

Return Value	Condition
Written character	Success.
-1	Write Failed.

Example

```
void myfunc (char *filename, long num)
{
   F_FILE *file=f_open(filename,"w");

   while (num--)
   {
      int ch='A';
      if(ch!=(f_putc('ch', file))
      {
         printf("Error!!!");
         break;
      }
   }
   f_close(file);
   return 0;
}
```

6.2.24 f_getc

Read a character from the current position in the open target file. The current file position will be incremented.

```
int f_getc(
             F_FILE       *filehandle
)
```

Parameter	Description
filehandle	File handle of target.

Return Value	Condition
Character read from file	Success.
-1	Read Failed.

Example

```
int myreadfunc(char *filename, char *buffer, long buffsize)
{
   F_FILE *file=f_open(filename,”r”);
   while (buffsize--)
   {
      int ch;
      if((ch=f_getc(file))== -1)
         break;
      *buffer++=(char)ch;
      buffsize--;
   }

   f_close(file);
   return 0;
}
```

7 Application Programming

7.1 Introduction

System design includes all phase from system analysis, through specification to system design, coding and testing. In this chapter we will give you some useful information of what methods, techniques and structures are available in SCIOPTA to fulfil your real-time requirements for your embedded system.

7.2 System Partition

When you are analysing a real-time system you need to partition a large and complex real-time system into smaller components and you need to design system structures and interfaces carefully. This will not only help in the analysing and design phase, it will also help you maintaining and upgrading the system.

In a SCIOPTA controlled real-time system you have different structure elements available to decompose the whole system into smaller parts.

Please consult the chapter "Application Programming" of the ARM - Kernel, User's Guide for more information how to use

- modules
- processes
- interprocess communications
- messages
- message pools
- triggers
- daemons
- trap interface
- error handling
- interrupt handling.

7.3 Tiny Flash File System Atmel DataFlash™ Block Diagram

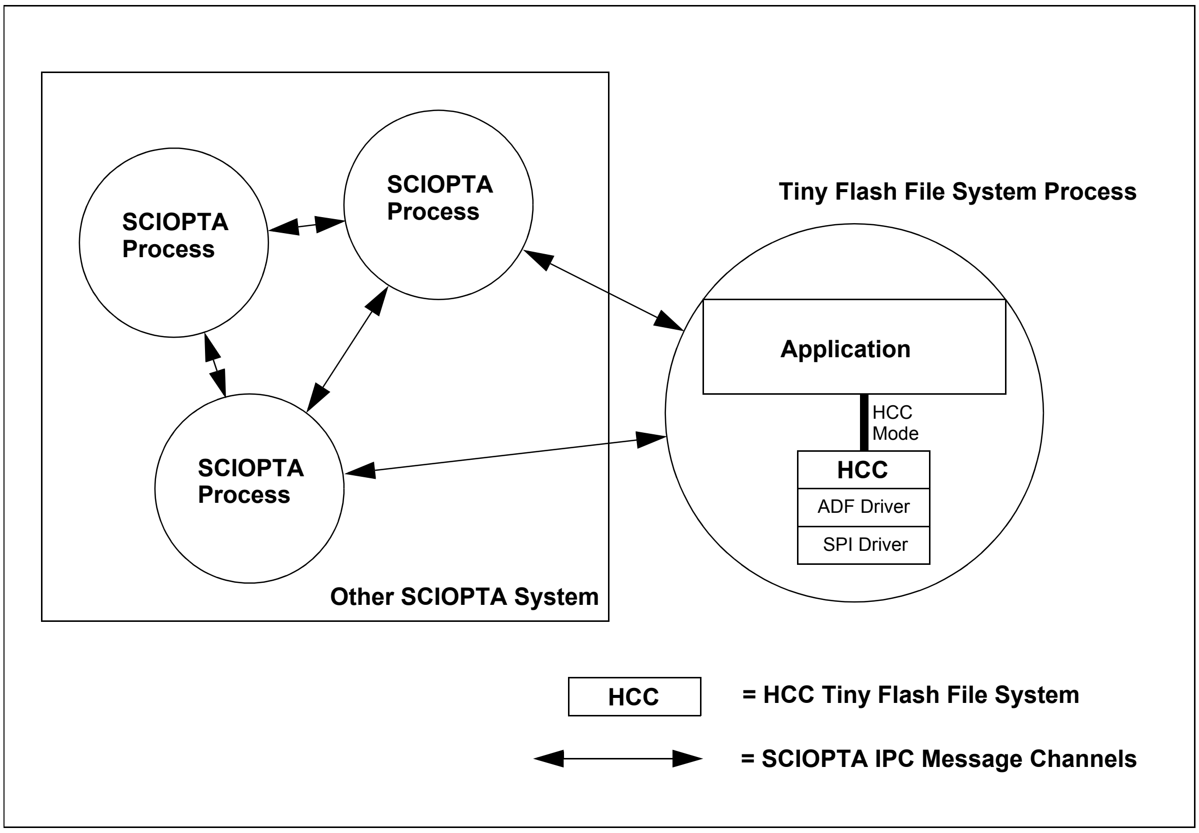


Figure 7-1: SCIOPTA Tiny Flash File System for Atmel DataFlash™ Block Diagram

7.4 Tiny Flash File System Process

The SCIOPTA Tiny Flash File System process contains the user code to access the file system. It is using the Tiny Flash File System from HCC Embedded, the Atmel DataFlash driver and the SPI driver.

In order to use the SCIOPTA Tiny Flash File System you need to include some HCC Tiny Flash File System source files.

7.4.1 HCC Tiny Flash File System

The SCIOPTA Tiny Flash File System for Atmel DataFlash™ uses and integrates the HCC Tiny Flash File System. It consists of the following files:

dev.c	Serial number functions.
f_dir.c	Directory entry handling.
f_ext.c	File and directory changed detection.
f_file.c	File handling functions.
f_volume.c	Volume handling functions.
port.c	Time and date function.

The user does not need to modify these files. They must be included in the application build process.

The HCC Tiny Flash File System files can be found at:

File location<installation_folder>\sciopta\<version>\sfs\hcc\effs-tiny\

7.4.2 Accessing the HCC Tiny Flash File System

For the SCIOPTA Tiny Flash File System for Atmel DataFlash™ the HCC Tiny Flash File System functions are called directly by the user. The HCC Tiny Flash File System runs in the context of the user process.

In this mode the HCC Tiny Flash File System API must be used. Please consult chapter 6 “HCC Tiny Flash File System API” on page 6-1.

In the example directory of SCIOPTA Flash File System for Atmel DataFlash™ delivery you will find an example how to use the API.

For the “effs_tiny” SCIOPTA Tiny Flash File System for Atmel DataFlash™ getting started example the following file contains the HCC Mode flash File System user process:

Files

tiny_effstest.c	Example tiny file system process

File location<installation_folder>\sciopta\<version>\exp\sfs\common\tinyeffs\

7.5 Implementing Atmel DataFlash™ Driver

This driver provides the following primary functions:

1. Manages the 10K writes per sector issue and ensures that the data is always refreshed as required but with minimum overhead.
2. Provides a "safe" write capability such that critical data can be written or copied without the fear of data loss during power failure. Because data written to devices RAM buffer is only committed to the flash after an erase of the page in flash the possibility of power-loss during this process must be protected against.

The driver interfaces to the device through the SPI driver.

Please consult chapter 9.20 "Atmel DataFlash™ Driver" on page 9-34 for information about flash file system drivers.

7.6 Error Hook

Contrary to most conventional real-time operating systems, SCIOPTA uses a centralized mechanism for error reporting called Error Hooks. In traditional real-time operating system, the user needs to check return values of system calls for a possible error condition. In SCIOPTA all error conditions will end up in an Error Hook. This guarantees that all errors are treated and that the error handling does not depend on individual error strategies which might vary from user to user.

There are two error hooks available:

A) Module Error Hook

B) Global Error Hook

If the kernel detect an error condition it will first call the module error hook and if it is not available call the global error hook. Error hooks are normal error handling functions and must be written by the user. Depending on the type of error (fatal or non-fatal) it will not be possible to return from an error hook.

If there are no error hooks present the kernel will enter an infinite loop (at label **SC_ERROR**) and all interrupts are disabled.

Please consult the chapter "Application Programming>Error Hook" of the SCIOPTA ARM - Kernel, User's Guide for more information about Error Information, Error Hook Registering, Error Hook Declaration Syntax and Error Hook Example.

In the example directory of SCIOPTA delivery you will find an example of an error hook.

File

error.c	Error hook example.

File location: <installation_folder>\sciopta\<version>\exp\common\

8 System and Application Configuration

8.1 Introduction

Besides the kernel configuration described in chapter 10 "Kernel Configuration" on page 10-1 which defines system characteristics, modules, processes and message pools, you need to setup and initialize your board and your application.

System and application configuration is done in some specific files such as resethook.S and cstartup.S.

Other system and application configuration functions for the system module such as start hooks, system module hooks and hook registration is usually done in a specific file called system.c which can be found in the SCIOPTA examples deliveries.

System and application configuration functions for other modules (module hooks and hook registration) is usually done in specific files having the same name as the module (dev.c, ips.c etc.) which can also be found in the SCIOPTA examples deliveries.

8.2 System Start

8.2.1 Start Sequence

After a system hardware reset the following sequence will be executed:

1. The kernel calls the function reset_hook.
2. The kernel performs some internal initialization.
3. The kernel calls the C startup function cstartup.
4. The kernel calls the function start_hook.
5. The kernel calls the function TargetSetup. The code of this function is automatically generated by the **SCONF** configuration utility and included in the file sconf.c. TargetSetup creates the system module.
6. The kernel calls the dispatcher.
7. The first process (init process of the system module) is swapped in.

The code of the following functions is automatically generated by the **SCONF** configuration utility and included in the file sconf.c.

8. The INIT process of the system module creates all static modules, processes and pools.
9. The INIT process of the system module calls the system module start function.
10. The process priority of the INIT process of the system module is set to 32 and loops for ever.
11. The INIT Process of each created static module calls the user module hook of each module.
12. The process priority of the INIT process of each created static module is set to 32 and loops for ever.
13. The process with the highest system priority will be swapped-in and executed.

8.2.2 Reset Hook

Description

In SCIOPTA a reset hook must always be present and must have the name **reset_hook**.

The reset hook must be written by the user.

After system reset the SCIOPTA kernel initializes a small stack and jumps directly into the reset hook.

The reset hook is mainly used to do some basic chip and board settings. The C environment is not yet initialized when the reset hook executes. Therefore the reset hook must be **written in assembler**.

Reset hooks are compiler manufacturer and board specific. Reset hook examples can be found in the SCIOPTA Board Support Package deliveries.

Please consult also chapter 9 “Board Support Packages” on page 9-1 for more information.

Source File

resethook.S	Board setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\<cpu>\<board>\src\

Syntax

```
int reset_hook (void);
```

Parameter	Description
none	

Return Value	Description
!= 0	The kernel will immediately call the dispatcher. This will initiate a warm start.
0	The kernel will jump to the C startup function. This will initiate a cold start.

8.2.3 C Startup

After a cold start the kernel will call the C startup function. The C startup function is written in assembler and has the name cstartup. It initializes the C system and replaces the library C startup function. C startup functions are compiler specific. Please note that this file is not needed for IAR.

Source File

cstartup.S	C startup assembler source for GNU GCC

File location: <install_folder>\sciopta\<version>\bsp\arm\src\gnu\

8.2.4 Start Hook

Description

The start hook must always be present and must have the name start_hook. The start hook must be written by the user. If a start hook is declared the kernel will jump into it after the C environment is initialized.

The start hook is mainly used to do chip, board and system initialization. As the C environment is initialized it can be written in C. The start hook would also be the right place to include the registration of the system error hook and other kernel hooks.

In the delivered SCIOPTA examples the start hook is usually included in the file system.c:

system.c	System configuration file including hooks and other setup code.

File location: <installation_folder>\sciopta\<version>\exp\krn\arm\tinyeffs\<board>\

After the start hook has executed the kernel will call the dispatcher and the system will start.

Prototype

```
void start_hook (void);
```

8.2.5 INIT Process

The INIT process is the first process in a module. Each module has at least one process and this is the init process. At module start the init process gets automatically the highest priority (0). After the init process has done some important work it will change its priority to the lowest level (32) and enter an endless loop.

Priority 32 is only allowed for the init process. All other processes are using priority 0 - 31. The INIT process acts therefore also as idle process which will run when all other processes of a module are in the waiting state.

The INIT process of the system module will first be swapped-in followed by the init processes of all other modules.

The code of the module INIT Processes are automatically generated by the **SCONF** configuration utility and placed in the file sconf.c. The module INIT Processes will automatically be named to <module_name>_init and created.

8.2.6 Module Hooks

8.2.6.1 System Module Hook

After all static modules, pools and processes have been created by the INIT Process of the system module the kernel will call a System Module Hook. This is function with the same name as the system module and must be written by the user. Blocking system calls are not allowed in the system module hook. All other system calls may be used.

In the delivered SCIOPTA examples the system module start function is usually included in the file system.c:

system.c	System configuration file including hooks and other setup code.

File location: <installation_folder>\sciopta\<version>\exp\sfs\arm\tinyeffs\<board>\

8.2.6.2 User Modules Hooks

All other user modules have also own individual User Module Hooks. These are functions with the same name of the respective defined and configured modules which will be called by the INIT Process of each respective module.

After returning from the module start functions the INIT Processes of these modules will change its priority to 32 and go into sleep. These module hooks can use all SCIOPTA system calls.

8.3 Specific HCC Tiny File System Configuration

8.3.1 Tiny User Definitions

The **tinyuser.h** contains a set of definitions which configure the file system when used in conjunction with the definitions in the driver. The definitions are documented in the table on the following page.

The API options are available to control the code size.

If there is a real time clock on the system then the RTC_PRESENT should be enabled and the dummy function in **port.c** should be implemented (please consult chapter 7.4.1 "HCC Tiny Flash File System" on page 7-3 for the location of file port.c). If no RTC is present the system will increase a tick counter when a file is written.

The file system settings are used to set limits on the system. If larger numbers are used then the RAM and flash overhead is increased. The alignment settings should not be changed without reference to SCIOPTA. The system is only tested with the default settings.

Include Files

tinyuser.h	User definitions file.

File location: <install_folder>\sciopta\<version>\include\hcc\effs-tiny\

Define	Default Value	Notes
	API Settings	
F_WILDCARD	1	This enables the use of wildcards in filenames and paths.
F_CHECKNAME	1	This enables name checking. If this is disabled the name assigned to a file or directory will not be validated.
F_CHECKMEDIA	1	This enables checking if the file system on the media matches the file system compilation. This is only necessary where the storage media can be replaced.
F_DIRECTORY	1	Enables directories.
F_CHDIR	1	Enables the change directory function.
F_MKDIR	1	Enables the make directory function.
F_RMDIR	1	Enables the remove directory function.
F_GETCWD	1	Enables the get cwd function.
F_DIR_OPTIMIZE	1	Allows for more efficient directory handling but uses more flash and RAM for storage.
F_FIND_ENABLE	1	Enables the find first/findnext functions.
F_CHANGE_NOTIFY	1	Enables the file changed notify function. This is needed to interface to uCDrive.
F_FILELENGTH	1	Enables the file length function.
F_GETFREESPACE	1	Enables the get free space function.

Define	Default Value	Notes
F_DELETE	1	Enables the delete file function.
F_RENAME	1	Enables the file rename function.
Miscellaneous Settings		
F_SEEK_WRITE	1	Enables seek to and write to middle of a file. If this is not required it helps both performance and code size.
RTC_PRESENT	0	Enable this if the system has an RTC and the get time function in **port.c** has been implemented.
File System Settings		
F_MAX_OPEN_FILE	2	Sets the number of simultaneous open files allowed.
F_MAX_FILE_NAME_LENGTH	16	Sets the maximum filename length.
F_MAX_DIR	16	Sets the total number of sub-directories allowed in the system.
Alignment/Type Definitions		
_SUL	4	This should be set to the size of a long on the target system.
_SUS	2	This should be set to the size of an int on the target system.
_SUC	1	This should be set to the size of a char on the target system.
_SALIGN	4	This specifies management structures alignment. This should not be changed.

9 Board Support Packages

9.1 Introduction

Only the device drivers which are needed in a typical SCIOPTA ARM - Tiny Flash File System are listed here.

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for all other information about SCIOPTA BSPs. For other SCIOPTA products please consult the BSP chapter of the respective manual.

9.2 General System Functions and Drivers

System functions and drivers which do not depend on a specific board and a specific CPU and are common for SCIOPTA systems. Files for external systems, chips and controllers.

For the SCIOPTA Tiny Flash File System for Atmel DataFlash™ we will use HCC Tiny Flash File System drivers. Please consult chapter 9.20 “Atmel DataFlash™ Driver” on page 9-34 for more information.

Source Files

None

9.3 ARM System Functions and Drivers

Setup and driver descriptions which do not depend on a specific board and are common for all ARM based processors and controllers.

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for a complete list of kernel drivers.

Project Files

module.ld	Linker script: Module sections (common to all boards) for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\include\

Source Files

cstartup.S	C startup assembler source for GNU GCC.
exception.S	Exception handler for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\gnu\

cstartup.s	C startup assembler source for ARM RealView.
exception.s	Exception handler for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\arm\

exception.s79	Exception handler for IAR Embedded Workbench.

File location: <install_folder>\sciopta\<version>\bsp\arm\src\iar\

9.4 AT91SAM7 System Functions and Drivers

System functions which do not depend on a specific board and are common for all boards with AT91SAM7 based processors.

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for a complete list of kernel drivers.

9.4.1 General AT91SAM7 Functions and Definitions

Include Files

at91sam7.h	AT91SAM7xxx definitions.
at91sam7a3.h, at91sam7a3_inc.h	AT91SAM7A3 definitions.
at91sam7s64.h, at91sam7s64_inc.h	AT91SAM7S64 definitions.
at91sam7s.h, at91sam7s_inc.h	AT91SAM7Sxx definitions.
at91sam7se512.h, at91sam7se512_inc.h	AT91SAM7SE512 definitions.
at91sam7x256.h, at91sam7x256_inc.h	AT91SAM7X256 definitions.
at91sam7x.h, at91sam7x_inc.h	AT91SAM7X definitions.
sys_irq.h	SysIRQ definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\include\

Source Files

irq_handler.S	Interrupt handler for AT91SAM7 and GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\src\gnu\

irq_handler.s	Interrupt handler for AT91SAM7 and ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\src\arm\

irq_handler.s79	Interrupt handler for AT91SAM7 and IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\src\iar\

9.4.2 AT91SAM7 System Tick Driver

9.4.2.1 AT91SAM7 System Tick Configuration

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick driver configuration.

9.4.2.2 AT91SAM7 System Tick Interrupt Process

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick interrupt process.

Source Files

systick.c	System timer setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\src\

9.4.3 AT91SAM7 UART Functions

This is not a full featured serial driver. Just some basic UART routines for the AT91SAM7 controllers are supplied to be used for system logging and printf debugging.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for detailed descriptions of serial drivers.

Include Files

simple_uart.h	Simple UART routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\include\

Source Files

simple_uart.c	Simple polling uart function for printf debugging or logging.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\src\

9.4.4 AT91SAM7 SPI HCC Driver

This is an SPI driver only to be used together with the HCC Tiny Flash File System.

Include Files

spi_hcc.h	SPI definitions for Tiny Flash File System

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\include\

Source Files

spi_hcc.c	SPI driver for Tiny Flash File System.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\src\

9.5 AT91SAM9 System Functions and Drivers

System functions which do not depend on a specific board and are common for all boards with AT91SAM9 based processors.

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of kernel drivers.

9.5.1 General AT91SAM9 Functions and Definitions

Include Files

at91sam9.h, at91sam9_inc.h	Common header for all sam9.
at91sam9260.h, at91sam9260_inc.h	AT91SAM9260 definitions.
at91sam9261.h, at91sam9261_inc.h	AT91SAM9261 definitions.
at91sam9261_inc.inc	AT91SAM9261 definitions.
lib_at91sam9260.h	AT91SAM9260 inlined functions.
lib_at91sam9261.h	AT91SAM9261 inlined functions.
sys_irq.h	SysIRQ definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\include\

Source Files

irq_handler.S	Interrupt handler for AT91SAM9 and GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\src\gnu\

irq_handler.s	Interrupt handler for AT91SAM9 and ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\src\arm\

irq_handler.s79	Interrupt handler for AT91SAM9 and IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\src\iar\

9.5.2 AT91SAM9 System Tick Driver

9.5.2.1 AT91SAM9 System Tick Configuration

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick driver configuration.

9.5.2.2 AT91SAM9 System Tick Interrupt Process

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick interrupt process.

Source Files

systick.c	System timer setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\src\

9.5.3 AT91SAM9 UART Functions

This is not a full featured serial driver. Just some basic UART routines for the AT91SAM9 controllers are supplied to be used for system logging and printf debugging.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for detailed descriptions of serial drivers.

Include Files

simple_uart.h	Simple UART routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\include\

Source Files

simple_uart.c	Simple polling uart function for printf debugging or logging.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\src\

9.6 LPC21xx System Functions and Drivers

System functions which do not depend on a specific board and are common for all boards with LPC21xx based processors.

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of kernel drivers.

9.6.1 General LPC21xx Functions and Definitions

Include Files

lpc21xx.h	LPC2xxx defines.
lpc21xx.iar	LPC2xxx defines.
lpc21xx.inc	LPC2xxx defines.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\include\

Source Files

eintx_irq.S	Interrupt wrapper for eint0..3 for LPC21xx and GNU GCC.
irq_handler.S	Interrupt handler for LPC21xx and GNU GCC.
spi_irq.S	Interrupt wrapper for spi0 and spi1 for LPC21xx and GNU GCC.
systick_irq.S	Interrupt wrapper for systick for LPC21xx and GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\src\gnu\

eintx_irq.s	Interrupt wrapper for eint0..3 for LPC21xx and ARM RealView.
irq_handler.s	Interrupt handler for LPC21xx and ARM RealView.
spi_irq.s	Interrupt wrapper for spi0 and spi1 for LPC21xx and ARM RealView.
systick_irq.s	Interrupt wrapper for systick for LPC21xx and ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\src\arm\

eintx_irq.s79	Interrupt wrapper for eint0..3 for LPC21xx and IAR EW.
irq_handler.s79	Interrupt handler for LPC21xx and IAR EW.
spi_irq.s79	Interrupt wrapper for spi0 and spi1 for LPC21xx and IAR EW.
systick_irq.s79	Interrupt wrapper for systick for LPC21xx and IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\src\iar\

9.6.2 LPC21xx System Tick Driver

9.6.2.1 LPC21xx System Tick Configuration

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick driver configuration.

9.6.2.2 LPC21xx System Tick Interrupt Process

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick interrupt process.

Source Files

systick.c	System timer setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpx21xx\src\

9.6.3 LPC21xx UART Functions

This is not a full featured serial driver. Just some basic UART routines for the LPC21xx controllers are supplied to be used for system logging and printf debugging.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for detailed descriptions of serial drivers.

Include Files

simple_uart.h	Simple UART routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\include\

Source Files

simple_uart.c	Simple polling uart function for printf debugging or logging.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\src\

9.7 LPC23xx and LPC24xx System Functions and Drivers

System functions which do not depend on a specific board and are common for all boards with LPC23xx and LPC24xx based processors.

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of kernel drivers.

9.7.1 General LPC23xx and LPC24xx Functions and Definitions

Include Files

lpc23xx.h	HW register for LPC23xx/LPC24xx.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\include\

Source Files

irq_handler.S	Interrupt handler for LPC23xx/LPC24xx and GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\src\gnu\

irq_handler.s	Interrupt handler for LPC21xx/LPC24xx and ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\src\arm\

irq_handler.s79	Interrupt handler for LPC21xx/LPC24xx and IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\src\iar\

9.7.2 LPC23xx and LPC24xx System Tick Driver

9.7.2.1 LPC23xx and LPC24xx System Tick Configuration

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for a detailed description of the system tick driver configuration.

9.7.2.2 LPC23xx and LPC24xx System Tick Interrupt Process

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for a detailed description of the system tick interrupt process.

Source Files

systick.c	System timer setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpx24xx_lpc23xx\src\

9.7.3 LPC23xx and LPC24xx UART Functions

This is not a full featured serial driver. Just some basic UART routines for the LPC23xx and LPC24xx controllers are supplied to be used for system logging and printf debugging.

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for detailed descriptions of serial drivers.

Include Files

simple_uart.h	Simple UART routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\include\

Source Files

simple_uart.c	Simple polling uart function for printf debugging or logging.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\src\

9.8 STR7 System Functions and Drivers

System functions which do not depend on a specific board and are common for all boards with STR7 based processors.

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of kernel drivers.

9.8.1 General STR7 Functions and Definitions

Include Files

gpio.h	GBIO functions.
rccu.h	RCCU functions and definitions.
str71x.h	STR71x memory map.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\include\

Source Files

irq_handler.S	Interrupt handler for STR7 and GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\src\gnu\

irq_handler.s79	Interrupt handler for STR7 and IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\src\iar\

9.8.2 STR7 System Tick Driver

9.8.2.1 STR7 System Tick Configuration

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick driver configuration.

9.8.2.2 STR7 System Tick Interrupt Process

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick interrupt process.

Source Files

systick.c	System timer setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\src\

9.8.3 STR7 UART Functions

This is not a full featured serial driver. Just some basic UART routines for the STR7 controllers are supplied to be used for system logging and printf debugging.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for detailed descriptions of serial drivers.

Include Files

simple_uart.h	Simple UART routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\include\

Source Files

simple_uart.c	Simple polling uart function for printf debugging or logging.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\src\

9.9 STR9 System Functions and Drivers

System functions which do not depend on a specific board and are common for all boards with STR9 based processors.

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for a complete list of kernel drivers.

9.9.1 General STR9 Functions and Definitions

Include Files

str91x.h	STR91x memory map.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\include\

91x_*.h	STR91x definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\include\str91x\

Source Files

irq_handler.S	Interrupt handler for STR9 and GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\src\gnu\

irq_handler.s79	Interrupt handler for STR9 and IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\src\iar\

9.9.2 STR9 System Tick Driver

9.9.2.1 STR9 System Tick Configuration

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick driver configuration.

9.9.2.2 STR9 System Tick Interrupt Process

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the system tick interrupt process.

Source Files

systick.c	System timer setup.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\src\

9.9.3 STR9 UART Functions

This is not a full featured serial driver. Just some basic UART routines for the STR9 controllers are supplied to be used for system logging and printf debugging.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for detailed descriptions of serial drivers.

Include Files

simple_uart.h	Simple UART routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\include\

Source Files

simple_uart.c	Simple polling uart function for printf debugging or logging.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\src\

9.10 Atmel AT91SAM7A3-EK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.10.1 Atmel AT91SAM7A3-EK Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Atmel AT91SAM7A3-EK board.

9.10.2 General Board Functions and Definitions

Project Files

at91sam7a3-ek.ld	GCC linker script example.
at91sam7a3-ek.sct	ARM RealView linker script example.
at91sam7a3-ek.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7a3-ek\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7a3-ek\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7a3-ek\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7a3-ek\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7a3-ek\src\iar\

9.10.3 LED Driver

Simple functions to access the Atmel AT91SAM7A3-EK board LEDs.

Include Files

led.h	Defines for the Atmel AT91SAM7A3-EK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7a3-ek\include\

Source Files

led.c	Routines to access the LEDs on the Atmel AT91SAM7A3-EK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7a3-ek\src\

9.11 Atmel AT91SAM7SE-EK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.11.1 Atmel AT91SAM7SE-EK Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Atmel AT91SAM7SE-EK board.

9.11.2 General Board Functions and Definitions

Project Files

at91sam7se-ek.ld	GCC linker script example.
at91sam7se-ek.sct	ARM RealView linker script example.
at91sam7se-ek.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7se-ek\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7se-ek\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7se-ek\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7se-ek\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7se-ek\src\iar\

9.11.3 LED Driver

Simple functions to access the Atmel AT91SAM7SE-EK board LEDs.

Include Files

led.h	Defines for the Atmel AT91SAM7SE-EK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7se-ek\include\

Source Files

led.c	Routines to access the LEDs on the Atmel AT91SAM7SE-EK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7se-ek\src\

9.12 Atmel AT91SAM7S-EK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.12.1 Atmel AT91SAM7S-EK Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Atmel AT91SAM7S-EK board.

9.12.2 General Board Functions and Definitions

Project Files

at91sam7s-ek.ld	GCC linker script example.
at91sam7s-ek.sct	ARM RealView linker script example.
at91sam7s-ek.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7s-ek\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7s-ek\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7s-ek\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7s-ek\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7s-ek\src\iar\

9.12.3 LED Driver

Simple functions to access the Atmel AT91SAM7S-EK board LEDs.

Include Files

led.h	Defines for the Atmel AT91SAM7S-EK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7s-ek\include\

Source Files

led.c	Routines to access the LEDs on the Atmel AT91SAM7S-EK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7s-ek\src\

9.13 Atmel AT91SAM7X-EK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.13.1 Atmel AT91SAM7X-EK Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Atmel AT91SAM7X-EK board.

9.13.2 General Board Functions and Definitions

Project Files

at91sam7x-ek.ld	GCC linker script example.
at91sam7x-ek.sct	ARM RealView linker script example.
at91sam7x-ek.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7x-ek\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7x-ek\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7x-ek\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7x-ek\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7x-ek\src\iar\

9.13.3 LED Driver

Simple functions to access the Atmel AT91SAM7X-EK board LEDs.

Include Files

led.h	Defines for the Atmel AT91SAM7X-EK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7x-ek\include\

Source Files

led.c	Routines to access the LEDs on the Atmel AT91SAM7X-EK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam7\at91sam7x-ek\src\

9.14 Atmel AT91SAM9260-EK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.14.1 Atmel AT91SAM9260-EK Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Atmel AT91SAM9260-EK board.

9.14.2 General Board Functions and Definitions

Project Files

at91sam9260-ek.ld	GCC linker script example.
at91sam9260-ek.sct	ARM RealView linker script example.
at91sam9260-ek.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\include\

Include Files

config.h	Board configuration definitions.
config.inc	Board configuration definitions.
spi_cnf.h	SPI configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\include\

Source Files

sdram.c	SDRAM initialization.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\src\

pre_reset.S	Reset initialisation done after loading by romboot for GNU GCC.
resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\src\gnu\

pre_reset.s	Reset initialisation done after loading by romboot for ARM RealView.
resethook.s	Board setup for RAM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\src\iar\

9.14.3 LED Driver

Simple functions to access the Atmel AT91SAM9260-EK board LEDs.

Include Files

led.h	Defines for the Atmel AT91SAM9260-EK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\include\

Source Files

led.c	Routines to access the LEDs on the Atmel AT91SAM9260-EK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9260-ek\src\

9.15 Atmel AT91SAM9261-EK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.15.1 Atmel AT91SAM9261-EK Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Atmel AT91SAM9261-EK board.

9.15.2 General Board Functions and Definitions

Project Files

at91sam9261-ek.ld	GCC linker script example.
at91sam9261-ek.sct	ARM RealView linker script example.
at91sam9261-ek.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\include\

Include Files

config.h	Board configuration definitions.
config.inc	Board configuration definitions.
spi_cnf.h	SPI configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\include\

Source Files

sdram.c	SDRAM initialization.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\src\

pre_reset.S	Reset initialisation done after loading by romboot for GNU GCC.
resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\src\gnu\

pre_reset.s	Reset initialisation done after loading by romboot for ARM RealView.
resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\src\iar\

9.15.3 LED Driver

Simple functions to access the Atmel AT91SAM9261-EK board LEDs.

Include Files

led.h	Defines for the Atmel AT91SAM9261-EK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\include\

Source Files

led.c	Routines to access the LEDs on the Atmel AT91SAM9261-EK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\at91sam9\at91sam9261-ek\src\

9.16 Phytec phyCORE-LPC2294 Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.16.1 Phytec phyCORE-LPC2294 Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Phytec phyCORE-LPC2294 board.

9.16.2 General Board Functions and Definitions

Project Files

phyCore2294.ld	GCC linker script example.
phyCore2294.sct	ARM RealView linker script example.
phyCore2294.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\phyCore2294\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\phyCore2294\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\phyCore2294\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\phyCore2294\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\phyCore2294\src\iar\

9.16.3 LED Driver

Simple functions to access the Phytec phyCORE-LPC2294 board LEDs.

Include Files

led.h	Defines for the Phytec phyCORE-LPC2294 board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\phyCore2294\include\

Source Files

led.c	Routines to access the LEDs on the Phytec phyCORE-LPC2294 board.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc21xx\phyCore2294\src\

9.17 Embedded Artists LPC2468 OEM Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.17.1 Embedded Artists LPC2468 OEM Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the Embedded Artists LPC2468 OEM board.

9.17.2 General Board Functions and Definitions

Project Files

EA_LPC2468_16_OEM.ld	GCC linker script example.
EA_LPC2468_16_OEM.sct	ARM RealView linker script example.
EA_LPC2468_16_OEM.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\EA_LPC2468_16_OEM\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\EA_LPC2468_16_OEM\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\EA_LPC2468_16_OEM\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\EA_LPC2468_16_OEM\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\lpc24xx_lpc23xx\EA_LPC2468_16_OEM\src\iar\

9.18 STR711-SK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for a complete list of board drivers.

9.18.1 STR711-SK Board Description

Please consult the chapter “Board Support Packages” of the SCIOPTA ARM - Kernel, User’s Guide for a detailed description of the STR711-SK board.

9.18.2 General Board Functions and Definitions

Project Files

str711-sk.ld	GCC linker script example.
str711-sk.sct	ARM RealView linker script example.
str711-sk.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\str711-sk\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\str711-sk\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\str711-sk\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\str711-sk\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\str711-sk\src\iar\

9.18.3 LED Driver

Simple functions to access the STR711-SK board LEDs.

Include Files

led.h	Defines for the STR711-SK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\str711-sk\include\

Source Files

led.c	Routines to access the LEDs on the STR711-SK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\str7\str711-sk\src\

9.19 STR912-SK Board

Only the basic drivers are listed here which are needed in a typical SCIOPTA Tiny Flash File System application.

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a complete list of board drivers.

9.19.1 STR912-SK Board Description

Please consult the chapter "Board Support Packages" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the STR912-SK board.

9.19.2 General Board Functions and Definitions

Project Files

str912-sk.ld	GCC linker script example.
str912-sk.sct	ARM RealView linker script example.
str912-sk.xcl	IAR EW linker script example.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\str912-sk\include\

Include Files

config.h	Board configuration definitions.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\str912-sk\include\

Source Files

resethook.S	Board setup for GNU GCC.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\str912-sk\src\gnu\

resethook.s	Board setup for ARM RealView.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\str912-sk\src\arm\

resethook.s79	Board setup for IAR EW.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\str912-sk\src\iar\

9.19.3 LED Driver

Simple functions to access the STR912-SK board LEDs.

Include Files

led.h	Defines for the STR912-SK board's LED routines.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\str912-sk\include\

Source Files

led.c	Routines to access the LEDs on the STR912-SK board.

File location: <install_folder>\sciopta\<version>\bsp\arm\str9\str912-sk\src\

9.20 Atmel DataFlash™ Driver

This driver provides the following primary functions:

1. Manages the 10K writes per sector issue and ensures that the data is always refreshed as required but with minimum overhead.
2. Provides a “safe” write capability such that critical data can be written or copied without the fear of data loss during power failure. Because data written to devices RAM buffer is only committed to the flash after an erase of the page in flash the possibility of power-loss during this process must be protected against.

The driver interfaces to the device through the SPI driver which is documented in the next section.

To implement the Atmel DataFlash driver add the **f_atmel.c**, **f_atmel.h** and **flashset.h** files to the project build.

Include Files

f_atmel.h	Atmel DataFlash™ driver defines.
f_flashset.h	Atmel DataFlash™ driver settings.

File location: <install_folder>\sciopta\<version>\sfs\hcc\effs-tiny\driver\

Source Files

f_atmel.c	Atmel DataFlash™ driver.

File location: <install_folder>\sciopta\<version>\sfs\hcc\effs-tiny\driver\

9.20.1 Setting the Device Type

In the f_atmel.h file set the device type from the available definitions:

```
#define AT45DB11B
#define AT45DB21B
#define AT45DB41B
#define AT45DB81B
#define AT45DB161B
#define AT45DB321B
#define AT45DB642B
```

If the device required is not in the available list please contact SCIOPTA.

This device type selection will automatically set the following driver definitions:

ADF_PAGE_SIZE	The size of pages in the specified device.
ADF_RESERVED_PAGES	The number of pages the driver uses to manage the flash usage. These pages must all be in one sector.
ADF_REAL_PAGE_COUNT	The actual number of pages in the specified device.
ADF_NUM_OF_SECTORS	The number of sectors in the device. In this respect the first two sectors in these devices is counted as one sector.
ADF_PAGES_PER_SECTOR	The number of pages in each of the sectors that are used for data storage.

In the flashset.h file set the definitions in the following table:

Definition	Value	Notes
F_BEGIN_ADDR	(ADF_PAGE_SIZE* ADF_RESERVED_PAGES)	The start address of the flash to be used by the file system.
F_PAGE_SIZE	ADF_PAGE_SIZE	The size of the flash pages.
F_PAGE_COUNT	ADF_PAGE_COUNT	Number of flash pages available to the file system.
F_PAGE_PER_CLUSTER	16	This sets the cluster size. This may be adjusted, smaller values give more flexibility but higher RAM usage. Larger values use less RAM but there is more wastage e.g. the minimum space used by a file is the cluster size.

Please Note:

By using F_BEGIN_ADDR and the F_PAGE_COUNT pages from the device can be hidden from the file system and used for other purposes while maintaining the benefits of using this driver for managing the device.

9.20.2 Atmel DataFlash™ Driver Functions

The following functions are included in the Atmel DataFlash™ driver:

Function	Description
adf_cmd	Sends the specified command and three address bytes trough the SPI.
adf_wait_ready	Waits till the flash chip is busy.
adf_init	Initializes the driver.
adf_copy	Will copy whole pages from the src to the destination address. NOTE: size shall be the integer multiple of the page size. Less that page size will nothing, more than it will copy an additional whole page.
adf_read	Will read size bytes from the flash chip into the buffer pointed by ramaddr.
adf_erase	Erase one or more pages. NOTE: size shall be the intheger multiple of the page size. If size specifyes a partial page, the whole page will be erased.
adf_erase_safe	Erase one page. If the eare is interrupted by a power failure, the driver will resume the operation at the next startup. Note: for any value of size except 0 the function will erase only one page. If size is zero nothing will be done.
adf_write	Will write size bytes to the flash chip from the buffer pointed by ramaddr.
adf_transfer_buff_1	This function fills ram buffer 1 with the contents of a page from the FLASH main area. Note: buffer one functions as a cache.
adf_program_buff_1	This function programs the contents of ram buffer 1 to the FLASH main area. Note: buffer one functions as a cache.
adf_lock_page	This function will generate and program a flash page that logically marks a page. NOTE: this function will destroy the contents of buffer one.
adf_unlock_page	This function will remove a lock page.
adf_check_lock	Will read a "lock" page and check its contents to see if it is valid.
adf_write_safe	Will write size bytes to the flash chip from the buffer pointed by ramaddr. only one page may be accessed at the same time.
find_bmp_page	Finds the most recent bitmap page in the specified management sector based on the stored sequence numbers.
adf_recover	This function will recover any interrupted safe writes or safe_erases.
inc_mgm_wr_ctr	Increases the write counter of the management sector. Will do an update if needed to avoid the 10000 problem.
adf_write_mgm	Will write buffer2 to the management area, and will update necessary variables.
adf_write_mgm	Will write buffer2 to the management area, and will update necessary variables.
update_bmp	This function will update the "10000 bitmap" after a write operation.
format_mgm	This function will format a management sector (write bitmap pages and locks to it).
adf_low_level_format	This function will write default contents into the management area of the flash chip. (Clean bitmap, etc...)

9.21 HCC SPI Driver

Please see the SPI driver sections of the BSPs described in the previous chapters for information where to find specific SCIOPTA SPI drivers (e.g. see chapter 9.4.4 "AT91SAM7 SPI HCC Driver" on page 9-4 for an AT91SAM7).

A sample SPI driver is included in the base HCC delivery. If the flash device being used requires SPI then the (adapted) **spi.c** and **spi.h** files should be included in the project build.

Include Files

spi.h	SPI driver defines.

File location: <install_folder>\sciopta\<version>\sfs\hcc\effs-tiny\driver\

Source Files

spi.c	SPI driver.

File location: <install_folder>\sciopta\<version>\sfs\hcc\effs-tiny\driver\

The sample driver is designed and tested on an MSP430F169. Every microcontroller has its own SPI implementation. If the developer is porting to a new target they should design the driver so that it provides the same upper layer interface as in the MSP430 sample driver. The driver upper layer interface is described below.

The following functions should be provided by the SPI driver:

9.21.1 SPI Driver Functions

9.21.1.1 SPI_INIT

```
void spi_init (void)
```

This function is called by the upper layer driver to initialise the SPI driver for use.

9.21.1.2 SPI_SET_BAUDRATE

```
void spi_set_baudrate (unsigned long)
```

This function is called by the upper layer driver to set the interface baudrate. The call parameter is the value of the baud rate to be set.

9.21.1.3 SPI_TX_8

```
void spi_tx_8 (unsigned char)
```

This function is called by the upper layer to transmit 8bits.

9.21.1.4 SPI_RX_8

```
unsigned char spi_rx_8 (void)
```

This function is called by the upper layer to receive 8bits.

9.21.1.5 SPI_RX_32

```
unsigned long spi_rx_32 (void)
```

This function is called by the upper layer to receive a 32bits.

9.21.2 SPI Driver Macros

The following macros are provided for the upper level driver to control the SPI chip select pin:

SPI_CS_LO	Sets the SPI Chip select for the target device to low.
SPI_CS_HI	Sets the SPI Chip select for the target device to high.

10 Kernel Configuration

10.1 Introduction

The SCIOPTA ARM kernel needs to be configured before you can build and download your application. In the SCIOPTA configuration utility **SCONF** (sconf.exe) you will define the parameters for SCIOPTA systems such as name of systems, static modules, processes and pools.

For a detailed description of the SCONF configuration utility, please consult the chapter "SCONF Kernel Configuration Utility" of the SCIOPTA ARM - Kernel, User's Guide.

10.2 System

For a SCIOPTA project it would be possible to define more than one system. You could configure a SCIOPTA ARM target system and other SCIOPTA target systems from within the same **SCONF** configuration window. But usually you will define just one target system.

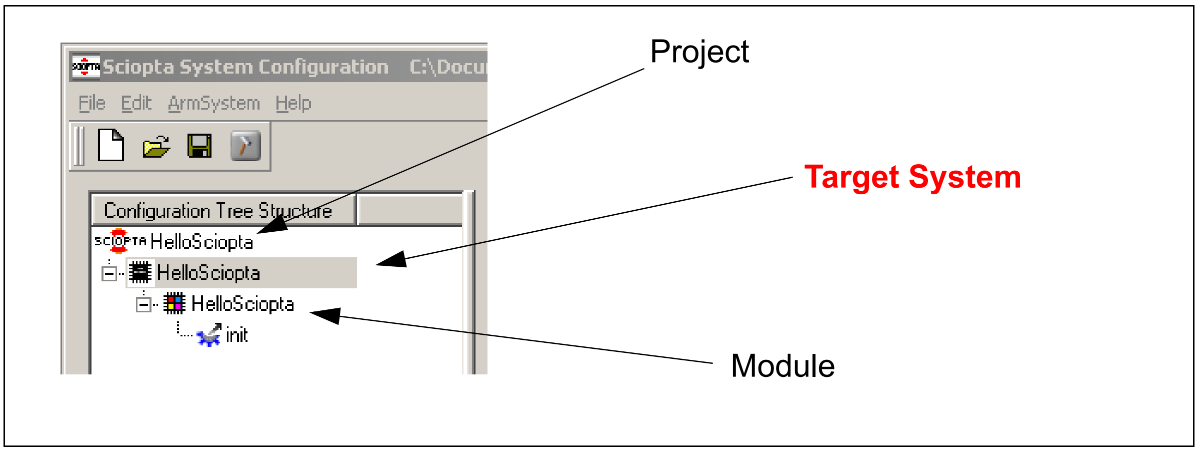


Figure 10-1: SCIOPTA System

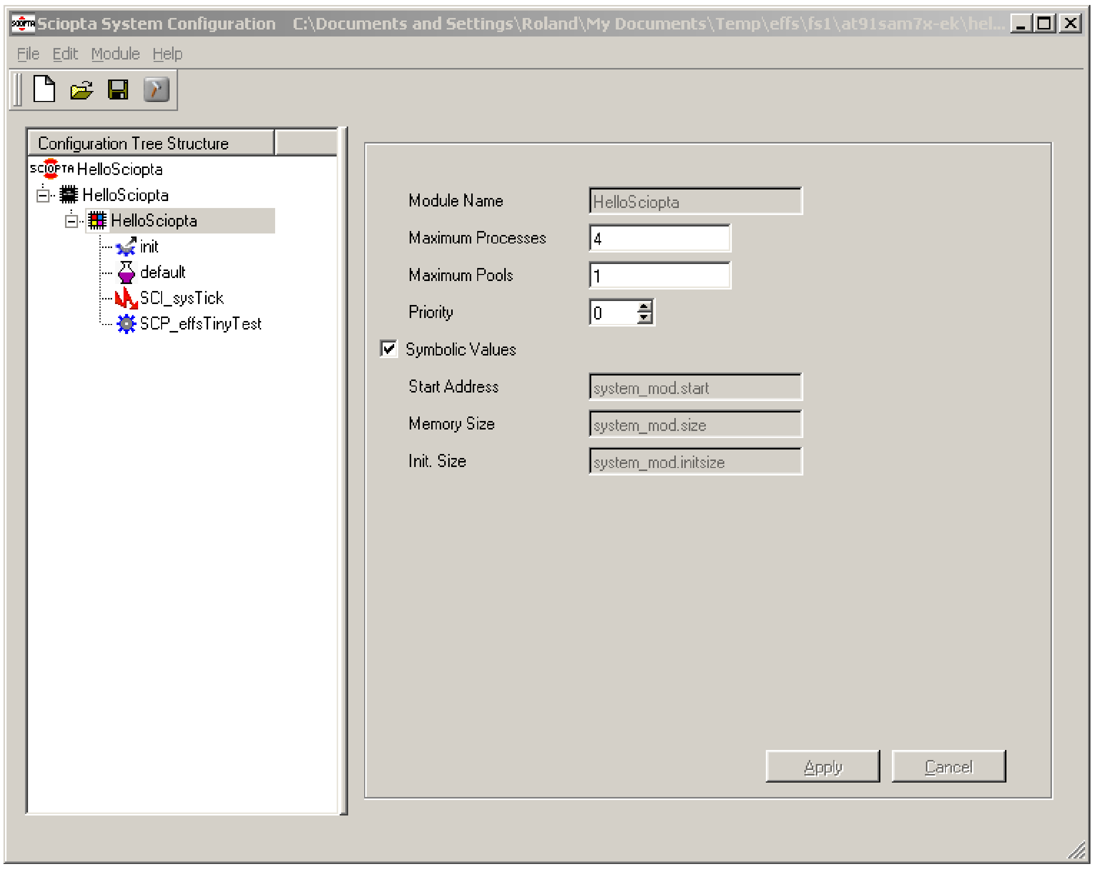
Please consult the chapter "Kernel Configuration" of the SCIOPTA ARM - Kernel, User's Guide for a detailed description of the SCONF target system, module, process and pool configuration and the configuration parameters.

10.3 Modules

Processes can be grouped into modules to improve system structure. A SCIOPTA system must have at least one module, also called module 0 or system module. The system module gets automatically the name of the system.

10.3.1 Modules in a Typical Tiny File System Application

In larger or more complex system it is usually good design practice to partition the system up into more modules. But SCIOPTA Tiny Flash File Systems are rather smaller projects which can also well be placed into one module.


Sciopta System Configuration C:\Documents and Settings\Roland\My Documents\Temp\effs\fs1\at91sam7x-ek\hel...
File Edit Module Help
Configuration Tree Structure
HelloSciopta
HelloSciopta
HelloSciopta
init
default
SCI_sysTick
SCP_effsTinyTest
Apply
Cancel

Module Name: HelloSciopta
Maximum Processes: 4
Maximum Pools: 1
Priority: 0
[✓] Symbolic Values
Start Address: system_mod.start
Memory Size: system_mod.size
Init. Size: system_mod.initsize

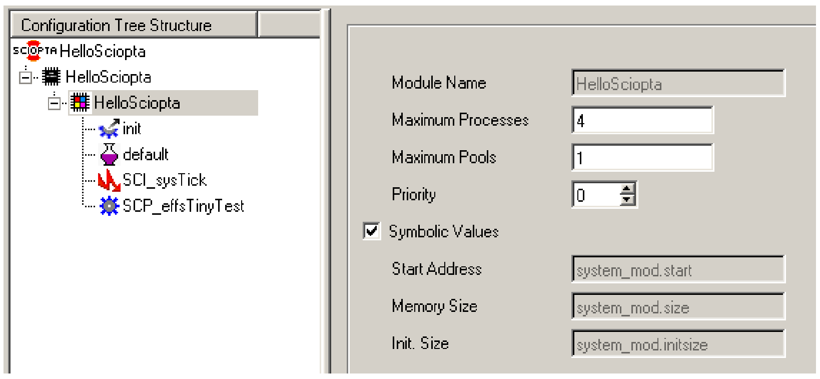
Figure 10-2: Tiny Flash File System SCONF Configuration Example

Above small example system consists of just one module.

HelloSciopta	This is the system module and contains the tick interrupt process, the system pool and application processes. The system module gets automatically the name of the system.

10.4 System Module

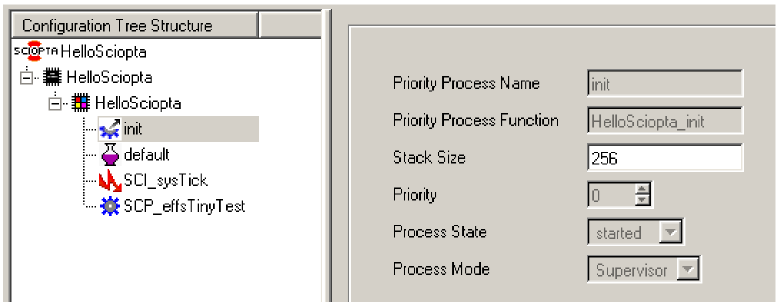
The system module gets automatically the name HelloSciopta which is the system name.



Module Name	HelloSciopta
Maximum Processes	4
Maximum Pools	1
Priority	0
☑ Symbolic Values	
Start Address	system_mod.start
Memory Size	system_mod.size
Init. Size	system_mod.initsize

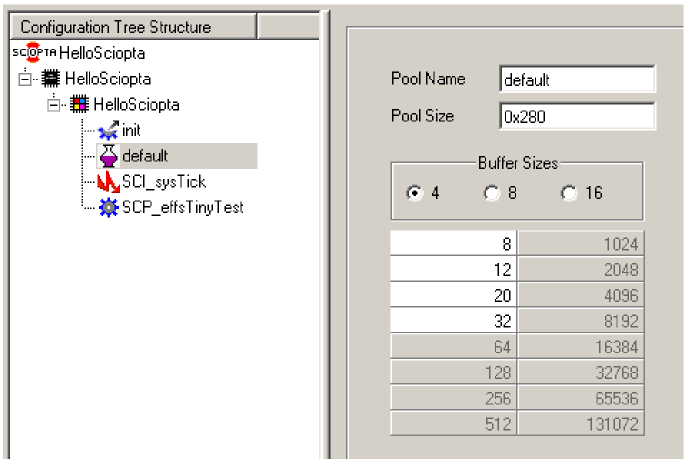
10.4.1 System Module Init Process Configuration

The init process is automatically created and gets the name **init** and the function **HelloSciopta_init**. Only the stack size need to be configured.



Priority Process Name	init
Priority Process Function	HelloSciopta_init
Stack Size	256
Priority	0
Process State	started
Process Mode	Supervisor

10.4.2 Default Pool of the System Module


Configuration Tree Structure

- HelloSciopta
  - HelloSciopta
    - HelloSciopta
      - init
      - default
      - SCI_sysTick
      - SCP_effsTinyTest

Pool Name: default

Pool Size: 0x280

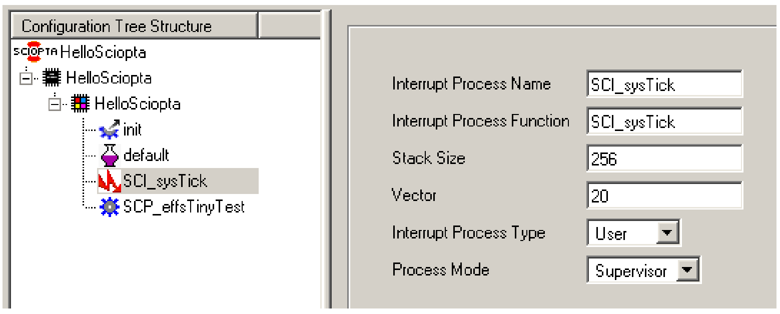
Buffer Sizes: (● 4) ○ 8 ○ 16

8	1024
12	2048
20	4096
32	8192
64	16384
128	32768
256	65536
512	131072

10.4.3 System Tick Interrupt Process

This is an interrupt process included in the SCIOPTA ARM Board Support Package.

Please consult the SCIOPTA ARM - Kernel, User’s Guide for more information about the system tick and system tick interrupt process.


Configuration Tree Structure

- HelloSciopta
  - HelloSciopta
    - HelloSciopta
      - init
      - default
      - SCI_sysTick
      - SCP_effsTinyTest

Interrupt Process Name: SCI_sysTick

Interrupt Process Function: SCI_sysTick

Stack Size: 256

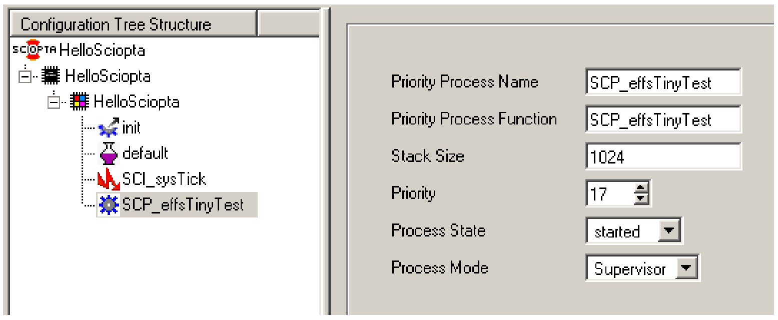
Vector: 20

Interrupt Process Type: User

Process Mode: Supervisor

10.4.4 SCIOPTA Tiny Flash File System User Process

This is the example process of the getting started example.



Priority Process Name	SCP_effsTinyTest
Priority Process Function	SCP_effsTinyTest
Stack Size	1024
Priority	17
Process State	started
Process Mode	Supervisor

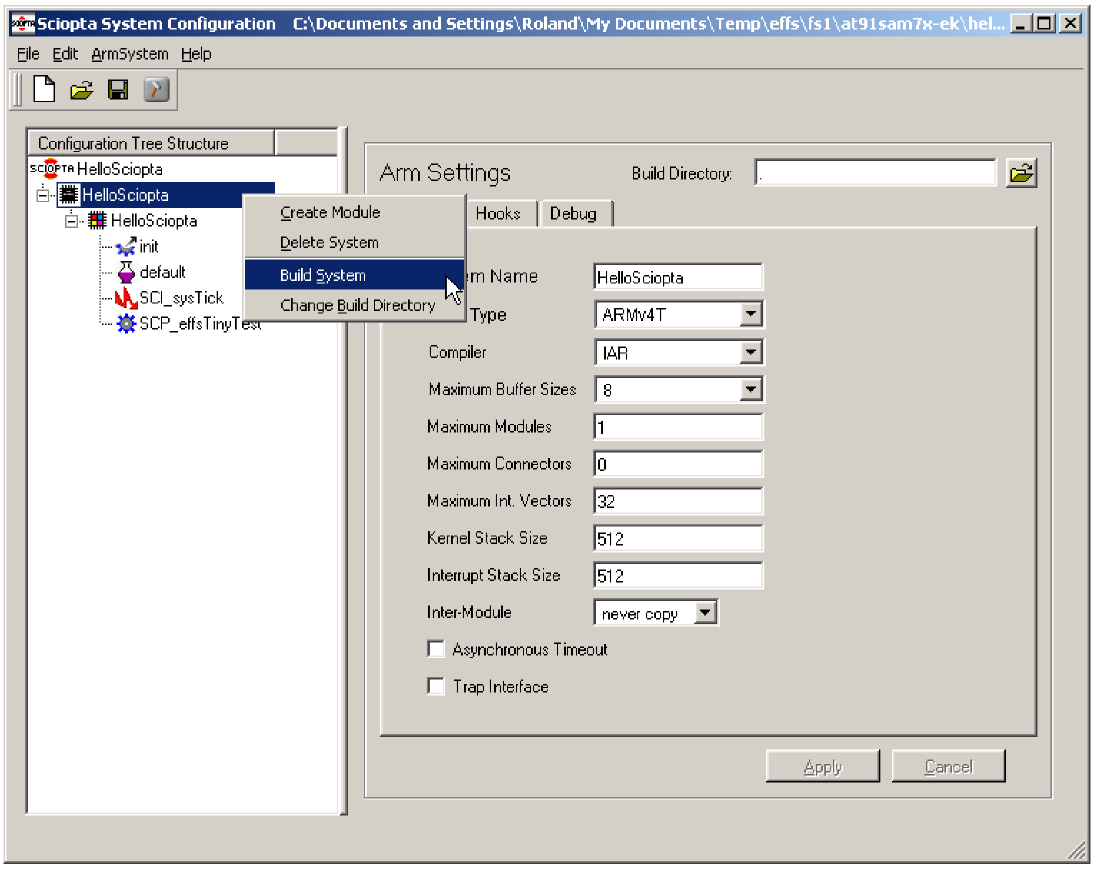
10.5 Generating the Configuration Files

After your configuration is correct the **SCONF** utility will generate three files sciopta.cnf, sconf.h and sconf.c which need to be included into your SCIOPTA project.

10.5.1 Generated Kernel Configuration Files

sciopta.cnf	This is the configured part of the kernel which will be included when the SCIOPTA kernel is assembled.
sconf.h	This is a header file which contains some configuration settings.
sconf.c	This is a C source file which contains the system initialization code.

To build the three files click on the system and right click the mouse. Select the menu Build System. The files sciopta.cnf, sconf.h and sconf.c will be generated into your defined build directory.



System Name: HelloSciopta
Type: ARMv4T
Compiler: IAR
Maximum Buffer Sizes: 8
Maximum Modules: 1
Maximum Connectors: 0
Maximum Int. Vectors: 32
Kernel Stack Size: 512
Interrupt Stack Size: 512
Inter-Module: never copy
☐ Asynchronous Timeout
☐ Trap Interface

Figure 10-3: Build System

11 Libraries and Include Files

11.1 Kernel

The kernel is delivered as a stripped and undocumented assembler source file for each supported compiler.

SCIOPTA Kernel

sciopta.S	SCIOPTA kernel for GNU GCC.
sciopta.s	SCIOPTA kernel for ARM RealView
sciopta.s79	SCIOPTA kernel for IAR EW.

File location: <installation_folder>\sciopta\<version>\lib\arm\krn\

11.2 Kernel Libraries

For the SCIOPTA generic device driver (GDD) functions, the shell functions and the SCIOPTA utilities some prebuilt libraries are included in the delivery.

11.2.1 GNU GCC Kernel Libraries

The file name of the libraries have the following format:

lib<name>_Xt.lib

11.2.1.1 Optimization "X"

The libraries are delivered for three different compiler optimization. The letter **X** defines one of three compiler optimization levels.

"X" can have a value of 0,1 and 2 and defines the optimization.

0	No Optimization.
1	Optimization for size.
2	Optimization for speed.

11.2.1.2 "t"

If the SCIOPTA Trap Interface is used, the libraries with the letter "t" after the Optimization letter X must be included. Systems using a Memory Management Unit (MMU) and the SCIOPTA MMU support module (SMMS, SCIOPTA Memory Management System) need to link these libraries.

11.2.1.3 Included SCIOPTA GNU GCC Kernel Libraries

Utilities	libutil_**X**.a
Generic device driver	libgdd_**X**.a
Shell	libsh_**X**.a

11.2.1.4 Building Kernel Libraries for GCC

Please consult the chapter "Libraries and Include Files" of the SCIOPTA ARM - Kernel, User's Guide for information how to build the kernel libraries.

11.2.2 ARM RealView Kernel Libraries

The file name of the libraries have the following format:

<name>_Xt.l

11.2.2.1 Optimization "X"

The libraries are delivered for three different compiler optimization. The letter **X** defines one of three compiler optimization levels.

"X" can have a value of 0,1 and 2 and defines the optimization.

0	No Optimization.
1	Optimization for size.
2	Optimization for speed.

11.2.2.2 "t"

If the SCIOPTA Trap Interface is used, the libraries with the letter "t" after the Optimization letter X must be included. Systems using a Memory Management Unit (MMU) and the SCIOPTA MMU support module (SMMS, SCIOPTA Memory Management System) need to link these libraries.

11.2.2.3 Included SCIOPTA GNU GCC Kernel Libraries

Utilities	util_**X**.l
Generic device driver	gdd_**X**.l
Shell	sh_**X**.l

11.2.2.4 Building Kernel Libraries for ARM RealView

Please consult the chapter "Libraries and Include Files" of the SCIOPTA ARM - Kernel, User's Guide for information how to build the kernel libraries.

11.2.3 IAR EW Kernel Libraries

The file name of the libraries have the following format:

<name>_Xt.r79

11.2.3.1 Optimization “X”

The libraries are delivered for three different compiler optimization. The letter **X** defines one of three compiler optimization levels.

“X” can have a value of 0,1 and 2 and defines the optimization.

0	No Optimization.
1	Optimization for size.
2	Optimization for speed.

11.2.3.2 “t”

If the SCIOPTA Trap Interface is used, the libraries with the letter “t” after the Optimization letter X must be included. Systems using a Memory Management Unit (MMU) and the SCIOPTA MMU support module (SMMS, SCIOPTA Memory Management System) need to link these libraries.

11.2.3.3 Included SCIOPTA GNU GCC Kernel Libraries

Utilities	util_**X**.s79
Generic device driver	gdd_**X**.s79
Shell	sh_**X**.s79

11.2.3.4 Building Kernel Libraries for IAR EW

Please consult the chapter “Libraries and Include Files” of the SCIOPTA ARM - Kernel, User’s Guide for information how to build the kernel libraries.

11.3 SCIOPTA Tiny File System Library

There is no specific file system library to include for the SCIOPTA Tiny Flash File System for Atmel DataFlash™.

11.4 Include Files

11.4.1 Include Files Search Directories

Please make sure that the environment variable **SCIOPTA_HOME** is defined as explained in the chapter "Installation" of the SCIOPTA ARM - Kernel, User's Guide.

Define the following entries the include files search directories field of your IDE:

```
.
%(SCIOPTA_HOME)\include
%(SCIOPTA_HOME)\include\sciopta\arm
```

Depending on the CPU and board you are using:

```
%(SCIOPTA_HOME)\bsp\arm\include
%(SCIOPTA_HOME)\bsp\arm\<cpu>\include
%(SCIOPTA_HOME)\bsp\arm\<cpu>\<BOARD>\include
```

11.4.2 Main Include File sciopta.h

This file contains some main definitions and the SCIOPTA Application Programming Interface.

Each module or file which is using SCIOPTA system calls and definitions must include the file **sciopta.h**.

sciopta.h	Main include file.

File location: <installation_folder>\sciopta\<version>\include\

The file sciopta.h includes all specific API header files.

File location: <installation_folder>\sciopta\<version>\include\kernel

11.4.3 Configuration Definitions sconf.h

Files or modules which are SCIOPTA configuration dependent need to include first the file **sconf.h** and then the file **sciopta.h**.

The file **sconf.h** needs to be included if for instance you want to know the maximum number of modules allowed in a system. This information is stored in **SC_MAX_MODULES** in the file **sconf.h**. Please remember that **sconf.h** is automatically generated by the **sconf** configuration tool.

11.4.4 Main Data Types types.h

These types are introduced to allow portability between various SCIOPTA implementations.

The main data types are defined in the file types.h located in ossys. These types are not target processor dependent.

types.h	Processor independent data types.

File location: <installation_folder>\sciopta\<version>\include\ossys\

11.4.5 ARM Data Types types.h

The ARM specific data types are defined in the file types.h located in \arm\arch.

types.h	ARM data types.

File location: <installation_folder>\sciopta\<version>\include\sciopta\arm\arch\

11.4.6 Global System Definitions defines.h

System wide definitions are defined in the file defines.h. Among other global definitions, the base addresses of the IDs of the SCIOPTA system messages are defined in this file. Please consult this file for managing and organizing the message IDs of your application.

defines.h	System wide constant definitions.

File location: <installation_folder>\sciopta\<version>\include\sciopta\arm\arch\

12 Manual Versions

12.1 Manual Versions 1.1

- Chapter 3 Getting Started, some minor corrections.
- Chapter 3.2.3.1 Equipment, MSYS utility removed.
- Chapter 4.6 SCIOPTA Tiny Flash File System Files, sfs\hcc\ folder corrected.
- Chapter 4.10 General System Functions and Drivers, sfs\hcc\ folder corrected.
- Chapter 7.4.1 HCC Tiny Flash File System, sfs\hcc\ folder corrected.
- Chapter 8.3.1 Tiny User Definitions, sfs\hcc\ folder corrected.
- Chapter 9.16 Phytec phyCORE-LPC2294 Board, added.
- Chapter 9.20 Atmel DataFlash™ Driver, sfs\hcc\ folder corrected.
- Chapter 9.21 HCC SPI Driver, sfs\hcc\ folder corrected.

12.2 Manual Version 1.0

- Initial Version.

13 Index