NTNU

Norwegian University of
Science and Technology

Linux Support for AVR32 UC3A

Adaption of the Linux kernel and toolchain

Pål Driveklepp
Olav Morken
Gunnar Rangøy

Master of Science in Computer Science

Submission date: June 2009

Supervisor: Morten Hartmann, IDI

Co-supervisor: Håvard Skinnemoen, Atmel

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The goal of this project is to adapt the Linux kernel and a toolchain to support the Atmel AVR32 UC3A0512 microcontroller. This involves adaption of the GNU Compiler Collection (GCC) and associated tools, and the Linux kernel and drivers specific to the Atmel AVR32 UC3 CPU architecture. In addition, a set of useful applications should be selected, compiled and tested.

Assignment given: 15. January 2009
Supervisor: Morten Hartmann, IDI

Abstract

The use of Linux in embedded systems is steadily growing in popularity. The UC3A is a series of high performance, low power 32-bit microcontrollers aimed at several industrial and commercial applications including Programmable Logic Controllers (PLCs), instrumentation, phones, vending machines and more. The main goal of this project was to complete the adaptation of the Linux kernel, compiler and loader software, in order to enable the Linux kernel to load and run applications on this device. In addition, a set of useful applications should be picked, compiled and tested on the target platform to indicate a complete software solution.

This master's thesis is a continuation, by the same three students, of the work of a student project during the fall of 2008. In this report we present in detail the findings, challenges, choices and and solutions involved in the working process. During the course of this project, we have successfully adapted the Linux kernel, and a toolchain for generating binaries loadable by Linux. A set of test applications have been compiled and tested on the resulting platform. This project has resulted in the submission of a revised patch series for the U-Boot boot loader, one patch series for Linux, and one for the toolchain. Requirements have been created, and tests for the requirements have been carried out.

Preface

This master's thesis documents the work done by a group of three students working on their thesis assignment during the spring of 2009 at the Department of Computer and Information Science at the Norwegian University of Science and Technology.

We would like to thank Håvard Skinnemoen at Atmel Norway for his help and guidance, and Atmel for providing the required hardware. We would also like to thank our supervisor Morten Hartmann. A special thanks goes to Øyvind Rangøy, who took the time to read and comment errors in this report.

Contents

Abstract **i**

Preface **iii**

Contents **v**

1 Introduction **1**
1.1 Assignment 1
1.2 Project continuation 2
1.3 Interpretation 3
1.3.1 Requirements 4
1.4 Structure of this report 5

2 Background **7**
2.1 Virtual memory 7
2.1.1 Copy-on-write 8
2.2 Memory Protection Unit (MPU) 9
2.3 Unaligned memory copy 9
2.4 Static Random Access Memory (SRAM) 9
2.5 AVR32 Architecture 11
2.5.1 Registers 11
2.5.2 Instructions 14
2.5.3 Sub-architectures 14
2.5.4 Revisions 15
2.5.5 Execution modes 15
2.5.6 Exception and interrupt handling 16
2.6 The AP7000 microcontroller 16
2.7 The UC3A0512 microcontroller 17
2.7.1 Logical layout 17
2.7.2 Features 19
2.7.3 Chip revisions 25
2.7.4 AP7000 versus UC3A0512 25
2.8 EVK1100 26

2.9 JTAG . . . 28
2.10 Binary formats . . . 28
2.10.1 Terminology . . . 29
2.10.2 ELF . . . 30
2.10.3 FDPIC ELF . . . 33
2.10.4 Flat . . . 36
2.10.5 Comparison of binary formats . . . 36
2.11 Linux . . . 36
2.11.1 Configuration . . . 37
2.11.2 Tasks . . . 38
2.11.3 uClinux . . . 39
2.12 U-Boot . . . 40
2.12.1 Contributions . . . 40
2.13 Toolchain . . . 40
2.13.1 Terminology . . . 40
2.13.2 Linux toolchain . . . 41
2.13.3 GCC . . . 42
2.13.4 GNU Binutils . . . 43
2.13.5 `elf2flt` . . . 43
2.13.6 Libraries . . . 43
2.13.7 GDB . . . 44
2.14 BusyBox . . . 44
2.15 Server protocols . . . 45
2.15.1 DHCP . . . 46
2.15.2 TFTP . . . 46
2.15.3 NFS . . . 46
2.16 Open-source collaboration . . . 46
2.16.1 Git . . . 46
2.16.2 Merging with current versions . . . 46
2.16.3 Splitting up patches . . . 47
2.16.4 Patch submission format . . . 47
2.16.5 Signing your work . . . 47
2.16.6 Upstream . . . 48
2.17 Previous work . . . 48
2.17.1 AP7 series . . . 48
2.17.2 Linux support for MMU-less systems . . . 48
2.17.3 Implementations for other architectures . . . 48
2.17.4 SRAM expansion board . . . 49

3 Implementation **51**
3.1 Methodology . . . 51
3.1.1 Setting goals and preliminary milestones . . . 53
3.1.2 Milestone identification and implementation . . . 53

3.1.3 Review . . . 54
3.2 Expected changes . . . 54
3.2.1 U-Boot . . . 54
3.2.2 Select binary format . . . 54
3.2.3 Linux . . . 54
3.2.4 Toolchain . . . 55
3.2.5 User space . . . 55
3.3 Development setup . . . 55
3.3.1 JTAG . . . 56
3.3.2 Serial cable . . . 57
3.3.3 Networking setup . . . 57
3.4 U-Boot . . . 57
3.4.1 Network speed limiting . . . 58
3.4.2 Adding the EVK1100 board to lists . . . 58
3.4.3 Precedence safety fix . . . 58
3.4.4 Esthetical and other minor changes . . . 59
3.4.5 Auto detection of PHY address . . . 59
3.4.6 Removal of bug workaround . . . 59
3.5 Binary format selection . . . 59
3.6 Linux kernel . . . 60
3.6.1 Rebasing . . . 60
3.6.2 Configuration files and make files . . . 61
3.6.3 UC3A support . . . 61
3.6.4 Cache . . . 62
3.6.5 Clocks . . . 62
3.6.6 Limiting network device speed . . . 63
3.6.7 GPIO . . . 63
3.6.8 LED device driver . . . 63
3.6.9 Serial Peripheral Interface (SPI) with DMA support . . . 63
3.6.10 Interrupt bug workaround . . . 64
3.6.11 Memory to memory copying . . . 64
3.6.12 Memory copying with checksumming . . . 64
3.6.13 User space memory access . . . 65
3.6.14 Address space layout . . . 67
3.6.15 Event handling entry points . . . 68
3.6.16 FDPIC ELF . . . 72
3.6.17 Splitting of paging_init . . . 73
3.6.18 Use of existing macro . . . 74
3.6.19 Patch summary . . . 74
3.7 Toolchain adaptation . . . 76
3.7.1 GCC . . . 76
3.7.2 Binutils . . . 78
3.7.3 uClibc . . . 82

3.7.4 `elf2flt` . . . 83
3.7.5 PIE support . . . 84
3.8 SRAM optimization . . . 84
3.8.1 Routing of signals . . . 85
3.8.2 Joystick pull-up conflict . . . 85
3.8.3 LED resistor conflict . . . 86
3.9 SPI chip enable . . . 86
3.10 BusyBox . . . 88
3.11 Obtaining and distributing source code . . . 89
3.11.1 Buildroot . . . 89
3.11.2 GCC . . . 89
3.11.3 GNU Binutils . . . 90
3.11.4 uClibc . . . 90
3.11.5 `elf2flt` . . . 90
3.11.6 U-Boot . . . 90
3.11.7 Linux . . . 91
3.11.8 BusyBox . . . 91

4 Testing and results **93**
4.1 U-Boot . . . 93
4.1.1 SPI support, requirement 1 . . . 93
4.1.2 Loading from DataFlash or SD card, requirement 2 . . . 93
4.1.3 Patch cleanup, requirement 3 . . . 93
4.2 Linux . . . 94
4.2.1 Booting Linux kernel, requirement 4 . . . 94
4.2.2 Running user space binaries, requirement 5 . . . 95
4.2.3 Hardware support, requirement 6 . . . 95
4.2.4 Exceptions, requirement 7 . . . 96
4.2.5 Code submission, requirement 8 . . . 98
4.3 Toolchain . . . 98
4.3.1 Select binary format, requirement 9 . . . 98
4.3.2 Produce binaries, requirement 10 . . . 98
4.3.3 Produce libraries, requirement 11 . . . 99
4.3.4 Code submission, requirement 12 . . . 99
4.4 Linux user space . . . 99
4.4.1 BusyBox, requirement 13 . . . 99
4.5 Patch submission feedback . . . 99
4.5.1 U-Boot . . . 99
4.5.2 Linux . . . 104
4.5.3 Toolchain . . . 111

5 Conclusion **113**

6 Future work **115**
6.1 U-Boot . . . 115
6.2 Linux . . . 115
6.2.1 PDCA support . . . 115
6.2.2 SPI support . . . 116
6.2.3 MPU support . . . 116
6.2.4 Support for on-chip devices . . . 116
6.2.5 Memory copy optimization . . . 116
6.2.6 Debug support . . . 116
6.2.7 FDPIC ELF support for systems with an MMU . . . 117
6.3 Toolchain . . . 118
6.3.1 Dynamic linking . . . 118
6.3.2 Error handling . . . 118
6.4 AVR32B series compatibility . . . 118

7 Bibliography **121**

A Acronyms **123**

B U-Boot patch cleanup **127**
B.1 Network limiting reorganization . . . 127
B.2 Add board to lists . . . 128
B.3 Precedence safety fix . . . 128
B.4 Board configuration . . . 128
B.5 Keeping lists sorted . . . 129
B.6 Removal of TODOs . . . 129
B.7 Coding style fixes . . . 131

C Unsubmitted U-Boot changes **135**

D Linux kernel patches **137**
D.0 Cover letter . . . 137
D.1 Network speed limiting . . . 141
D.2 Avoid register reset . . . 141
D.3 Split paging function . . . 142
D.4 Use task_pt_regs macro . . . 143
D.5 FDPIC ELF support . . . 144
D.6 Introduce cache and aligned flags . . . 145
D.7 Disable mm-tlb.c . . . 146
D.8 fault.c for !CONFIG_MMU . . . 146
D.9 ioremap and iounmap for !CONFIG_MMU . . . 147
D.10 MMU dummy functions . . . 148
D.11 mm_context_t for !CONFIG_MMU . . . 149
D.12 Add cache function stubs . . . 149

D.13 copy_user.S for !CONFIG_NOUNALIGNED . . . 150
D.14 csum_partial: support for chips that cannot do unaligned accesses . . . 152
D.15 Avoid unaligned access in uaccess.h . . . 154
D.16 memcpy for !CONFIG_NOUNALIGNED . . . 155
D.17 Mark AVR32B code with subarch flag . . . 156
D.18 mm-dma-coherent.c: ifdef AVR32B code . . . 157
D.19 Disable ret_if_privileged macro . . . 157
D.20 AVR32A-support in Kconfig . . . 158
D.21 AVR32A address space support . . . 158
D.22 Change maximum task size for AVR32A . . . 160
D.23 Fix ___range_ok for AVR32A in uaccess.h . . . 160
D.24 Support for AVR32A entry-avr32a.S . . . 161
D.25 Change HIMEM_START for AVR32A . . . 170
D.26 New pt_regs layout for AVR32A . . . 170
D.27 UC3A0512ES interrupt bug workaround . . . 171
D.28 UC3A0xxx support . . . 172
D.29 Board support for ATEVK1100 . . . 204

E PDCA, SPI and DataFlash support 207

F Toolchain patches 217
F.1 Coverletter . . . 217
F.2 GCC changes . . . 217
F.3 GNU binutils changes . . . 219
F.4 uClibc changes . . . 227
F.5 Unsubmitted GCC change . . . 232

G Patch for elf2flt 233

H EVK1100 SRAM expansion board 237

I Test source code 239
I.1 Linux exception tests . . . 239
I.1.1 Unaligned read . . . 239
I.1.2 Unaligned write . . . 239
I.1.3 Invalid read . . . 240
I.1.4 Invalid write . . . 240
I.1.5 Invalid opcode (aligned) . . . 241
I.1.6 Invalid opcode (unaligned) . . . 241
I.2 Toolchain tests . . . 241
I.2.1 Simple program . . . 241
I.2.2 More complex program . . . 242

J Digital appendices **243**
J.1 Linux patches 243
J.2 U-Boot patches 243
J.3 U-Boot unsubmitted changes 243
J.4 Toolchain patches 243
J.5 elf2flt changes 243
J.6 SPI DMA changes 243
J.7 Tests 243

Chapter 1

Introduction

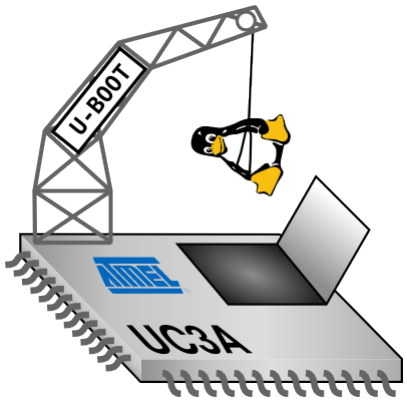


Figure 1.1: U-Boot loading Linux on the UC3A

1.1 Assignment

This master's thesis is the continuation of an earlier project with unfinished goals. The project description was formulated by Atmel, and the requirements and goals were derived from it. In this master's thesis, we resume the work by continuing where the previous work was suspended.

Atmel's problem formulation, given as a an assignment proposition to us via our supervisor is quoted below. Figure 1.1 sums up the project concept in an informal illustration.

> ***Linux kernel support for Atmel AVR32 UC3 processors***
>
> *The project's goal is to boot a Linux kernel on an Atmel AVR32 UC3A0512 microcontroller. In order to boot a Linux system, the following software requires specific adaption / porting:*

- *A boot loader. Das U-Boot is currently the only boot loader capable of loading AVR32 Linux.*
- *The Linux kernel. Obviously.*
- *A toolchain capable of generating "flat" binaries. On AP7, ELF binaries are used, but the Linux ELF loader does not support systems without an Memory Management Unit (MMU). The AP7 core includes an MMU, the UC3 core does not.*
- *Linux applications. This is the easy part, once all the other pieces are in place. But picking out a set of applications that is useful on specific UC3-based development boards is still a task that needs to be done.*

The work will thus include adaption of the GNU Compiler Collection (GCC) and associated tools, boot loader, Linux kernel and driver-implementation specific to the Atmel AVR32 UC3 Central Processor Unit (CPU) architecture.

All work will be completed using Atmel's development boards and debugging tools, including ATEVK1100, JTAGICE mkII or AVRONE! All work will be covered by the GNU General Public License (GPL), as defined by the individual LICENSE and COPYRIGHTs of the projects and will be published in an open source context, through the AVR32 Community Website at http://www.avr32linux.org

A list of URLs to detailed descriptions of relevant projects were also given:

- Linux on UC3:
  http://avr32linux.org/twiki/bin/view/Main/LinuxOnUC3
- U-Boot bootloader:
  http://avr32linux.org/twiki/bin/view/Main/UBootOnUC3
- Linux kernel on UC3:
  http://avr32linux.org/twiki/bin/view/Main/LinuxKernelOnUC3

Håvard Skinnemoen was our contact at Atmel. He provided us with further specifications and guidance, and the necessary tools for the project.

1.2 Project continuation

This master's thesis picks up the threads from the project done by the same three students during the fall of 2008. At the beginning of the work with this thesis, the status could be summed up as follows:

- U-Boot was able to successfully load the kernel via Ethernet or serial port.
- Patches for U-Boot had been submitted, but never revised.
- The hardware setup in the boot sequence of the Linux kernel was partly adapted.

- Linux booted and gave output to the serial console, but halted when trying to load the first user space program (`init`).
- No patches for the Linux kernel had beenassembled or submitted.
- No changes had been done to the toolchain.

Because we had the same main objectives in the project during the fall of 2008, much of the background material from that report is still relevant. Applicable parts of the background chapter have been reused, and new sections have been added. Sections written for the previous project that is still used in this report, is listed in table 1.1.

Section		**Re-use**
Section 2.4	SRAM,	Unchanged
Section 2.3	Unaligned memory copying,	Unchanged
Section 2.4	SRAM,	Unchanged
Section 2.5	AVR32 Architecture	Revised
Section 2.5.3	Sub-architectures	Expanded
Section 2.6	AP7000	Unchanged
Section 2.7	The UC3A0512 microcontroller	Unchanged
Section 2.7.1	UC3a0512 logical layout	Revised
Section 2.7.2	Internal flash	Expanded
Section 2.7.2	EBI	Expanded
Section 2.7.2	SPI	Expanded
Section 2.7.4	AP7000 versus UC3A0512	Expanded
Section 2.9	JTAG	Unchanged
Section 2.11	Linux	Unchanged
Section 2.11.1	Configuration	New
Section 2.11.2	Tasks	New
Section 2.11.3	uClinux	Revised
Section 2.12	U-Boot	Reduced
Section 2.13	Toolchain	Heavily reworked

Table 1.1: Sections reused

1.3 Interpretation

The initial goals and guidelines were defined by Atmel for the preceding student project of fall 2008, but as an independent university group we were free to modify the assignment in any way we wanted as long as our supervisor would approve that the educational goals were satisfied. However, there were no conflicts between our desired goals and the goals suggested by Atmel.

1.3.1 Requirements

This subsection groups and lists the requirements defined for this thesis. The requirements are based on the assignment previously formulated by Atmel, the unmet requirements and suggested future work from the previous project. All of the requirements assume the use of the EVK1100 evaluation kit with the UC3A0512 microcontroller. Requirements marked with [1] are unsolved by this project, and requirements marked with [2] are only partly fulfilled.

Software and hardware components involved are introduced in chapter 2.

U-Boot

1. SPI support (needed if the kernel is loaded from DataFlash or SD card, and for using the LCD display)[1]
2. Load Linux from DataFlash or SD memory card[1]
3. Clean up patches and commit a new version

The Linux kernel

The Linux kernel should have the following features:

4. The Linux kernel must be able to boot.
   (a) Output to serial console
   (b) Initialize networking
   (c) Receive network configuration using DHCP.
   (d) Mount necessary file systems:
      i. NFS root file system.
      ii. proc file system
      iii. sysfs file system
      iv. devpts file system
      v. devshm file system
   (e) Load and execute an init application.
5. The Linux kernel must be able to run user space binaries.
6. The Linux kernel must support the most central hardware located on the EVK1100. The following hardware were identified as central and important:
   (a) Light Emitting Diodes (LEDs) to give status information (optional)
   (b) DataFlash (optional)[1]
   (c) LCD display (optional)[1]
   (d) SD Card (optional)[1]

(e) SPI (optional, needed for requirement 6b, 6c and 6d)[1]

(f) DMA (optional, suggested by requirement 6e)[1]

(g) Network adapter

7. Exceptions must be handled.[2]
8. Resulting source code must be submitted to the appropriate source code maintainers.[2]

Toolchain

The toolchain should be adapted to be capable of generating executables for the UC3A running Linux. This involves the following:

9. A suitable binary format must be selected, this could be either:

   (a) FDPIC ELF

   (b) Flat

10. GCC must be able to generate statically linked executables.
11. GCC must be able to generate dynamically linked executables and libraries.[1]
12. Resulting source code must be submitted to the appropriate source code maintainers.[2]

Linux applications

13. A shell and tools for basic file manipulation, user management and networking should be able to compile, load and run. This includes tools like ls, cp, cat, grep, find, mkdir, rm, rmdir, df, du, vi, diff, adduser, passwd, mount, less, ifconfig, telnet server, free, ps and a shell (ash/hush/msh)

1.4 Structure of this report

This chapter has introduced the assignment, continuation of the previous project, our interpretation, the scope of the project, and a formal requirements specification formulated from the assignment. Chapter 2 introduces the concepts, software and hardware components involved in the development process. The last section of chapter 2 also explores previous work relevant to this project.

Chapter 3 describes in detail the work carried out, the decisions made and the arguments for these. The requirements have been tested, and the tests and results are listed in chapter 4. This chapter also presents some of the feedback from our code submissions. In chapter 5 we conclude the project as a whole. Chapter 6 discusses further work that should be carried out in the future, either by us or others. The final chapter, chapter 7, lists our references.

Note that a list of acronyms is included as the first appendix, appendix A. Most of our patches are included in the appendices. The schematics for the expansion board, and the source code for some of the tests are also included in as appendices. A digital appendix with all submitted and unsubmitted patches for U-Boot, Linux, uClibc, GNU Compiler Collection (GCC) and GNU Binutils also accompanies this report.

Chapter 2

Background

This chapter gives an introduction to the devices, tools, hardware and software relevant to this project. It also describes the fundamental concepts necessary to understand the problems addressed.

The first four sections of this chapter introduce memory management concepts, alignment issues for memory access, and describes SRAM, the memory type of main focus in this report.

Section 2.5 to 2.9 introduces the AVR32 architecture and relevant Atmel products, including the JTAG, the UC3A0512 microcontroller and its sibling, the AP7000.

Section 2.10 introduce file formats for executables and shared libraries we have looked at. An introduction to Linux is given in section 2.11.

The generic introduction to Linux and uClinux is from the earlier project, but the technical details are written for this project. This section is followed by an introduction to the boot loader, Das U-Boot, in section 2.12.

The toolchain that is used for developing Linux for the chip is presented in section 2.13.

The next section, section 2.14 introduces BusyBox, which was used during this project.

The next section, section 2.15 give a short introduction to the networking servers used to support the board during boot and runtime.

A short introduction to open-source collaboration software and principles is given in 2.16. Git, the software system used for revision control of the source code is briefly introduced in section 2.16.1.

Finally, previous related work is presented in section 2.17

2.1 Virtual memory

The contents of this section is based on [19]. Virtual memory is a method for abstracting memory addresses used in programs from their physical addresses. This allows each separate application to have its own private address space, which in turn can be used to enforce memory protection. This is typically implemented with a Memory Management

Unit (MMU). With a virtual memory system, two separate address spaces needs to be considered – the virtual address space, and the physical address space. The physical address space refers to the physical memory, while the virtual address space is per-application.

The MMU's task is to translate virtual addresses to physical addresses. It works by splitting the memory area into separate pages, where each page is a fixed size. A quick survey of the Linux source code shows that typical page sizes for various architectures are 4096 and 8192 bytes.

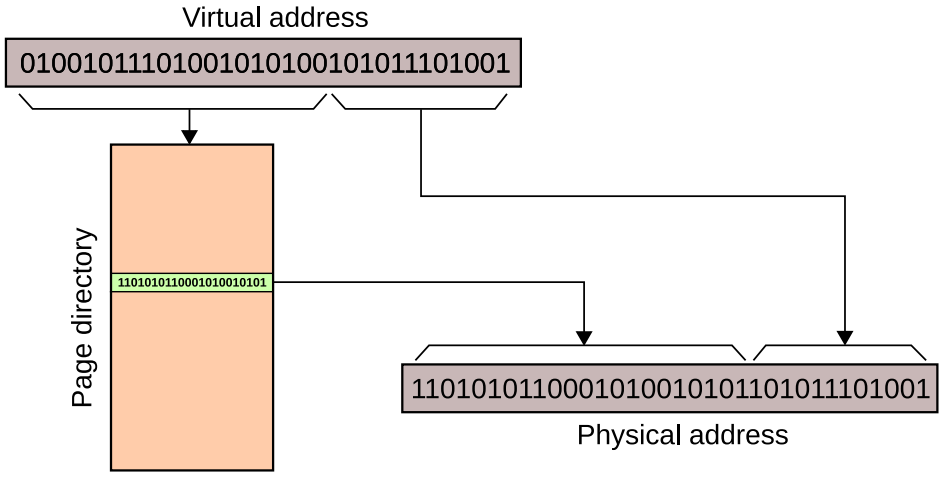


Figure 2.1: Simplified operation of an MMU

Figure 2.1 shows the operation the MMU does when translating a virtual address. It splits the virtual address into two parts – the page number, and the offset into the page. The page number will be looked up in a page directory. The page directory contains the mapping from virtual addresses to physical addresses. The physical address retrieved from the page directory will be combined with the offset into the page to form the physical address.

The page directory contains information about each page, such as whether it is present, and what types of access is allowed to this page. For example, an application can be allowed to read from a page, but not write to it. Invalid accesses to the page will trigger an exception that the operating system can handle.

2.1.1 Copy-on-write

Copy-on-write is a method for saving memory by sharing equal pages between different applications. When two applications load the same part of a file into memory, they can be shared until one of the applications tries to modify it. This is implemented by the operating system by marking the page as read-only when the sharing begins. When one of the applications writes to a read-only page, it will be copied, and the data will be

written to the new copy of the page. Since much of the memory contains code that is never written to, copy-on-write can save a significant amount of memory.

2.2 Memory Protection Unit (MPU)

Without an MMU, all applications must share the same physical address space. If one application is flawed or malicious, the application may read from or write to any memory location. By doing this, the application could potentially sabotage or access any information about the kernel or any process. An MPU[5] provides a way of protecting the processes from each other by having dedicated hardware checking the address of every memory access. The MPU is usually configured by setting up a number of allowed memory areas, and an exception is generated if the application attempts to access memory outside these areas.

Usually, because MPUs are implemented in hardware, only a limited set of allowed/disallowed memory areas can be configured simultaneously. To work around this, it is possible trap the exception, and replace an old memory area with a new new memory area if a memory area not listed in the MPU is accessed. This allows an operating system to support a more or less unlimited set of memory areas.

2.3 Unaligned memory copy

The way processors copy blocks of data from one position in memory to another is vital for performance, and is handled differently depending on the architecture. Some processors can only read and write whole words (32 bits) if they are aligned on word boundaries. Others have optimized hardware instructions for unaligned accesses. Figure 2.2 shows an example of how 10 bytes can be copied between unaligned addresses. Processors that can not perform unaligned accesses must copy these 10 bytes one at a time. If a processor supports halfword copying, the data can be copied one halfword at the time, if both the source and destination address are even or odd. When a processor has support for unaligned accesses, the usual approach for the software is to copy single bytes or halfwords until either the source or the destination are aligned.

2.4 Static Random Access Memory (SRAM)

SRAM, often just called static memory, has a relatively simple memory interface. It consists of n_a address lines, n_d data lines and three control signals. The control lines are a chip enable signal, a read signal and a write signal.

The number of data lines is usually either 8, 16 or 32. It is possible to connect two SRAM chips in parallel to double the number of data bits. For example, by connecting two 8-bit SRAM chips so that they share all lines except the data lines, they will behave like a single 16-bit SRAM chip. See appendix H for an example of this setup.

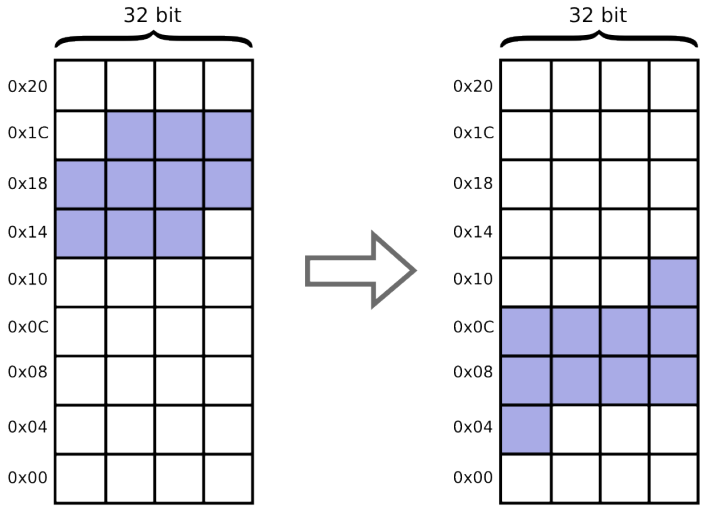


Figure 2.2: Copying of unaligned memory blocks

When the chip enable signal is asserted, a read can be done by placing the address on the address bus, and then setting the read signal. The SRAM chip will then place the requested data on the data lines. Similarly a write can be done by placing the data on the data lines, and the address on the address lines, and then setting the write signal. A read operation is shown in figure 2.3.

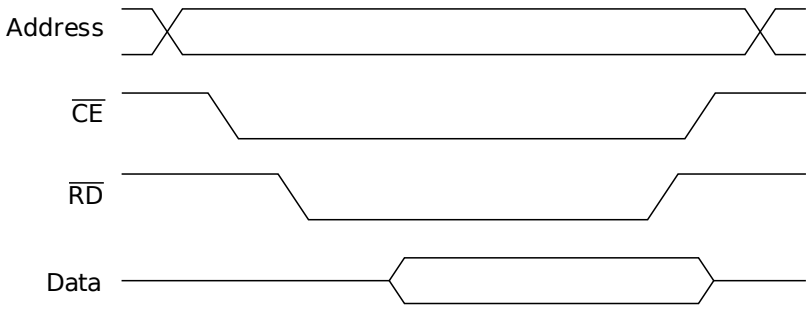


Figure 2.3: Example SRAM read cycle

Reads and writes with SRAM chips are not instantaneous, but require some time to complete. Each SRAM chip has its own specific timing requirements. The requirements define the relationship between the different signals to the SRAM chip. These requirements can for example say that the read signal must be set for at least 7 ns before data will be valid.

2.5 AVR32 Architecture

The contents of this section regarding AVR32 is based on [5], unless otherwise stated.

The AVR32 architecture is a 32 bit load/store RISC architecture by Atmel, designed with emphasis on low power consumption. AVR32 is not binary compatible with 8/16 bit AVR microcontrollers. It was first launched in 2006 with the AVR32 AP core.

The AVR32 architecture defines an optional Java extension module. This module is not available on the microcontroller used during this project, and will therefore not be discussed any further.

2.5.1 Registers

The AVR32 architecture has 16 registers, shown in figure 2.4, with 13 of these being purely general purpose. The remaining three are the program counter, the stack pointer and the link register. The link register is used to hold the return address of the current function. This reduces the amount of stack accesses required for function calls, since simple function calls do not need to access the stack at all. Both the stack pointer and the link register can also be used as general purpose registers.

An interesting feature of the architecture is that all instructions that accept register operands can take any register. This includes the program counter, the link register or the stack pointer. This means that a jump can be implemented in the following way:

```
1 lsl     r10, 2
2 add     pc, pc, r10
```

What this code does is: pc = pc + r10 * 4

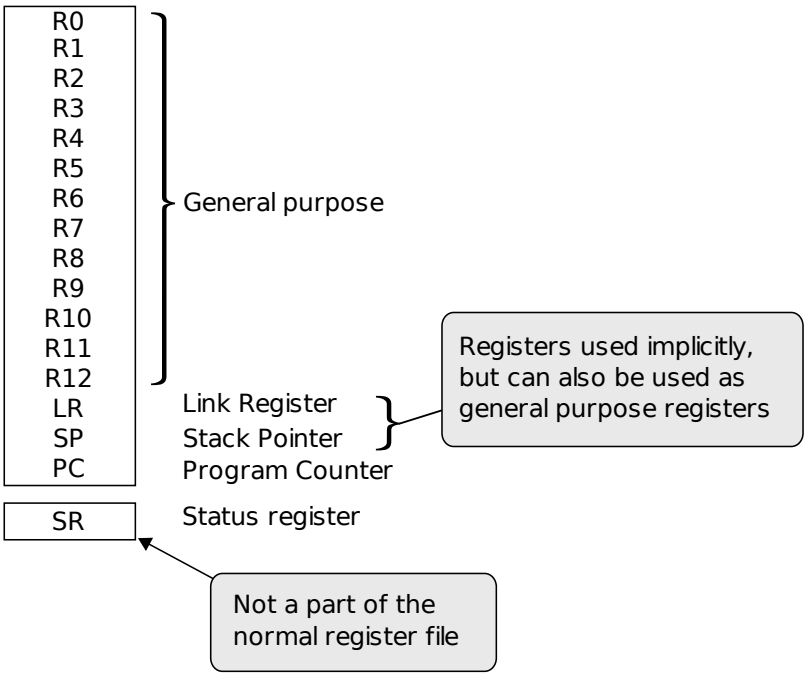


Figure 2.4: Registers in the AVR32 architecture

System registers

In addition to the normal registers there are a large number of system registers. Most of these are used for accessing the configuration and status of various features on the processor. Exception vectors, MMU and MPU are examples of features that can be configured with these registers.

One of the system registers is the status register. This register is shown in figure 2.5. It is split into two parts – the upper and lower halfword. User applications can only access the lower halfword.

The lower halfword contains several flags set by results of arithmetic and logical operations, such as a zero flag, an overflow flag, and several others. These flags are used by conditional branches and operations. The lock-bit is used to implement atomic operations, the scratch bit can be used for any purpose by applications, and the register remap flag is used by the Java extension module.

The upper halfword contains the status of the processor. Among other things this includes the current execution mode and whether interrupts and exceptions are enabled.

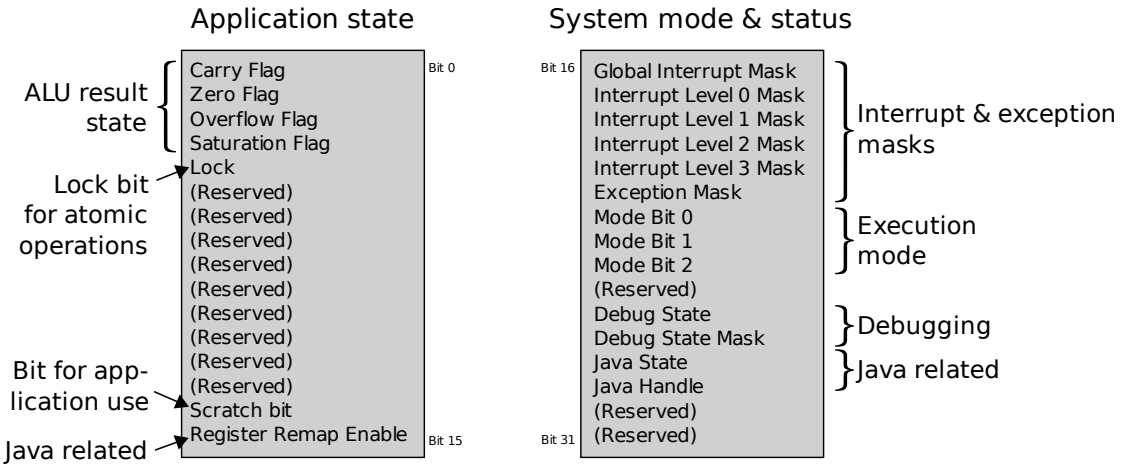


Figure 2.5: The AVR32 status register

Register shadowing

Another feature of the AVR32 architecture is register shadowing. When the processor changes to an interrupt execution mode (see 2.5.5), it may replace some part of the register file with one reserved for that mode. Also, whenever the CPU changes from application mode, the user-mode stack-pointer is replaced with a system stack pointer.

There are three levels of shadowing: `small`, `half` and `full`. In mode `small`, no general purpose registers are shadowed. In mode `half`, registers r8 to r12 and the link register are shadowed. With `full`, registers r0 to r12 and the link register are shadowed. This is illustrated in figure 2.6.

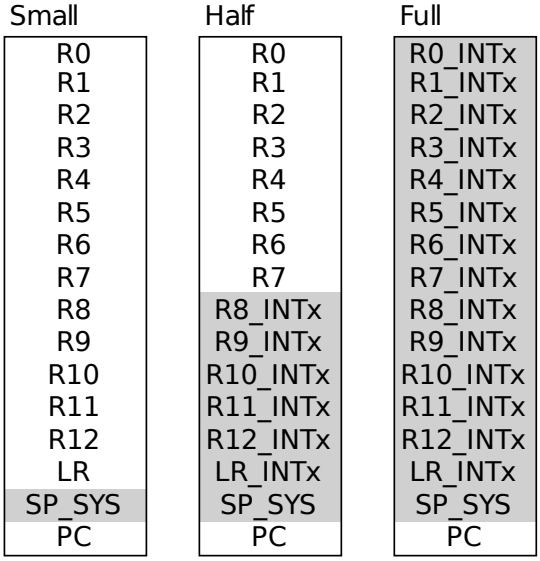


Figure 2.6: Register shadowing in the AVR32 architecture

This feature makes it possible to handle some interrupts without having to access memory. If the registers are not shadowed, they must be saved to the stack before being used. If this is not done, the interrupt handler may overwrite or change registers in use by a running application. This may then lead to incorrect execution of the application.

2.5.2 Instructions

The AVR32 architecture specification defines 214 instructions. Each instruction in the AVR32 architecture is either two or four bytes wide. Many instructions have multiple different encodings. For example, some instructions has a two-byte encoding for small immediate values, and a four-byte encoding for larger immediate values. Also, some instructions may take one or two operands. For example, the ADD instruction has a two-byte variant with `Rd = Rd + Rs`, and a four-byte variant with `Rd = Ra + (Rb << shift)`.

2.5.3 Sub-architectures

There are two different sub-architectures of the AVR32 architecture; AVR32A and AVR32B. Figure 2.7 shows the structure of the AVR32 sub-architecture hierarchy, and the UC3A0512 and AP7000 microcontrollers are shown as examples of implementations. AVR32A targets cost sensitive, lower-end applications and AVR32B targets applications where low interrupt latency is important. Since the AVR32A architecture is simpler than the AVR32B architecture, the hardware implementation of it is simpler, and lower power consumption is achievable.

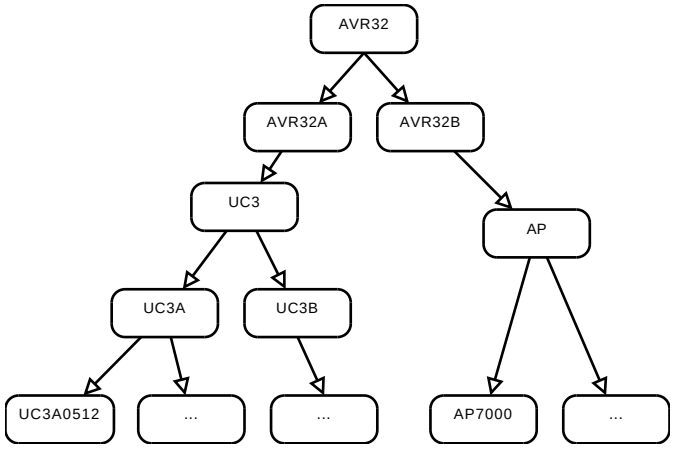


Figure 2.7: Relationships between the AVR32 architectures and implementations

The main difference between AVR32A and AVR32B is the method of interrupt and exception handling. The difference is in the way the state is saved and restored when control is transferred to and from the exception and interrupt handlers. This topic will be discussed in more detail in section 2.5.6. Because the AVR32B architecture is

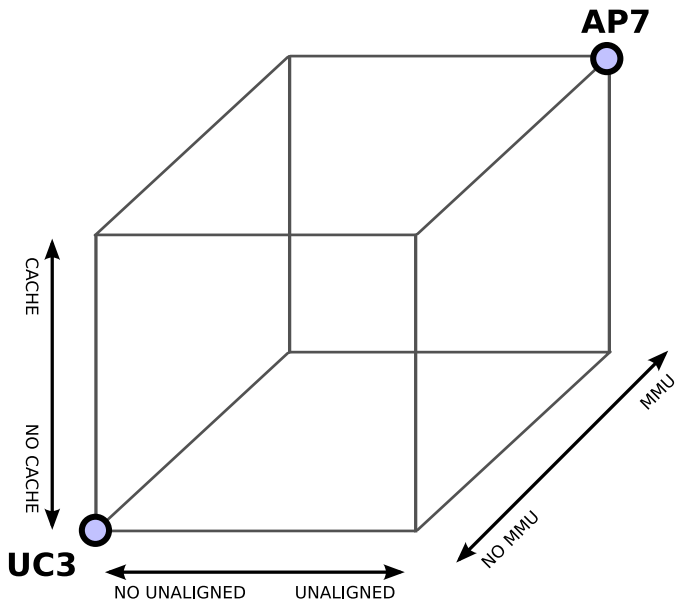


Figure 2.8: AVR32 feature variations [8][7]

focused on interrupt latency, dedicated registers are implemented for holding the status register and return address for interrupts, exceptions and supervisor calls. The AVR32A architecture also does not implement register shadowing of any registers except for the stack pointer.

Three properties that are not defined by the sub-architecture, are the presence of cache and MMU, and the ability to perform unaligned memory access. These properties are defined by the implementation of the CPU core. Figure 2.8 shows how these three properties can vary and potentially be implemented in eight different combinations. As depicted in the figure, AP7 implements all of these features, and the UC3 none. This difference may be significant in the adaptation of the Linux kernel. For a comparison of the AP7 and UC3 implementations AP7000 and UC3A0512, see section 2.7.4.

2.5.4 Revisions

There are two revisions of the AVR32 instruction set, revision 1 and revision 2. Revision 2 introduces 15 new instructions. One of these instructions load an immediate value into the upper halfword of a register. The others are for conditional operations, such as conditional loads and stores and conditional add, sub, and such instructions.

2.5.5 Execution modes

The AVR32 architecture defines eight different execution modes. These are:

- Application mode
- Supervisor mode

- Four interrupt levels
- Exception mode
- Non-maskable interrupt

The mode can be changed by executing certain instructions, or as a result of signals from events occurring outside the CPU core (interrupts, exceptions etc). Each mode has a designated priority that determines whether execution can be interrupted to switch to another mode. In other words, execution in a mode with a certain priority will be interrupted if an event occurs that is handled in a mode with higher priority.

When running in application mode, the processor restricts access to various system registers, and the top half of the status register. This makes it possible to prevent applications from tampering with the CPU state and the execution of the kernel.

2.5.6 Exception and interrupt handling

There are two sources of "breaks" in the instruction flow. These are interrupts and exceptions. Interrupts are typically external events, while exceptions are internal events.

The AVR32 architecture implements exception handling by jumping to specific addresses when an exception occurs. The base address of the exceptions is configurable through a system register. There are four bytes between most exceptions, which is big enough for a jump instruction. Some of the more performance critical exceptions have more space between them. These are those that deal with updating the Translation Lookaside Buffer (TLB) on systems with MMU.

Interrupts in the AVR32 architecture are handled mostly in the same way as exceptions. However, the jump offset is configurable. The jump offset for each interrupt group can be set to an offset based on the exception vector.

The process for handling interrupts or exceptions varies between the AVR32A and AVR32B architecture. The AVR32B architecture has extra system registers for saving the return address and status register for each execution mode. The AVR32A architecture on the other hand does not have those extra registers. Instead, they are pushed onto the stack.

2.6 The AP7000 microcontroller

The AP7000, released in 2006, was the first microcontroller that implemented the AVR32 architecture. It was based on a new CPU core, named AP7. "AP" stands for Application Processor, and the microcontroller was meant for network and multimedia applications.

The AP7000 microcontroller runs at up to 150 MHz, and can execute 210 Dhrystone MIPS at that speed. The AP7 core implements a 7-stage pipeline with three subpipes – the multiply, the execute and the data pipe. Instructions are issued in-order, but can be completed out-of-order.

The AP7 core implements the more complex of the two AVR32 sub-architectures – the AVR32B architecture. It has separate instruction and data caches, each of them 16 KB. It also has support for a MMU, with a 32-entry TLB.

In addition to the CPU core, this microcontroller has a number of on-chip peripherals, including serial ports, Ethernet Media Access Controller (MAC)s, USB, and several others. U-Boot and Linux has been ported to this microcontroller previously, more about this in section 2.17.

2.7 The UC3A0512 microcontroller

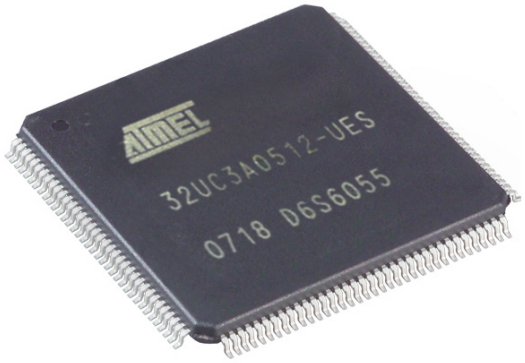


Figure 2.9: The UC3A0512 chip

This section is based on [8]. The AT32UC3A0512 (figure 2.9) microcontroller is part of a new product line released by Atmel in 2007. These microcontrollers were based on a new core – the UC3 core. The UC3 core is the first CPU core based on the AVR32A architecture. It is used in two series of microcontrollers – the UC3A series and the UC3B series. The main differences between the two series of microcontrollers are what on-chip peripherals are available. The UC3A series is the most feature-rich of the two series[6], described as "Communication Family". The UC3B series lacks three features that are found in the UC3A series. This is the external memory interface, the Ethernet interface, and the Audio Bitstream DAC.

The UC3A series focus on high performance, low power 32-bit microcontrollers and is aimed at several industrial and commercial applications including PLCs, instrumentation, phones, vending machines and more.[6]

2.7.1 Logical layout

Figure 2.10 shows the logical layout of the UC3A0512 microcontroller. There are four internal buses in the microcontroller, including the CPU local bus. Because of a bug in the chip, the CPU local bus was never used during this project, and it has been excluded from the figure.

The High Speed Bus Matrix (HSB) is the main bus of the chip. It is implemented as a many-to-many connector with a number of bus masters and slaves. Each bus master can read or write data to any of the slaves. Each slave is responsible for a subset of

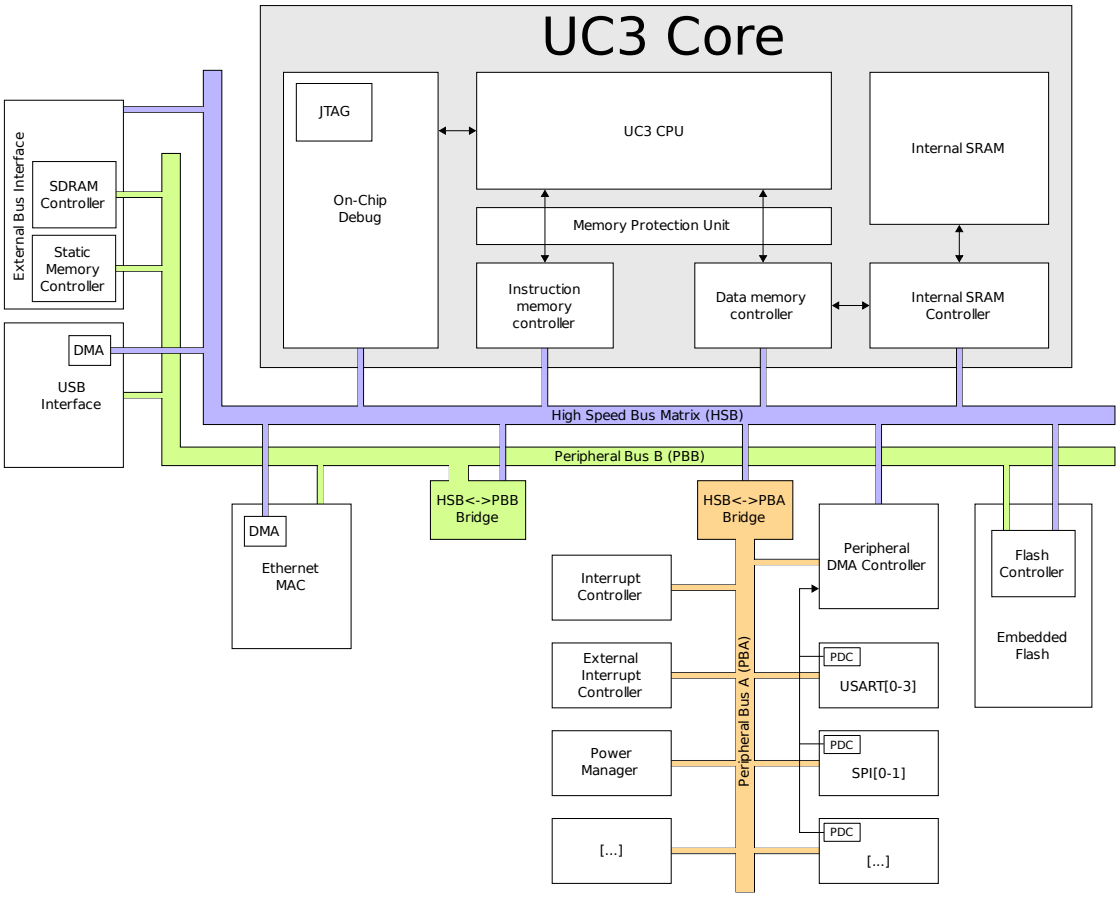


Figure 2.10: UC3A0512 microcontroller (based on several figures in [8])

the address space. With one exception, all memory access has to go through the HSB. The exception is that there is a shortcut for the CPU core to the internal SRAM, which permits single-cycle access to the internal SRAM.

The following devices are connected to the HSB:

- Ethernet MAC (master): Reads and writes packets to memory.
- USB (master): Reads and writes packets to memory.
- USB (slave): Allows access to the packet buffers in the USB interface.
- EBI (slave): Allows access memory connected connected to the EBI bus.
- Flash (slave): Allows access to the internal flash memory.
- Peripheral DMA controller (master): Reads and writes data from/to peripheral devices.
- OCD (master): Allows debugger reads and writes to different peripherals and memory banks.
- CPU instruction (master): Reads CPU instructions.
- CPU data (master): Reads and writes data to different memory banks.
- Internal SRAM (slave): Allows access to the internal SRAM on the chip.

There are two peripheral buses: Peripheral Bus A (PBA) and Peripheral Bus B (PBB). The different peripherals are connected to the peripheral buses, which in turn are connected to the High Speed Bus (HSB) through bridges.

Note that the bridges act as slave devices on the HSB. The bridge itself is the sole master device on the peripheral bus. This means that it is impossible for devices only connected to the peripheral buses to access main memory.

The different devices expose data and configuration registers on the peripheral bus. These can be read and written by the CPU, or any other device able to act as a master on the HSB bus.

The Peripheral DMA Controller (PDC) is a device which can copy between data registers of different devices and main memory. This allows the CPU to offload the work required to receive and send data via the data registers of the devices. The different devices signal the PDC when they are able to receive or send more data. The PDC will then either copy data from main memory to the data register, or copy data from the data register to main memory. The PDC is connected to the HSB, and is able to access both main memory and the devices data registers through this bus.

2.7.2 Features

There are a number of features on the UC3A0512 microcontroller which may be of interest to us. In this section we will introduce them, and discuss why they may be of interest, and how we can use them.

Internal flash

The UC3A0512 microcontroller has 512 KB of internal flash (hence the "512" in its name). The microcontroller starts execution at address 0x80000000, which is the starting address of the internal flash. Therefore, the internal flash has to be programmed to be able to use this microcontroller. Different memories are mapped to separate areas in the physical address space available. Figure 2.11 shows the layout of the physical address space in the UC3A0512. Note that the memory areas may be missing or have different sizes in other UC3A microcontrollers (UC3Axxxx).

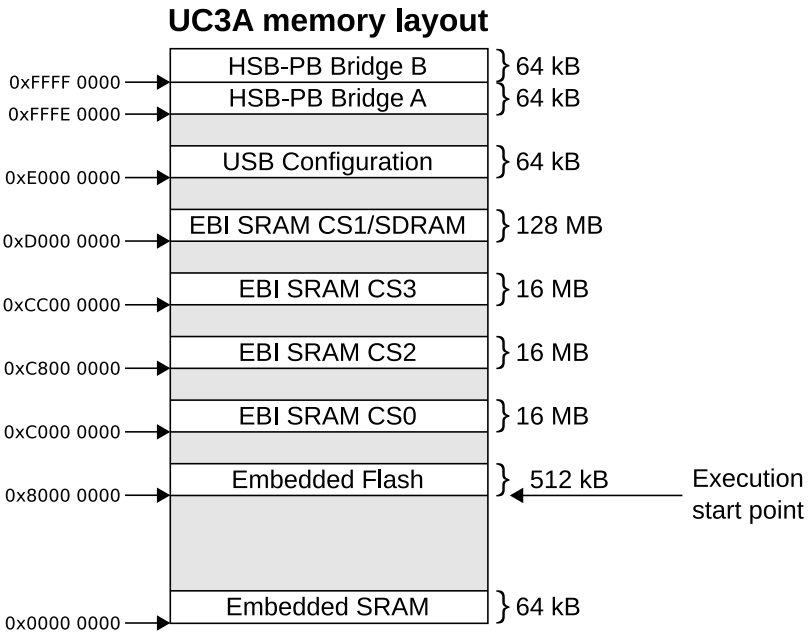


Figure 2.11: UC3A0512 physical memory map

Since the internal flash is a non-volatile internal memory of convenient size, it is a good option for an area to store the boot loader in. It could also potentially be used to store the Linux kernel, if we could get the kernel small enough to fit in the available space. The U-Boot boot loader (see section 2.12), uses the internal flash to save configuration options and user settings.

Flash is written in whole pages, and a page must be erased before a new can be written. On the UC3A0512, each flash page is 512 bytes. To write a flash page, data is added to a page buffer, and a write command is issued.

Internal SRAM

There are 64 KB of internal SRAM available on the UC3A0512. This is the only Random Access Memory (RAM) guaranteed to be available at start-up. Accesses to the internal SRAM take a single cycle to complete, and each access is word-sized (32 bits).

Since it is the only RAM guaranteed to be available at start-up, using it for the boot loader is a natural choice. The datasheet recommends to use it for the system stack, since it is the fastest memory available on the chip. Due to the multi-threaded nature of the Linux kernel, this may be difficult – if not impossible – to accomplish. Therefore we consider this to have a low priority.

External Bus Interface (EBI)

The EBI is a coordinator for a collection of Input/Output (IO) lines, and ensures successful data transfer between external devices and the microcontroller. The EBI has one Synchronous Dynamic Random Access Memory (SDRAM) controller and one SRAM controller muxed on a common output. These are together capable of handling several types of of external memory and devices, such as SRAM, Programmable Read-Only Memory (PROM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), Flash, and SDRAM. The EBI is capable of simultaneously handling transfers with up to four external devices with Static Memory Controller (SMC) interface[1]. One of these four "channels" can be set to SDRAM mode. Each device is memory mapped in its own address space.

Technical data:

- 16 bit data bus
- 24 bit address bus
- Four chip select lines
- Several control pins

The data and address lines are shared between the different memory controllers, and some of the control lines are also shared.

External SDRAM

The SDRAM controller in the EBI supports 2 or 4 banks, with up 8192 rows and up to 2048 columns per bank. Each access can be for either 16 or 32 bits. The total amount of SDRAM is limited to 128 MB by the size of the memory segment reserved in the physical memory map of the microcontroller.

There is a bug in the SDRAM controller in all current revisions of the UC3A0512 (see 2.7.3). This bug makes running code from SDRAM unreliable. There is currently no workaround for this problem, and leaves SRAM and internal or external flash as the only memories from which these chips can execute code from. The properties and capabilities of SDRAM will therefore not be described in detail.

[1] According to the datasheet it should be five. Based on the address memory map, we believe this is wrong.

External SRAM

External SRAM is interesting to us because of the SDRAM bug mentioned in section 2.7.2. Because the SDRAM controller is faulty, the only way to get enough memory to run U-Boot and Linux is to use RAM compatible with the SMC in the EBI. For this reason, Atmel provided us with a memory expansion card with SRAM and flash during previous work. This is why SRAM is used as the main memory throughout this project. The expansion card is described in section 2.17.4.

External SRAM can be connected to the UC3A0512 by using the SMC interface in the EBI. In the SRAM interface on the UC3A0512, the control lines are active low. See section 2.4 for a general introduction to SRAM. There are four chip-selects available, each of which allows for up to 16 MB of SRAM to be connected. The total possible amount of SRAM is therefore 64 MB.

On the UC3A0512, the SRAM timings are configured by specifying the waveforms for the different control signals. Read and write cycles have separate configurations, and can have entirely different timings, including the total length of the read or write cycle. For each read or write cycle there are five configuration values, and together they describe the total cycle. These are shown in figure 2.12.

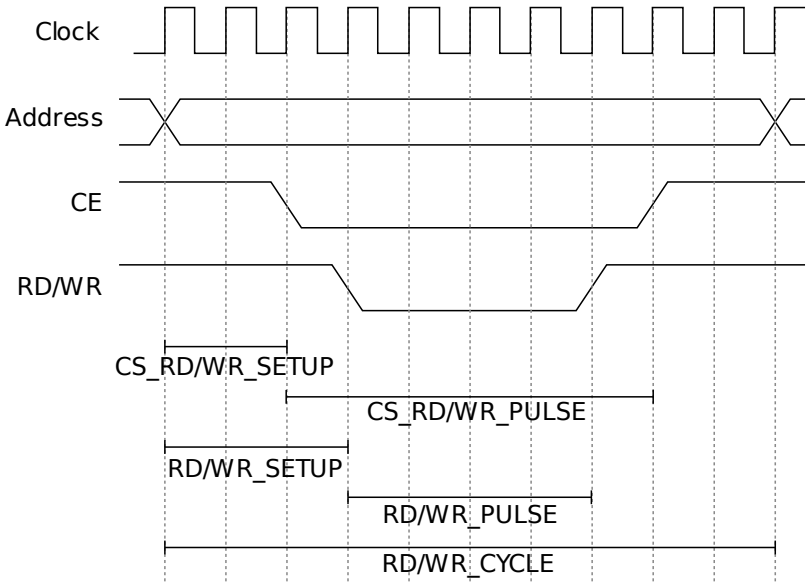


Figure 2.12: UC3A0512 SRAM timing configuration

Network Media Access Controller (MAC)

Both the AP7000 and the UC3A0512 microcontrollers have on-chip Ethernet controllers called MACs. The specific MAC implementation in the UC3A0512 is by Atmel given the name MACB. In this report we will use the abbreviation MACB when referring to Atmel's implementation. The MACB can be used in conjunction with an external chip

to provide Ethernet connectivity. The external chip that handles the physical layer of the Ethernet connection is called a PHY. Different PHY chips provides different physical layers, for example Ethernet over copper wires and Ethernet over fiber.

The PHY is connected to the Ethernet controller through either a Media Independent Interface (MII) or Reduced Media Independent Interface (RMII) bus. The MII bus requires 17 wires, while the RMII bus requires 10 wires. 4 wires are used in each direction for data transfers with a MII bus, while 2 wires are used with a RMII bus. To transmit 100 Mbit per second, a MII bus requires a clock rate of 25 MHz, while a RMII bus requires a 50 MHz clock.

In both the MII bus and the RMII bus, two lines are used for a management of the PHY. These two lines can be shared between multiple PHYs, though this feature is not used by the AP7000 or UC3A0512 microcontrollers. To allow for sharing of the management bus, each PHY has its own address. There are 32 different addresses, allowing for up to 32 PHYs to share once management bus. Figure 2.13 show the communication lines between the microcontroller and the PHY respectively.

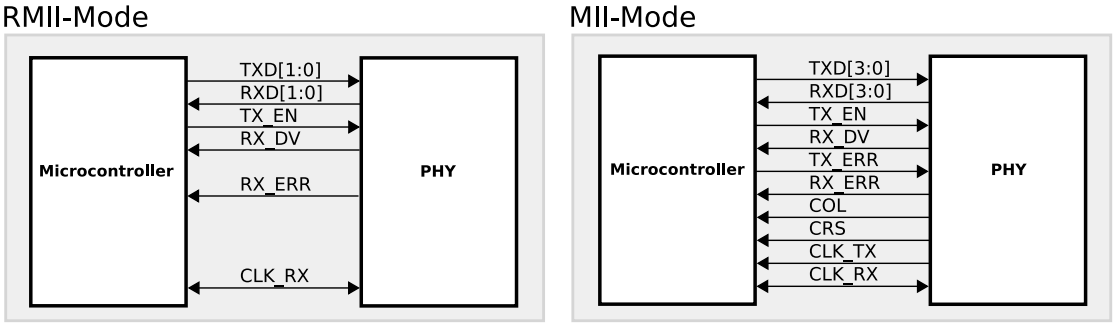


Figure 2.13: RMII and MII connection

Serial Peripheral Interface (SPI)

The microcontroller has several SPI interfaces. SPI is a full duplex synchronous serial data link for communicating with external peripherals or devices. SPI is a de facto standard that typically uses four wires[8]: one for each direction of data, one clock line and one chip selection line for every slave. Figure 2.14 shows how multiple devices on the EVK1100 are connected to the microcontroller. The EVK1100 will be introduced further in section 2.8. If all the slaves support “daisy chaining”, the slaves can be connected in a loop and share the chip selection line. Daisy chaining is not relevant for this project and will not be discussed any further.

SPI support and utilization is listed as a requirement 1 and 6e in section 1.3.1.

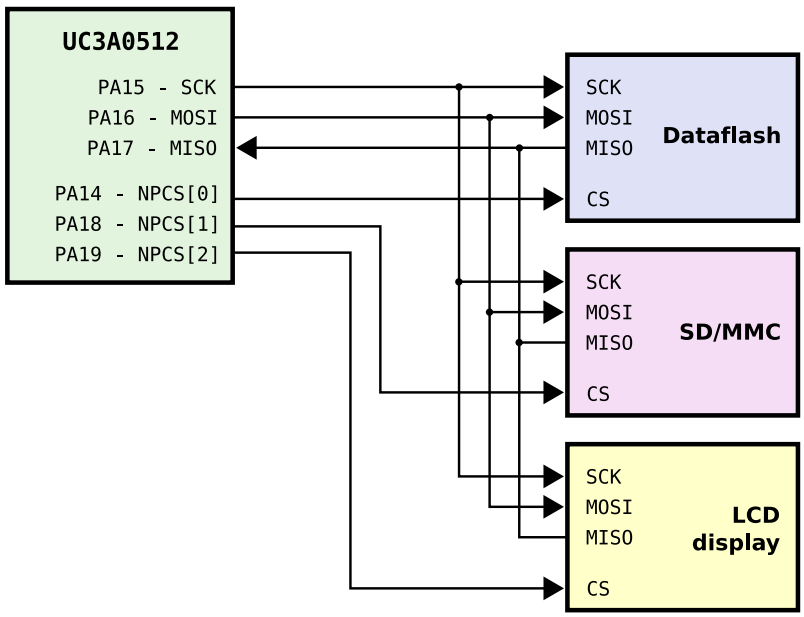


Figure 2.14: SPI connections on the EVK1100

General Purpose Input/Output (GPIO) controller

There is a GPIO controller on the UC3A0512 that controls the output of most of the pins on the chip. Different functions can be selected for each physical pin. In addition to the GPIO function of the pins, up to three other functions for different peripherals can be selected. The GPIO function of a pin is set by enabling the pin as a GPIO pin. The peripheral function can be set by disabling the GPIO function of the pin, and then selecting a peripheral function.

We are mainly interested in the peripheral function of the pins, and selecting which peripheral function each pin should employ. Typical peripheral functions we are interested in are the EBI bus and the serial port. There are also some LEDs, buttons and a joystick connected to the chip, which can be used through the GPIO controller. The LEDs may be useful, for example as indicators of the system state, or status lights during boot.

Universal Serial Bus (USB)

The UC3A0512 has a built in USB interface that is able act both as a device and a host (not simultaneously). It has a built-in PHY that takes care of the transmission on the physical medium, so the only external components required for USB to work are a few resistors, and optionally a connector. The USB unit on the chip has been extended with USB On-The-Go support since the AP7000 was made. USB On-The-Go is a relatively new standard that combines lower power requirements and a small form factor for connectors and cables with host capability and dynamic switching between

host and peripheral mode[24].

Support for USB under Linux would definitively be a useful feature. Linux already supports the internal USB interface in the AP7000[1], but the driver needs to be tested on the UC3A0512 microcontroller, and adapted if necessary. Because the UC3A series also have the On-The-Go capability, the driver may need significant extensions and changes in order to work.

2.7.3 Chip revisions

The microcontroller that will be used during this project is an engineering sample. The part number printed on the package of engineering samples of UC3 microcontrollers are suffixed ES, which makes the part number UC3A0512ES. There are several highly significant design errors in this particular revision of the chip. Design errors that may be relevant to this project are listed here. These can be found in the errata list in [8]. Note that the errata numbering is based on the numbers found in revision F (08/08) of the datasheet. The numbers may change between revisions of the datasheet, as more errata are added.

- SPI interface bugs (erratum 41.4.1)
- Two NOPs needed after instructions masking interrupts (erratum 41.4.5.5)
- Processor reports wrong processor ID (erratum 41.4.5.1)
- Bus error during debug mode causes processor to stop responding to debug commands (erratum 41.4.5.2)
- CPU cannot operate on a divided slow clock (erratum 41.4.5.12)
- Code execution from external SDRAM does not work (erratum 41.4.6.1)
- Memory Protection Unit is not functional (erratum 41.4.5.7)
- Peripheral Bus A maximum frequency is 33MHz instead of 66MHz (erratum 41.4.8.4)
- On some rare parts, the maximum HSB and CPU is 50MHz instead of 66MHz (erratum 41.4.8.6)
- Corrupted read in flash after FLASHC WP, EP, EA, WUP, EUP commands may happen (erratum 41.4.12.4)
- Stalled memory access instruction write-back fails if followed by a HW breakpoint (erratum 41.4.14.1)

2.7.4 AP7000 versus UC3A0512

Table 2.1 shows a rough comparison between the AP7000 and the UC3A0512. One very important difference is that the UC3A0512 does not have an MMU. It also has a shorter pipeline and does not have any cache.

A significant difference between the AP7000 and the UC3A0512 is that the UC3A0512 does not support unaligned memory reading or writing. In the example in figure 2.2,

Feature	**AP7000**	**UC3A0512**
Cache	YES	NO
Frequency	150MHz	66MHz
Pipeline	7-stage	3-stage
Dhrystone MIPS	210@150MHz	91@66MHz
Internal SRAM	32kB	64kB
Internal Flash	0kB	512kB
Unaligned memory access	YES	NO
Memory Management Unit	YES	NO
Memory Protection Unit	NO	YES
Ethernet MAC 10/100	YES	YES
Java Hardware Acceleration	YES	NO
Read-Modify-Write instructions	NO	YES

Table 2.1: AP7000 vs UC3A0512 comparison table[7, 8, 9]

the UC3A0512 is capable of per-halfword copying everything except for the first and the last byte. The UC3A0512 is capable of doing halfword copying like this whenever both input addresses are odd or even.

There are also several other similarities and differences in the available peripherals. Without going into much detail, we can mention the following:

- The power manager, which is responsible for clock generation to various peripherals, is mostly the same.
- The interrupt controller is the same.
- The Ethernet MAC controller is the same.
- The Parallel Input/Output controller on the AP7000 has been replaced with a General Purpose Input/Output controller
- Several peripherals are not included on the UC3A0512, such as the MultiMediaCard interface, the LCD controller, the PS/2 module.

2.8 EVK1100

The EVK1100 is an evaluation kit for the UC3A microcontroller series, and can be seen in figure 2.16. Like most Atmel products, the name is often prefixed with “AT” (AT-EVK1100), but the name EVK1100 will be used throughout this report. The EVK1100 and other development/evaluation kits mentioned in this report will often simply be referred to as “boards”. The EVK1100 has a UC3A0512 microcontroller and several devices, peripherals and connectors. The features of the EVK1100 that are most important in regards to this project are the clocks, the serial port and the Ethernet peripherals and connectors. Figure 2.15 shows a simplified block diagram of the organization of the

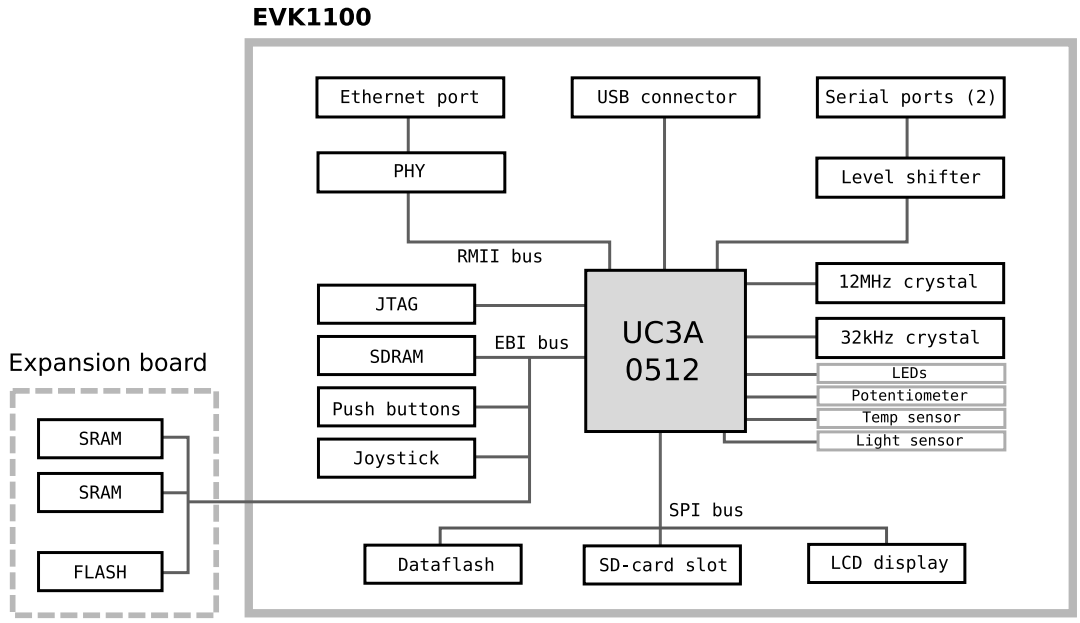


Figure 2.15: The main components of the EVK1100

most central components of the EVK1100. The figure also illustrates how some devices and peripherals share the same bus lines.

The three buses labeled in this figure are introduced in section 2.7.2. The network controller is a DP83848I from National Semiconductor and is connected to the microcontroller via an RMII bus. This is the same PHY as on Atmel's NGW100 network gateway kit, which is already supported by U-Boot and Linux. The RMII connection bus and the impacts of using it is explained in detail in section 2.7.2.

The JTAG connector is essential for programming and debugging. Power connectors and regulators are obviously also a necessity.

Located on the underside of the EVK1100 is a 32MB SDRAM chip. This memory was never used during this project due to a bug in the microcontroller (described in section 2.7.3). In newer revisions of the microcontroller, when this bug is corrected, the SDRAM will be very useful. The evaluation kit also has many other interesting and useful features including a Liquid Crystal Display (LCD) display, a Secure Digital (SD) card slot, LEDs, microswitches, a potentiometer, a joystick and a light sensor.

Figure 2.16: The EVK1100 evaluation kit with expansion board (right), connected to the JTAGICE MKII (left)

2.9 JTAG

JTAG is a hardware interface that provides a "back door" into a system for testing, analyzing behavior and debugging. Atmel's own JTAG device, called JTAGICE MKII, was at our disposal. The JTAGICE MKII supports On-chip debugging of all AVR and AVR32 microcontrollers with IEEE 1149.1 compliant JTAG interface. It is connected to a computer using either a serial or USB cable and is supported by both Windows and Linux. Under Linux, a command line application named `avr32program` is used to program microcontrollers via the JTAGICE MKII, and `avr32gdbproxy` enables a proxy for debugging with GNU Debugger. For more about GNU Debugger (GDB), see section 2.13.7.

2.10 Binary formats

This section will introduce the file formats for executables and shared libraries we have looked at. The Linux kernel has support for the following binary formats:

- Executable and Linkable Format (ELF)
- Function Descriptor Position Independent Code (FDPIC) ELF
- Flat
- A.out
- SOM

We focused on the ELF format already supported by the AVR32B architecture, and its derivative, the FDPIC ELF format. The FDPIC ELF format is a variant of the ELF format, and is designed to run on MMU-less systems.

We also looked into the Flat format, which is a simple binary format for MMU-less systems.

2.10.1 Terminology

In this section we will introduce the terminology we use to describe binary formats. An overview of the terminology is shown in figure 2.17.

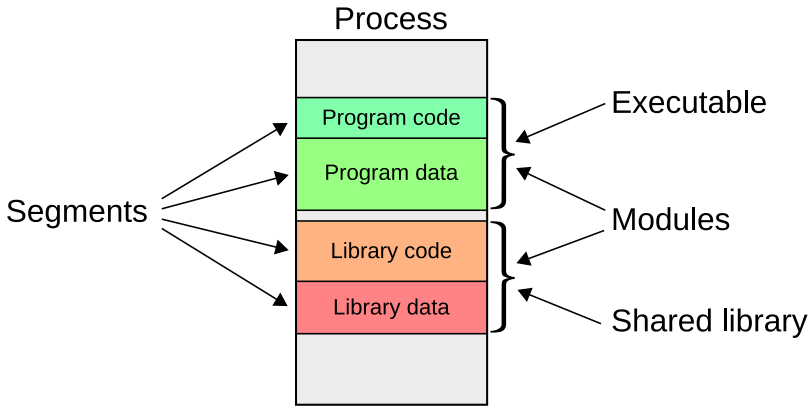


Figure 2.17: Terminology for binary formats

Process

A running instance of an executable. This is typically an application which is loaded into memory together with all shared libraries used by it.

Executable

An executable refers to a program file, which can be loaded into memory and executed.

Shared library

A shared library is a file with code and data which can be shared between several processes. A typical example of a shared library is the C library, which provides programs with the standard C functions, such as `printf`, `exit` and `sleep`. A program can link to several shared libraries, and each shared library can link to other shared libraries.

Module

A module refers to a single loadable file with code and data. It can either be a shared library or an executable.

Segment

This is a loadable part of a module. It refers to a block of code and/or data. This block is loaded at a specific address, and will have specific access rights. Typical examples are code-segments, which are readable and executable, and data segments, which are readable and writable.

On some architectures there are advantages to a more fine-grained separation, with separation into three segments: code, read-only data and read-write data. If the processor supports it, these segments can be given different permissions, so that data can never be executed. However, on systems without MMU or MPU, such separation does not increase security, since there is no memory protection.

2.10.2 ELF

The ELF[16] format was developed by UNIX System Laboratories, and in 1999 it became the standard format for Unix-like systems, possibly due to the 86open project group's effort. The 86open project group was formed in 1997 to discuss the need for a standard binary executable for x86 based unix systems. This project group were dissolved when the vendors orginally forming the group had chosen the Linux ELF format.

The ELF format is described in the System V ABI specification[21]. ELF files are usually created by the assembler and linker during the build process. ELF files can provide either a link view (2.19(a)), an executable view (2.19(b)) or both. The link view is used when the linker combines several ELF files into one ELF file, while the executable view is used when loading an ELF file for execution. The executable view is used in both shared libraries (called shared objects in ELF), and executables.

There are three main types of ELF object files:

- **Executable file.** These are the program files that are loaded by the operating system.
- **Shared object file**, which are the ELF shared libraries. These files are usually loaded by the dynamic linker when a program is executed, but can also be linked into an executable file while compiling the executable.
- **Relocatable file.** These files are intermediary files used when compiling programs. It is typically the compiler which generates these files, and then the linker will combine several of them into the final executable.

Figure 2.19 is an example of an ELF file. This example is based on an statically linked version of BusyBox. It shows the relationship between sections and segments. In this example there is an additional segment defined in the program header which is not shown, namely the stack. The stack is given with a filesize set to zero, but a memsize set to the wanted stack size. This is used to tell the loader to allocate memory for the stack when the program is loaded.

Figure 2.18: ELF Views

ELF Header
Program header table *optional*
Section 1
...
Section header table

(a) Link

ELF Header
Program header table
Segment 1
...
Section header table *optional*

(b) Executable view

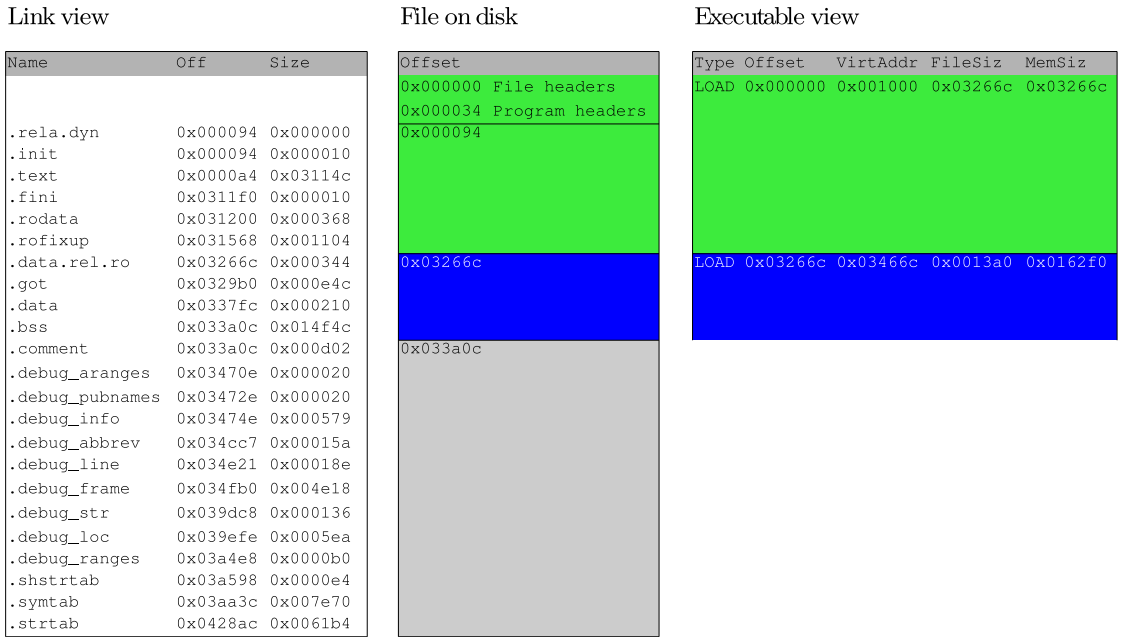


Figure 2.19: Object example (BusyBox)

Sections and segments

The sections are generated by the compilers, and form the individual parts of the relocatable file. Code, read-write data, read-only data, relocation info, debugging info and various other information is stored in individual sections. The linker will take equal sections from the all of its input files, and combine into one larger section. For example, it will combine all the sections with code into one big section. Later, the sections with equal access permissions (i.e. read-only, executable, read-write) will be combined into segments for the executable view.

Program header

The program header is a list of the segments in the file. For each segment, it contains a description of the segment, which will be used to load the segment into memory. The elements in this description is listed below:

- `type` tells what kind of segment it is.
- `offset` is the start address within the file for the segment
- `vaddr` is the virtual address where the segment should be located within memory
- `paddr` is reserved for the segments physical address for systems where it is needed.
- `filesz` is the size used by the segment in the file (may be zero).
- `memsz` is the size used by the segment in memory (may be zero)
- `flags` contains permission (read, write, execute) for the segment
- `align` contains the alignment necessary for this segment.

The memory size of a section may be larger than its file size. The remaining bytes in the section will then be padded with zero-bytes. This is used by data segments where variables are initialized to zero, and thus provides a simple way of saving disk space.

Loading and execution

The Linux kernel will identify an ELF file based on the first four bytes of the file. These bytes are `{0x7f, 'E', 'L', 'F'}`. Once the file is identified as an ELF file, the kernel will find the program header, and iterate over the segments listed there. Each segment will be loaded according to the descriptions in the headers. When all the segments are loaded, the kernel will transfer control to the new program.

Dynamic linking

An ELF binary using dynamic linking has a special program header that indicates which dynamic linker should be used. The dynamic linker is a program that knows how to load shared libraries, and link the executable with them at runtime. The Linux kernel will load both the original program, and the dynamic linker. Instead of passing control to the original program, the dynamic linker will be executed first. The dynamic linker will then do the actual loading and linking of the shared libraries.

Shared libraries and programs which use dynamic linking contains a segment with information for the dynamic linker. This is known as the `DYNAMIC` section. The `DYNAMIC` section contains relocation information, information about shared functions, and information about libraries used.

The dynamic linker will use this information to locate the libraries the program should use. It will then load those libraries. Sometimes the dynamic linker is unable to load the libraries at exactly the address they have requested in the program headers. In those cases it will use the relocation information stored in the `DYNAMIC` section of those libraries to relocate the library to a different address.

The program needs to be able to access functions and data in the shared libraries. Information about what functions and data is used is stored in the `DYNAMIC` section. The dynamic linker will find the parts of the program that needs to be updated to access the functions and data, and insert the correct reference.

2.10.3 FDPIC ELF

The FDPIC ELF format is an adaption of the ELF format. Its purpose is to be able to execute ELF files on platforms without MMU support. Our main source of information about the FDPIC ELF format was [13].

Memory layout

FDPIC ELF files can be loaded into memory in two different ways. If the file has a constant-displacement flag set, all the segments in the file will be loaded into one contiguous block of memory. If the constant-displacement flag is unset, each segment will be loaded separately.

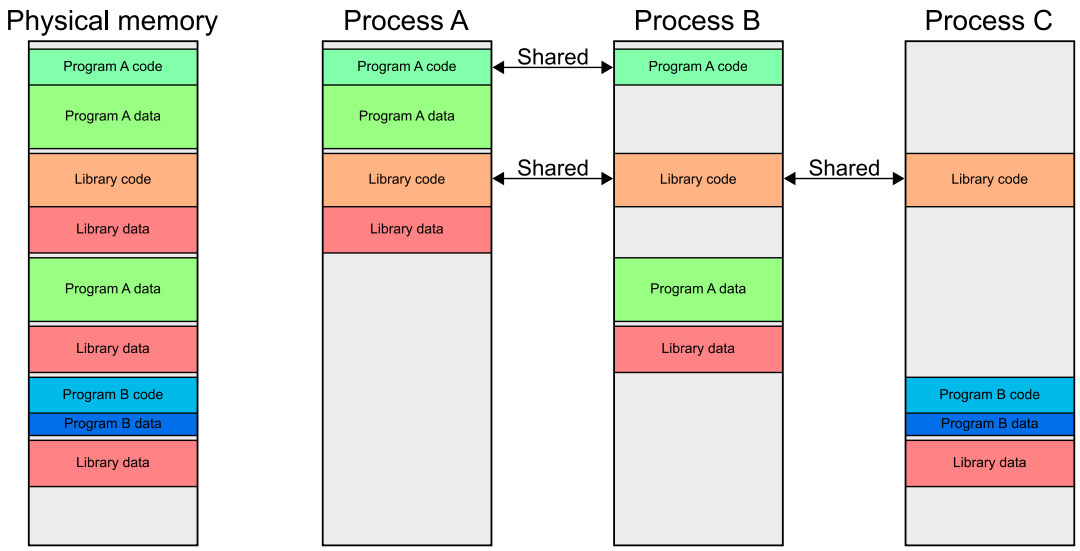


Figure 2.20: Memory layout of three programs without MMU

Figure 2.20 shows an example where three processes are running, and the constant-displacement flag is unset. Two of the processes are instances of `Program A`, and one process is an instance of `Program B`. All processes share a common library.

As shown in the figure, we have only one address space which is shared by all the processes. Only read-only segments can be shared between different processes. The code segments, which are read-only, are shared between the processes, while each process has its own copy of the data segments.

The challenge is that processes cannot make any assumptions about where each segment will be loaded. The typical situation is that each module has two segments – one segment for code and read-only data, and one segment for read-write data. The code which is running from the code segment needs to be able to locate its variables stored in the data segment. It also needs to be able to locate the address of functions in shared libraries.

The solution to this is to have a table in each module known as the "Global Offset Table", or Global Offset Table (GOT) for short. This is a table with offsets to various functions and variables. The table is stored in the data segment, and will be updated with current addresses of functions and variables during the startup of the program. The offset of this table is stored in a dedicated register. Whenever the application needs to access its data segment, it will look up the address of the variable in the GOT.

Calls between different modules need special handling. Because each module has its own GOT, the register which contains the current address of the GOT needs to updated with the address of the GOT from the new module. To accomplish this, the GOT address for the module containing the function is loaded into the register before the function is called. The old value is restored when the function returns.

In addition to the addresses in the GOT, there may be other addresses in the data segment which needs to be updated. Example:

```
const char *messages[] = {"OK", "Msg1", "Msg2"};
```

This will create an array with addresses to three strings. The addresses will be invalid when the program is loaded, and will therefore need to be updated. To update these addresses, there is a `rofixup` list in the program file. This list contains the location of all addresses that needs to be updated. The `rofixup` list is stored in the code segment of the file, and can therefore be reached by using a relative reference once the program has been started.

Stack

In addition to the memory layout differences, there is another difference between normal ELF files and FDPIC ELF files. If the processor running the application has an MMU, the operating system can grow the stack dynamically as the program uses it. This is infeasible without an MMU, so the stack has to be allocated before the program is started, and it has to be big enough to fit the requirements of the program.

In a FDPIC ELF file, there must be a program header which indicates how big the stack must be. The operating system will then allocate a stack with the required size for the program. The program header with the stack has the type `PT_GNU_STACKSIZE`.

Loading and execution

When the Linux kernel detects a FDPIC ELF file, it will start by loading the program header. It will check whether the file has the constant-displacement flag set. If the flag is set, it will iterate over all the segments in the file, and determine how big a memory area is needed for all the segments. The memory area will be allocated, and then all segments will be loaded with the offset and size which is specified in the segment list.

If the constant-displacement flag is unset, each segment will be loaded independently of all others. Some of the segments may then be shared with other processes.

After the program is loaded, the kernel will transfer control to the program. To allow the program to relocate itself, a loadmap is included as a parameter to the program. The loadmap describes where the various segments are located in the memory.

```
/* segment mappings for ELF FDPIC libraries/executables/interpreters */
struct elf32_fdpic_loadseg {
        Elf32_Addr      addr;     /* core address to which mapped */
        Elf32_Addr      p_vaddr;  /* VMA recorded in file */
        Elf32_Word      p_memsz;  /* allocation size recorded in file */
};

struct elf32_fdpic_loadmap {
        Elf32_Half      version;  /* version of these structures, just in case...
            */
        Elf32_Half      nsegs;    /* number of segments */
        struct elf32_fdpic_loadseg segs[];
};
```

The program will first locate the `rofixup` list. This can be done by using relative addressing – the `rofixup` list is stored in the code segment, and will have a constant

displacement from the initialization code. The program will iterate over the `rofixup` list, and update all the locations listed in that list with new addresses. Once this is done, the program is ready to begin execution.

Dynamic linking

The dynamic linking of FDPIC ELF binaries is done in mostly the same way as the dynamic linking of normal ELF binaries. The Linux kernel will load the dynamic linker in addition to the normal program, and pass control to the dynamic linker. The dynamic linker will receive a reference to both its own loadmap and the loadmap for the program which is executed.

It will load shared libraries, relocate them as needed, do run-time linking, and pass control to the executed program.

2.10.4 Flat

The bFLT format is a simple flat binary format based on the a.out format, and is the de facto format for uClinux. This section is based on [15] and [20].

It was designed to simplify the application load and execute process, create a small and memory efficient file format, support MMU-less systems and storage of GOT. bFLT is either a fully relocatable binary or a PIC. With Position Independent Code (PIC), it is possible to use execute-in-place, and share the text segment between multiple instances. PIC need support for relative addressing in the architecture (this is present in AVR32).

Figure 2.21 shows a conceptual view of the organization of the file. The header contains information about the file format version, where each section of the file is located, and how big the stack should be. A flat binary has one (and only one) text section (code), data section and bss section (relocations).

Usually, Flat binaries are generated by adding an additional tool to the toolchain, by employing a special linker script. `elf2flt` is such an utility, and is used during the linking process. `coff2flt` is an other example of such a utility.

2.10.5 Comparison of binary formats

Feature	**ELF**	**ELF FDPIC**	**FLAT**
Support for MMU-less systems	No	Yes	Yes
Support for shared libraries	Yes	Yes	Yes
Support for arbitrary number of segments	Yes	Yes	No
ELF Compatible	Yes	Yes	No
Need extra step during linking	No	No	Yes

2.11 Linux

Linux is an open source operating system initially written by Linus Torvalds with help from programmers around the world. It is a clone of the operating system Unix, and

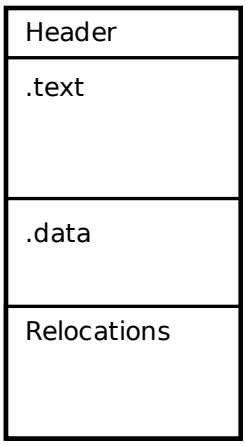


Figure 2.21: Overview of the flat format

aims towards POSIX and SUS compliance.

According to Kernel.org, Linux is easily portable to most general-purpose 32- or 64-bit architectures as long as they have a paged MMU and a port of the GNU C compiler (gcc). Linux has also been ported to a number of architectures without a paged MMU, although functionality is then obviously somewhat limited[14].

In a white paper on Linux in the embedded market, researchers from the VDC Research Group state the following reasons for Linux' growing popularity[25]:

- Licensing cost advantages
- Flexibility of source code access
- General familiarity
- Maturing ecosystem of applications and tools
- Growing developer experience with Linux as an embedded OS

Kernel.org claims that Linux has all the features you would expect in a modern fully-fledged Unix, including true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, proper memory management, and multi-stack networking including IPv4 and IPv6.

Linux was originally made for 32-bit x86, but has later been ported to a wide range of architectures, including:

Alpha AXP, Sun SPARC, Motorola 68000, PowerPC, ARM, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, Intel IA-64, AMD x86-64, AXIS CRIS, Renesas M32R, H8/300, NEC V850, Tensilica Xtensa, Analog Devices Blackfin architectures, Atmel AVR32 (AVR32b)

2.11.1 Configuration

The build process for Linux kernel can be configured through a framework named kbuild. This system consist of a top level makefile, one makefile for each architec-

ture, a set of kbuild Makefiles and a set of common rules for all kbuild makefiles (`scripts/Makefile.*`). Some documentation of this infrastructure can be found in the kernel documentation [17, kbuild/modules.txt and kbuild/kconfig-language.txt]

The configuration defines which subdirectories should be visited during the build process. Each of these subdirectories has a makefile for kbuild, and these use information from the (top level) file `.config` during the build process.

When started, the configuration utility uses information from the Kconfig file in the subdirectory for the currently selected architecture. The Kconfig file may also include other Kconfig files. The configuration utility presents to the user with available compile time options defined in the Kconfig files. Invoking '`make menuconfig`' (or equivalent) will read these files and construct a file named `.config`, located in the root folder of the kernel source tree. The `.config` file is read when the kernel is built. There are also targets defined in the makefiles that sets all, none, random or certain groups of compilation options (allyesconfig, allnoconfig, etc).

2.11.2 Tasks

Internally to the Linux kernel, all threads of execution are known as "tasks", and information about them are stored in a structure named `task_struct`. Each task contains references to the current virtual memory area of the task, the open files, the user the task is running as, and several other pieces of information. Much of that information can be shared with other tasks. For example, the virtual memory area of a task can be shared with other tasks.

By varying what information is shared between tasks, it is possible to accomplish different degrees of separation. Two threads in the same process will share almost everything in the task structure. Two separate processes will share much less, but they will still share some information. The information is still shared includes the current file system name-space and some other name-spaces.

It is also possible to create two tasks with no shared name-spaces. This can be used to create virtual servers, and is a field under active development in Linux.

Kernel stack

On Linux, each task has a kernel stack. The kernel stack is used as long as the task is executed in kernel mode. If the task also executes in user mode, it will have a separate stack for that part. As soon as the task enters the kernel, for example on a system call or on an interrupt, it will switch to using the kernel stack.

The first that is done upon entering kernel mode is always to save the user space registers. This means that the bottom of stack will always contain the user space registers, which makes it easy to retrieve the user space registers of a running thread.

The kernel stack is 8192 bytes large on the AVR32 architecture. Most of the stack is occupied by the stack itself, which grows from the top and downwards. The lowest part of the stack contains a structure named `thread_info`. This structure contains references

to the task this stack belongs to, and also some low-level information about the task. A simple overview of the kernel stack is shown in figure 2.22.

Storing the `thread_info` structure in the lowest part of the kernel stack makes it easy for the kernel to locate the currently executing task. It only needs to retrieve the current stack pointer and round it down to a 8192 byte boundary. This makes retrieving the current task a very low-cost operation.

There are two methods for accessing the information on the kernel stack. To retrieve the user space registers of a task, we have the `task_pt_regs` function. Given a task pointer, that function will locate the bottom of the kernel stack of that task, and retrieve the registers stored there. There is also the `current_thread_info` function, which retrieves the `thread_info` structure of the current task.

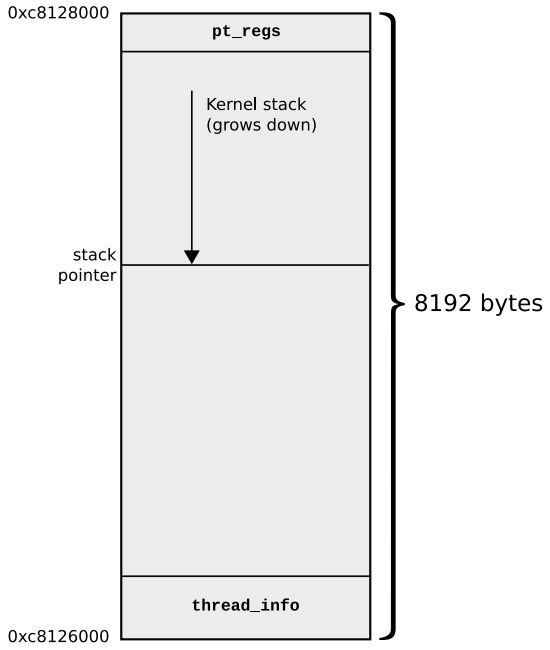


Figure 2.22: Kernel stack

2.11.3 uClinux

Originally, uClinux was a fork of the Linux 2.0 kernel, intended for microcontrollers without MMU support. However, the uClinux project has grown both in brand recognition and coverage of processor architectures, and the uClinux code has been integrated into the main line of Linux development since 2.5.46[22][18]. This is why no special uClinux kernel or patches are considered in this report, since uClinux is already integrated in the official releases from kernel.org. Note that the uClinux name is still used several places in the Linux kernel and the toolchain.

2.12 U-Boot

U-Boot is a boot loader for embedded systems. It is developed and maintained by Wolfgang Denk at DENX Software in Germany, and is mainly used to boot Linux. It also has support for several other operating systems, such as NetBSD and QNX. Several architectures are supported, including PPC, ARM, AVR32B, MIPS, x86, 68k, Nios, MicroBlaze. For each architecture, multiple boards with different CPUs can be supported. U-Boot is open source free software released under the GNU GPL.

U-Boot already supports the AVR32B architecture on Atmel's STK1000 and NGW100 development/evaluation boards. Support for the UC3A was implemented during our previous project during the fall of 2008.

2.12.1 Contributions

To contribute to the development of U-Boot, the code changes should be divided into logical chunks called patches. Patches are submitted to the official mailing list and should conform with its rules. The rules and conventions for the mailing list and U-Boot patches can be found on the DENX Software website[2].

2.13 Toolchain

In this context, a toolchain is a set of software tools capable of creating and debugging executables for a specific platform. It normally includes tools for working with binaries for the target machine, compilers and the C library.

2.13.1 Terminology

In this section, we will introduce some terms used when describing the toolchain:

- **Assembler**: A program for turning a textual representation of machine code into binary code.
- **Object file**: A file with binary code meant to be combined with other files with binary code into a program or library.
- **Linker**: A program for combining several object files (including libraries) into a program or library.
- **Compiler**: A program for turning a high-level language, such as C, C++ or Java into lower level code, such as assembler input, or directly into binary code. The output of the compiler can be a finished program, or an object file that must be linked with other files to form the program or library.
- **Library**: A collection of binary code that can be reused by other programs.

[2] http://www.denx.de/wiki/U-Boot/Patches

- **Shared library**: A library where the linking is done when the program is executed.
- **Static library**: A library that is linked into the program when the program is compiled.
- **C library**: A library implementing all the standard C-functions, such as `printf`, `malloc` and `atoi`.

2.13.2 Linux toolchain

A toolchain on Linux typically contains at least:

- **GNU Binutils** – handles linking of executables, and transformation of assembler files into machine code.
- **glibc** – the C library.
- **GCC** – the C compiler.

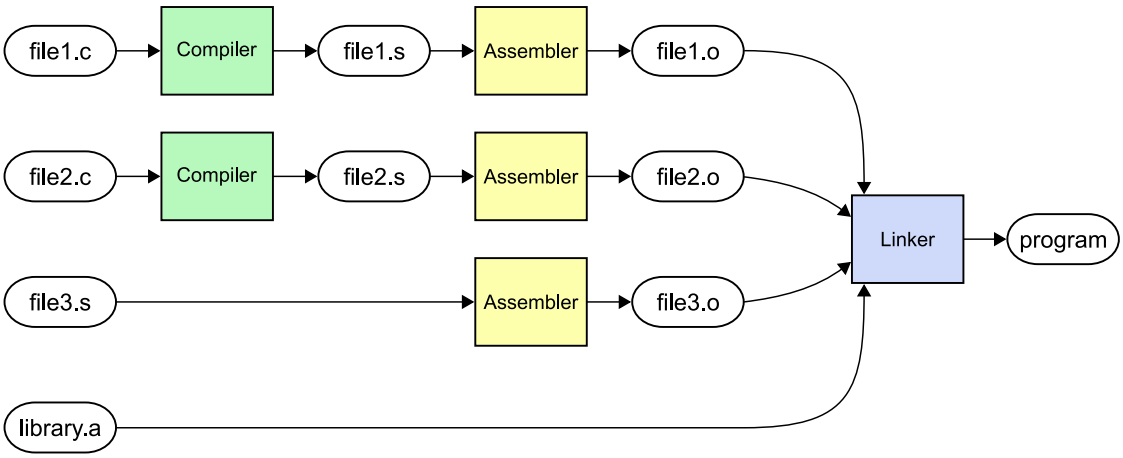


Figure 2.23: Elements of a toolchain

Figure 2.23 shows how various pieces of a toolchain interacts. The figure shows how a program is created from three source files and a statically linked library. Two of the source files are C-files (`file1.c` and `file2.c`), and one of the source files is an assembler-file (`file3.s`). A statically linked library (e.g. the C library) is also included.

The compiler transforms the C-code into assembler files, which in turn are transformed into machine-code by the assembler. This step is usually invisible to the user, as GCC automatically invokes the assembler. The linker takes the object-files with machine code and the static library, and combines them into a single program.

2.13.3 GCC

The GCC project is a collection of compilers for various languages, such as C, C++ and Fortran. It supports several target architectures, including x86, ARM, MIPS and many more.

The GCC mission states: "GCC development is a part of the GNU Project, aiming to improve the compiler used in the GNU system including the GNU/Linux variant. The GCC development effort uses an open development environment and supports many other platforms in order to foster a world-class optimizing compiler, to attract a larger team of developers, to ensure that GCC and the GNU system work on multiple architectures and diverse environments, and to more thoroughly test and extend the features of GCC." The first official beta of GCC was released 1987 and new versions has since then been released on a regular basis.[11]

The official version of GCC from the Free Software Foundation (FSF) does not currently support the AVR32 architecture. However, Atmel is providing support through patches that can be applied to the official version. Both the patches and pre-compiled binaries can be downloaded from Atmel's website. The patched version of GCC support the AVR32 architecture, including the UC3 series, but is unable to produce relocatable programs for Linux.[4]

Extending GCC

GCC consists of language-dependent frontends for handling various languages, optimizers, and machine-dependent backends. The machine-dependent backends handles the various architecture-dependent parts of the compilation process.

`gcc/config.gcc` contains a definition of all the targets GCC can be configured for. When building for AVR32 and uClinux, the following target definition will be used:

```
avr32*-*-uclinux*)
        tm_file="dbxelf.h elfos.h linux.h avr32/linux-elf.h avr32/uclinux-elf.h
            avr32/avr32.h"
        tmake_file="t-linux avr32/t-avr32 avr32/t-elf"
        extra_modes=avr32/avr32-modes.def
        gnu_ld=yes
        ;;
```

This tells us what files GCC will use. The `tm_file`-line lists files that define the target machine. The files will be evaluated in left-to-right order, so files later in the line can override earlier files. Three AVR32-specific files are on that line – `linux-elf.h`, `uclinux-elf.h` and `avr32.h`. All of these are located in the directory `gcc/config/avr32/`. These files configure most of the information about the target – everything from how the linker and assembler should be invoked to how many bits the registers are on that target.

There are also some other files of interest in the `gcc/config/avr32/`-directory:

- `avr32.opt`: The file defining the command line arguments that can be passed to GCC.

- `crti.asm` and `crtn.asm`: Start and end of `_init` and `_fini` sections. The linker combines the sections in these files with sections inn all other files to build two functions which should be called at program startup and program exit.

2.13.4 GNU Binutils

GNU Binutils is a collection of tools for working with binary files. We worked with version 2.18 of GNU Binutils since that version was the one Atmel's patches were created for. Amongst the operations which can be done with GNU Binutils are:

- Building binary files from assembler files, with the `as` tool.
- Linking files with binary code together to form executable programs, with the `ld` tool.
- Examining binary files, with the `readelf` and `objdump` tools.
- Trimming unnecessary parts from a program, with the `strip` tool.

GNU Binutils contains an abstraction layer for working with various types of binary formats[12]. This abstraction layer is known as the Binary Format Descriptor (BFD) library. Since many different platforms and architectures use the ELF binary format, with some variations, a base library of ELF functions has been defined. This library defines a basic implementation of the ELF format, and exposes a set of hooks where the target architecture can insert its own code for architecture-specific code.

2.13.5 `elf2flt`

`elf2flt` is a utility used during the link process to transform a ELF file into the flat binary format. The Flat binary format is described in section 2.10.4. `elf2flt` is developed as a part of the uClinux project.

2.13.6 Libraries

A C library, often called libc or similar, is a collection of header files and library routines that implement common operations. GNU is providing a library named GNU C Library (abbreviated glibc), which is used in the GNU system and most GNU/Linux desktop distributions. uClibc is a C library for embedded Linux systems. Compared to glibc, uClibc is much smaller and support MMU-less CPUs. Nearly all applications supported by glibc also work perfectly with uClibc[23].

uClibc currently supports the AP7 family of microcontrollers, but may need some significant modifications to work on the UC3A family. Dynamic linking of uClibc on MMU-less systems is currently not supported.

2.13.7 GDB

GDB is a feature-rich open source debugger that supports a wide range of platforms and hardware. Atmel maintains its own version of GDB and as features from this branch are matured they are merged into the official version of GDB. Currently, the AVR32 version of GDB (`avr32gdb`) is currently not in the official releases of GDB, but can easily be obtained from Atmel's official web page[3] by downloading the AVR32 GNU Toolchain. GDB enables the user to control and analyze in detail the program execution and states of hardware registers and memory. Instruction data can be disassembled, and breakpoints can be added at specific instructions or at specific line numbers.

2.14 BusyBox

BusyBox is an open-source software application that provides light-weight versions of many common UNIX utilities, and is called "The Swiss Army Knife of Embedded Linux" by its maintainers. It is written with size-optimization and limited resources in mind, and compiles to a single small executable.[2] Because it is open-source and extremely modular, it is very customizable and suitable for embedded systems. BusyBox also aims to achieve fast execution, and minimize run-time memory usage. This makes BusyBox a suitable set of tools for Linux running on the platform concerned in this thesis.

BusyBox is equipped with a a simple menu configuration system, based on the configuration system in the Linux kernel. A screenshot of the main menu of can be seen in Figure 2.24. By altering the configuration options, the BusyBox can be customized to fit the needs of a wide range of projects. It can be adjusted to find a balance between functionality, file size and memory usage requirements.

BusyBox contains a wide range of utilities, categorized by the build configuration system as depicted in figure 2.24. Each "application" of BusyBox is called an applet and most of these aims to be a replacement for the utilities normally found in an GNU system. The applets contain the most important features of the applications they imitate, but generally have fewer options.

[3] http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4118

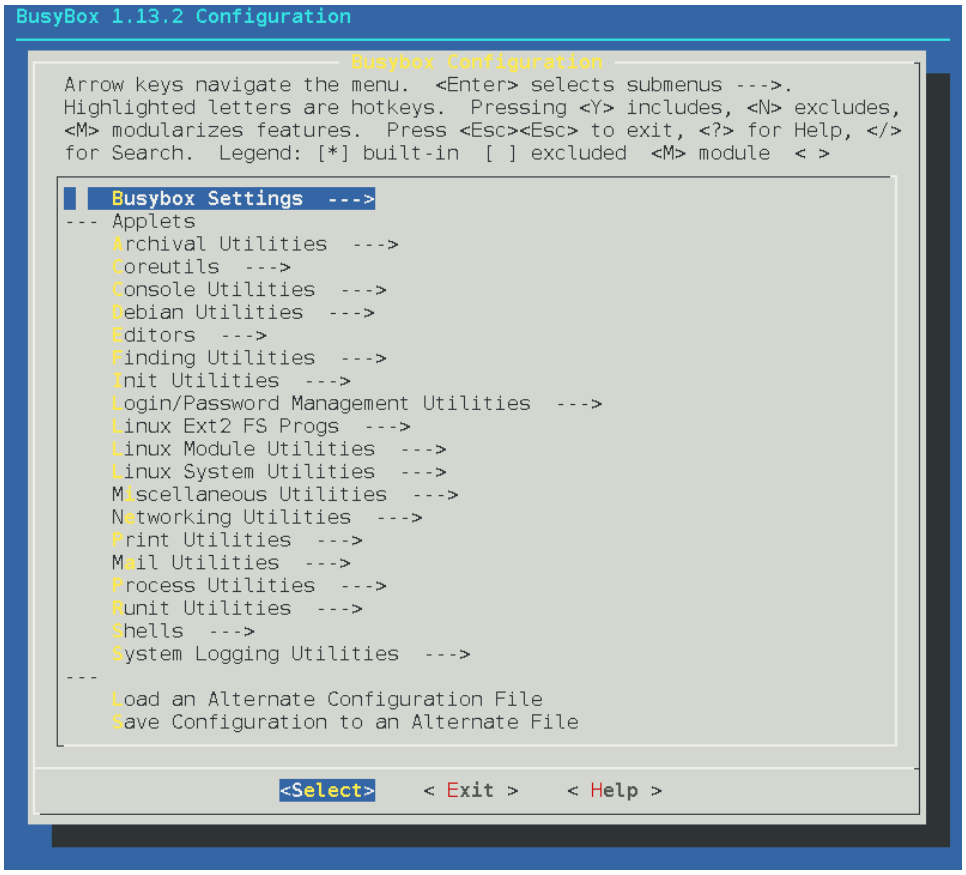


Figure 2.24: Screenshot from menuconfig for BusyBox

The list of applets is fairly long and is not listed here. They can be explored by using menuconfig. Here is short a short list of some of the applets contained in BusyBox: `ls`, `cp`, `cat`, `grep`, `find`, `mkdir`, `rm`, `rmdir`, `df`, `du`, `vi`, `diff`, `adduser`, `passwd`, `fsck`, `mount`, `less`, `ifconfig`, `free`, `ps`, `ash`/`hush`/`msh` (shells), `tar`, `gunzip`. BusyBox can be used as `init`, and thereby start necessary services and applications, e.g. a shell for the terminal and/or telnet server.

According to the official web page, BusyBox will build on any architecture supported by GCC, and is tested with both uClibc and glibc.

2.15 Server protocols

An embedded system can either contain all necessary software and configuration in the firmware, or it can rely on downloading parts from another system or server during start-up. This section introduces concepts and software often used to serve the software and configuration to such a system.

2.15.1 DHCP

A Dynamic Host Configuration Protocol (DHCP) server can be used to distribute configuration options to network devices. The DHCP server usually assign an IP address to the device, and can also provide information about where the root file system and kernel can be located.

2.15.2 TFTP

Trivial File Transfer Protocol (TFTP) is a simple protocol for transmission of files over a Internet Protocol (IP) network. It uses the User Datagram Protocol (UDP) for IP, and this enables it to be very lightweight compared to protocols that use the Transmission Control Protocol (TCP) for IP.

U-Boot can use the response from the DHCP server to locate the TFTP server and download the kernel image from this TFTP server.

2.15.3 NFS

The Network File System (NFS) protocol is, as the name suggests, a protocol for accessing files over a network in the same manner as files are accessed locally. The file system that is mounted in the topmost directory of the file system hierarchy is commonly called the root file system, and in Unix-like systems like Linux it is denoted "`/`". NFS can be used to set up a root file system for a disk-less system, e.g. an embedded system.

2.16 Open-source collaboration

This section gives an introduction to the typical tools and norms for collaboration in open-source projects. A text file named `SubmittingPatches` is included in the Linux kernel documentation[17]. This file describes the general guidelines and rules to follow when submitting patches for Linux.

2.16.1 Git

Git is a revision control system, and was used for maintaining the source code for all the software units changed during this project. Git was initially developed by Linus Torvalds for use with the Linux kernel. Git is a de-centralized version control system with strong focus on performance and many advantageous features for very large distributed development projects.

2.16.2 Merging with current versions

To make sure that any changes done to the source are compatible with the maintainer's current version, development branches should regularly be merged with the branch they are based on. Another reason for merging is to avoid development based on obsolete structures or frameworks. Rebasing can also be used as a way to extract the current

changes and apply them to a newer version. This should ideally give the same result as merging, but with a different revision history structure.

2.16.3 Splitting up patches

The rules for submitting patches for the Linux kernel, and many other projects, states that the patches must before submission be split into logical units of change. For example, if you are going to submit both a bug fix and performance enhancements for a single driver, these should be separated in to two separate patches.

2.16.4 Patch submission format

The patches should usually be sent as an email to the appropriate subsystem maintainer for review. This maintainer will, if the patch is approved, ask the main maintainer to pull this patch into the main branch.

It is important that other developers are able to comment and quote patches, and therefore all patches should be submitted inline in the mail. For submitting patches to the Linux kernel maintainers, the formatting rules listed below apply. Many other software project maintainers have adopted these same rules.

- No MIME
- No links
- No compression
- No attachments
- Max 40kB mails to the mailing list (for larger patches, an URL should be provided instead)

Git provides functionality for formatting and sending patches based on the Git revision history. By specifying the format-patch command (`git format-patch`), Git can be instructed to generate patch files from a given revision interval. Usually, a series of patches should be accompanied by a descriptive cover letter. The patches can be sent by invoking `git send-email` with the cover letter and patch files specified as parameters. Many other parameters can be specified (like sender, receiver, SMTP-server etc), but Git will ask if any obligatory parameters are missing.

2.16.5 Signing your work

Especially Linux, but also other open source projects use the sign-off procedure on patches that are being emailed around. The sign-off line is added at the end of the patch description, and is used to certify that you either wrote it or otherwise have the right to pass it on as a open-source patch. This tag indicates that the signer was involved in the development, or that he/she was in the patch's delivery path.

There is also a less formal tag used, namely “Acked-by”, which is used by developers who have reviewed the patch and indicated acceptance.

2.16.6 Upstream

To send a patch upstream is a term used when they are sent in direction of the original author or maintainer of the project. These could then be included in the next version if they are approved.

2.17 Previous work

In this section we will briefly present previous relevant work done by ourselves and others.

2.17.1 AP7 series

The AP7 series microcontrollers from Atmel are already supported by U-Boot, Linux, GCC, uClibc and GNU Binutils. Because the AP7 and UC3A microcontrollers are implementations of the same architecture, they have many similarities. This is of great significance, since much of the existing code can be reused on the UC3A with few or no changes.

2.17.2 Linux support for MMU-less systems

The uClinux project was started with the aim to run Linux on processors without MMU support, and most of this is now included in the main Linux kernel tree. This means that the main Linux kernel tree already contains the basic code for MMU-less systems, and we can base futher development on this code. We have examined the Linux source code, and have found several architectures that support devices both with and without MMU. The ARM architecture and the MIPS architecture are examples of such architectures. The implementations for these architectures can be used as examples on how this can be implemented for the AVR32 architecture.

2.17.3 Implementations for other architectures

Linux and necessary toolchain components is ported to some architectures without MMU. Some are ported in the uClinux project and most of these use the flat binary format, but there are also some implementations which uses the FDPIC, which can seem to be a more modern format. This section mentions implementations done for other systems without MMU, first for the kernel and thereafter the toolchain.

Linux kernel

All work descibed in this thesis is based on the latest stable Linux kernel version obtained at the beginning of this project (2.6.28.1). This kernel version support the FDPIC format

in three architectures, namely FR-V, Blackfin and SuperH. Each of these architectures have at least one MMU-less variant, and the existing code for these may be useful as inspiration when implementing support for a new architecture. Other implementations for MMU-less processors use the Flat binary format.

Binutils

The GNU Binutils version we based our work on is capable of producing FDPIC binaries for FR-V and Blackfin. Some code for dealing with FDPIC is almost architecture independent, and can in some cases be copied to a new architecture with minor changes.

`elf2flt`

Architectures supported by the current versions of `elf2flt` includes m68k/ColdFire, ARM, Sparc, NEC v850, MicroBlaze, h8300 and SuperH. If the Flat format is going to be used, the existing implementation for these architectures may be used as examples.

2.17.4 SRAM expansion board

On the EVK1100, a 32 MB SDRAM chip is connected to the microcontroller's EBI. Due to the SDRAM bug (see 2.7.2), this memory cannot be used to run code. The end result is that Linux can not be run on current versions of the EVK1100 evaluation kit without hardware modifications. Using the internal SRAM for Linux is infeasible, since only 64 KB of internal SRAM is available, and a running Linux kernel requires far more memory. To work around the SDRAM bug, an expansion board for the EVK1100 with SRAM and flash memory was developed by Atmel for our previous project in 2008.

The expansion board connected to the EVK1100 can be seen in figure 2.16. The board is just a circuit board with a connector for the EVK1100, footprints for two 2 MB SRAM chips and one 8 MB flash chip, and some resistors and de-coupling capacitors. See appendix H for the expansion board schematics.

Note that several pull-ups and capacitors have been removed from the EVK1100 to avoid conflicts with the expansion board.

Chapter 3

Implementation

In this chapter, we will present our approach to porting Linux to the UC3A0512 microcontroller and EVK1100 evaluation kit. We will also show what changes we did to GCC and GNU Binutils to support Linux on this platform.

In the first section we present the organization of our work flow, by explaining the approach we used to implement our requirements. The next section list what we excepted to be nessecary in order to achieve our goals. Our development setup is presented in section 3.3. The next section, section 3.4, present the changes (mostly cleanups) done to U-Boot in this project.

Section 3.5 presents the decision made as to which binary format is going to be used. The Linux kernel modifications and toolchain adaption is presented in the next sections, section 3.6 and 3.7 respectively.

Some adaptations and workaround had to be done to the development board and initializing code. These are presented in the sections SRAM optimization (3.8) and SPI chip select (3.9).

Our assignment text indicates that a list of applicable Linux programs should be compiled. In section 3.10 our use of BusyBox, the swiss army knife of embedded Linux, is presented.

The last section, section 3.11 present how we acquired the necessary code, and how we distributed our changes.

3.1 Methodology

An iteration based approach was used during this project. This approach is inspired by the "Incremental Process model"[10].

Each of the iterations in the process consists of the steps listed below. The Figure 3.1 shows the steps involved in the process flow.

1. Identifying the next goal needed to fulfill our requirements.

   In this context we use the word goal when we refer to the coarse grained steps required to reach the requirements we defined in section 1.3.1. A goal can be e.g.

making the kernel boot, getting GCC to build a proper binary, etc.

2. Sketching preliminary milestones for what we think is the most reasonable way to reach the goal.

   In this context we use the word milestone for the more fine grained steps required to reach a goal. A milestone can be e.g. adding the new linker target in GCC, updating the BFD, etc.

3. Run short iterations consisting of the steps listed below. If we during these iterations found that our milestone still were too coarse, we would refine them into smaller steps.

   (a) Define or refine the milestone. We usually had a general idea of what the milestone should be, and we jumped directly to the next step.

   (b) Identify the necessary steps to reach the milestone. More on this in section 3.1.2.

   (c) Implement what we identified as necessary.

   (d) Test whether we reached the milestone or not.

   - If the goal is reached we go to step 1, and enter a new iteration with the next goal.
   - If the milestone is reached, we clean up our code by removing debug output and try to format it to conform with coding guidelines. We then enter a new iteration for the next milestone.
   - If the milestone is not reached, we begin a new iteration for the same milestone.

   We kept these iterations short by keeping our milestones fine grained.

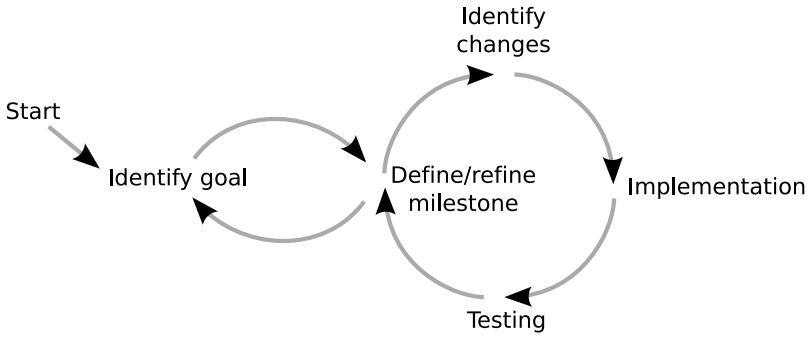


Figure 3.1: Development process

3.1.1 Setting goals and preliminary milestones

At the beginning of this project, we mostly had a superficial idea about what had to be done to the code. To identify the necessary changes in complete detail, we would have to analyze all the relevant code. With the time frame set for this project, this would be infeasible, due to the size and complexity of the kernel and toolchain. Instead, the parts of the code necessary to understand were gradually uncovered during implementation. Some figures to support this decision (these figures as counted with a simple script and used the 2.6.28.1 kernel as source):

- The core kernel code (counting the `kernel`, `mm/`, `init/` and `fs/` subdirectories), contains almost one million lines of code.
- The AVR32 architecture implementation of the kernel contained about 23000 lines of code when we started our development.
- The FR-V architecture often used as reference, had about 19000 lines of code.

3.1.2 Milestone identification and implementation

When a milestone was identified, attempts were made to identify the necessary changes. Techniques varied a lot between the milestones, but typically consisted of one or more of the following steps:

- Reading documentation.
- Analyzing code, including code for other architectures.
- Tracepoints in the code (ie. printf/kprintf).
  This enabled us to get a overview of the execution flow and study the internal state during execution.
- Single stepping with GDB was used when we did not get the whole picture from the tracepoints we used. This technique was vital when an error caused a stack corruption. Stack corruption makes it difficult to locate the problem, becuase the stack is useful for backtracking the execution.
- Analyzing binaries with readelf/objdump (this is relevant only to the toolchain adaption).

During this survey phase, we found that we had to add other goals and milestones. Often we saw that changes had to be done in an other part of the system. E.g. when we worked our way through the compilation process, we often realized that work had to be done in the Linux kernel's executable file loader and vice versa.

Some of the milestone identification and implementation attempts revealed that the milestone we set was not the right way to go, and we jumped right to the refining the milestone or even defining a completely different goal without completing the iteration.

3.1.3 Review

One important element of open-source development collaboration is the public review process, where other developers can read, test and comment the submitted code. All patches should be reviewed thoroughly and approved before they are applied to the maintainer's development branch. There were two main reasons for us to submit our work to the appropriate mailing lists. First of all, the lists can provide valuable feedback to our work. Secondly, we wanted to make the code available to anyone that could make use of it. Publishing our work was also indicated in both the original assignment and in our communication with Atmel. The received feedback is presented in section 4.5.

3.2 Expected changes

In this section we list the changes we identified early on as necessary to reach our goals.

3.2.1 U-Boot

U-Boot was already working when the development started this spring. We wanted to incorporate the feedback we received on the patches we had sent out during our previous project into a new version of U-Boot. Some changes could be valuable both for us during development and for other who can benefit from our work.

We also wanted to make another effort to improve the speed the SRAM worked on, since the memory speed severely limited our execution speed.

To implement support for SPI and SD card reader we would investige the possibility of employing existing drivers.

3.2.2 Select binary format

A binary format has to be selected in order to be able to fullfill the other requirements. This should be done by investigating both FDPIC ELF and Flat, and selecting the most fit.

3.2.3 Linux

The Linux port was at the start time of this project incomplete, and several things had to be done here.

The configuration files have to be updated to support both the new CPU and the new board.

Some hardware drivers have to verified and updated. In U-Boot the networking speed had to be limited to 10Mbps when the clock is to slow. This change has to be done to the Linux kernel as well. The GPIO subsystem is quite different from the PIO system found in AP7, and this requires some changes. Some similarities exists, so parts of the code can be reused.

The executable loader has to be adapted to support the FDPIC ELF format. This include adding the platform independent loader to the configuration, and implement platform dependent helper functions for this loader.

Exception and interrupt handling needs to be updated. Most of these changes here is to revise the `entry-avr32a.s`, so that it suits this processor.

Some differences in the memory system have to be taken account for. The address space layout has several differences, and this has to be fixed. The memory copying routine for this processor that cannot do unaligned access could be optimized. We found that it could be faster to do halfword copying or similar, when that was possible, instead of going to byte copying in all other cases than the trivial aligned copying.

3.2.4 Toolchain

We must create a toolchain that is capable to produce binaries which can be executed on our platform. Since this platform doesn't have an MMU, the executables must be relocatable. We have two choices when it comes to executable formats – FDPIC ELF or Flat, and must decide on one of these.

If we decide to add Flat support, we must modify the `elf2flt` tool. It might also be necessary to change some of the toolchain, so that it can generate relocatable ELF executables. These executables should then be processed by the `elf2flt` tool to produce Flat binaries.

If we decide to add FDPIC ELF support, we will have to change the linker. The linker must be able to generate valid FDPIC ELF executables, which requires the executables to contain relocation information. We must also change GCC to support the `-mfdpic` flag, and pass it to the linker.

3.2.5 User space

Some sort of user space programs are necessary for this project to be of any use. We must therefore compile some useful programs for the platform and verify that they work.

3.3 Development setup

In this section we will introduce the setup of hardware and software used for development during this project. The setup consists of a computer running Ubuntu Linux 8.04, the EVK1100 development board, and a JTAG debugger. These components are connected as shown in figure 3.2.

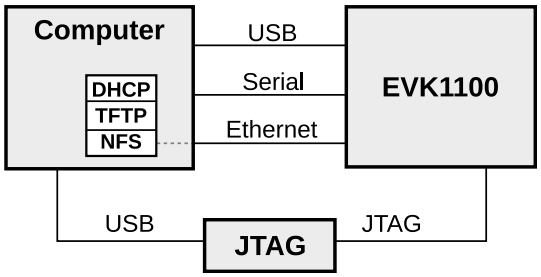


Figure 3.2: Development setup

3.3.1 JTAG

A JTAG connection is used for uploading the U-Boot boot loader to the board, and for debugging. The JTAG connection enables us to single step in both the running kernel, and in programs started by the kernel. By adding breakpoint instructions, we are also able to halt the execution at specific places, and inspect the data currently in memory, and the state of the CPU.

A programming and control utility called `avr32program` was used to program the microcontroller. The following command was used to upload U-Boot to the internal flash:

```
avr32program -pjtagicemkii -part UC3A0512ES program -finternal@0x80000000
-cint -F bin -O 2147483648 -e -R -r u-boot.bin
```

The parameters and arguments specifies the following:

- `-pjtagicemkii`: What programmer is connected to development board.
- `-part UC3A0512ES`: The device to be programmed.
- `program`: The action `avr32program` should perform. In this case it is "program memory".
- `-finternal@0x80000000`: Tells `avr32program` that the programming should be done to the internal flash memory located at offset `0x80000000`.
- `-cint`: Which clock source the CPU should use during programming. `int` selects the internal RC oscillator.
- `-F bin`: The input format. `bin` means a binary file.
- `-O 2147483648`: The offset that should be programmed. 2147483648_{10} is the same as 80000000_{16}, which is the start of the internal flash memory.
- `-e`: Erase the flash before programming.
- `-R`: Reset the chip after the programming is complete.
- `-r`: Start execution after the reset.
- `u-boot.bin`: Filename of the binary file to write to the flash.

3.3.2 Serial cable

A serial converter was used to access the console of U-Boot and Linux. It was a generic USB-to-serial converter, and a baud rate of 115200 baud/sec was used.

3.3.3 Networking setup

A DHCP server was used to distribute configuration options to the development board. The DHCP server assigns an IP address to the board, and also provides information about where the root file system and kernel can be located (line 8 and 9 in listing 3.1).

The TFTP server is installed on the development computer, and configured to respond to requests for files located in a designated folder. It was used to serve the boot image for the Linux kernel.

Listing 3.1 shows the configuration file used by our DHCP server. We used udhcpd[1] as DHCP server.

```
start           192.168.0.20                    #default: 192.168.0.20
end             192.168.0.254                   #default: 192.168.0.254
interface       eth0                            #default: eth0
max_leases      234                             #default: 254
opt             dns             192.168.0.1
option          lease           864000          # 10 days of seconds
siaddr          192.168.0.2                     #default: 0.0.0.0
boot_file       /srv/tftp/uImage                #default: (none)
opt             rootpath        /tftpboot/evk1100
```

Listing 3.1: DHCP configuration

The three last options gives information to clients about the TFTP server and NFS server.

- `siaddr` is the ip address of the server which hosts the TFTP server and NFS server.
- `boot_file` is the location of the kernel image file.
- `opt rootpath` sets the NFS root directory.

In our setup, U-Boot uses the response from the DHCP server to locate the TFTP server. It downloads the kernel image from this TFTP server, and executes it. The Linux kernel also receives a DHCP response, and uses it to locate the NFS server. This server is then used for the root file system.

3.4 U-Boot

At the end of our previous project, an updated set of patches for U-Boot was submitted to both the official U-Boot mailing list and the avr32linux.org's U-Boot mailing list. Some constructive criticism about these patches was posted on the mailing list, and we decided to clean up some of the things remarked. This section describes every change

[1] http://packages.ubuntu.com/hardy/udhcpd

done to U-boot during this project, grouped in logical subsections. The actual changes can be seen in appendix B. Note that appendix B only lists the changes to the patches, not the complete revised patch series. For the complete patch series, see the official U-Boot mailing list[2] or the digital appendix of this report.

Also note that some changes to the U-Boot source code were done after the submission of the revised patch series. These changes are described in section 3.4.5 and 3.4.6, and listed in appendix C.

3.4.1 Network speed limiting

During our previous project, a modification was done in the MACB driver in U-Boot to limit the operating speed of the PHY. This patch was replaced to reduce the changes in the MACB driver, and only limits the network speed if it is explicitly defined by setting a board configuration flag named `CONFIG_MACB_FORCE10M`. The original version the patch can be found on the U-Boot mailing list archive[3], and the modifications of it are listed in B.1. This new version of the patch received some criticism, and triggered some debate with suggested solutions on the mailing U-Boot mailing list, but no follow-up solution was implemented by us.

3.4.2 Adding the EVK1100 board to lists

We added the EVK1100 board to the files `MAKEALL` and `MAINTAINERS`. The `MAKEALL` file lists all boards for the AVR32 architecture supported by U-Boot, and the `MAINTAINERS` file lists the people that maintain different parts of U-Boot.

3.4.3 Precedence safety fix

When preprocessor macros are used to define simple mathematical expressions, the resulting expression substituted by the preprocessor may become a part of a larger expression. In some cases, if the macro is used without care, the resulting expression may produce the wrong result. To make sure that this never happens, we introduced some parentheses around the mathematical expressions. This change is shown in appendix B.3. Listing 3.2 shows an example of how careless use of the previous version of the macro can be used to produce the wrong result.

```
Old macro version:      Evaluates to:           Result:
SMC_CYCLE(42)*2         0x0008+(42)*0x10*2      1352

New macro version:      Evaluates to:           Result:
SMC_CYCLE(42)*2         (0x0008+(42)*0x10)*2    1360
```

Listing 3.2: Macro precedence error example

[2] http://lists.denx.de/pipermail/u-boot/

[3] http://lists.denx.de/pipermail/u-boot/2008-October/041568.html

3.4.4 Esthetical and other minor changes

The previously submitted U-Boot patches was updated to make the code conform with the coding style specified by the maintainer. The changes listed in appendix B.4 to B.7 are merely esthetical changes, removal of unused variables, and correction of comments. The only exceptions are the introduction of the network speed limiting flag, and the baud rate adjustment in the board configuration.

3.4.5 Auto detection of PHY address

The last patches submitted in the fall of 2008 included a routine for auto detecting the address of the external PHY. This routine would be invoked if and only if U-Boot was compiled with the PHY address set to `0xff`. This was changed so that a flag in the board configuration file determines whether or not the routine will be compiled and user. The board configuration file is a more appropriate place for this option, since the PHY address is determined by the board. A flag also enables the possibility to make the preprocessor remove the auto detect routine. This in turn, results in a slightly smaller binary output file. As can be seen in appendix C, the files changed to achieve this were `atevk1100.c`, `atevk1100.h` and `macb.c`.

3.4.6 Removal of bug workaround

In an early stage of development, a structure describing the layout of the GPIO registers were shared between the implementations for UC3 and AP7 families. The layout of the GPIO registers are not the same on these two architectures, and the structure defined in software was incompatible with the UC3A. When writing to the GPIO registers defined by the incompatible structure, the wrong memory locations were accessed. This error surfaced by causing an interrupt to occur when initializing the USART, and before the bug was found, a workaround was implemented. The bug was eventually removed, but the workaround remained. The removal of this obsolete workaround can be seen in line 65 in appendix C.

3.5 Binary format selection

The current Linux support for the AP7000 microcontroller is based on the ELF binary format for programs and shared libraries. This format requires that the architecture has an MMU, and is therefore unsuitable for the UC3A0512 microcontroller. We had to select another binary format suitable for MMU-less architectures, and implement support for this format in the toolchain. The Linux kernel currently has support for two such formats, and we found it most practical to choose one of them.

The original assignment text ask for Flat binary support in the toolchain, but on the web page given with the assignment FDPIC ELF is proposed. In discussions with our supervisor at Atmel, FDPIC ELF was suggested as an equal, if not better alternative.

In section 2.10 both FDPIC ELF and the Flat binary format is presented. During development we tried both formats, and ended up using the FDPIC ELF format. Even though the Flat file format is simpler and more widespread the FDPIC ELF was chosen due to several advantages with this format:

- Simpler toolchain usage (does not need additional programs).
- More compatible with existing toolchain (objdump, readelf, gdb, etc).
- ELF support for AVR32 is already implemented.
- Closer to the standard format used in Linux.
- Flat is limited to four shared libraries in total in a program.

3.6 Linux kernel

This section describes the changes we did to the Linux kernel during this project. Processor-specific folders, files and code in the Linux source tree had already been added and modified during our previous project. The changes done to the Linux source tree during both projects have now been cleaned up and grouped into logical patches. These patches are listed in appendix D, and also summarized in section 3.11.7.

The first section is about the rebasing done in the start of this project and the second section regards the added and modified configuration files. Section 3.6.3 describes changes done to support the UC3 core, and section 3.6.4 is about changes done because the microcontroller used does not have a cache. In section 3.6.5, we describe some changes we had to do to the clock setup. Section 3.6.6 discusses the changes done to the driver for the network adapter. The next section, section 3.6.7, describes changes done to get the GPIO system to work. Section 3.6.8 mentions the configuration of the LED driver. The attempt to support SPI with Direct Memory Access (DMA) is discussed in section 3.6.9. A workaround for a bug in the CPU is described in section 3.6.10.

The next four sections presents changes that had to be done regarding memory access, both due to the lack of support of unaligned memory access and the lack of MMU.

Section 3.6.15 regard changes nessecary for the differences in exception and interrupt handling in the processors. Section 3.6.16 discusses modifications done to support FDPIC ELF binaries. Section 3.6.17 and section 3.6.18 regard refactoring done to make other changes easier. The last section gives an overview of all the patches we created.

3.6.1 Rebasing

The 2.6.27-rc6 version we had started out with during the fall of 2008, was getting quite old. We therefore started development with rebasing our changes on version 2.6.28.1 of kernel. When rebasing, we take all of our changes, and apply them to a newer version. This was done for the same reasons as given for merging in section 2.16.2. We selected version 2.6.28.1 because it was the most recent stable version of the kernel, and a kernel

with few defects were desirable. A release candidate for version 2.6.29 were available, but this kernel was more likely to contain defects.

3.6.2 Configuration files and make files

During our previous project, some changes were made to the `Kconfig` and `Makefile`s to include the EVK1100 development board and the UC3A0512 microcontroller. To enable the compilation the UC3A0512 via the configuration system, the EVK1100 was added as a selectable board in `Kconfig` and `Makefile` in the `avr32`-folder (see patch 29 in appendix D.29).

Board support

During our project in 2008, we had copied the `atngw100` folder in `arch/avr32/boards` to a new folder named `atevk1100`. The file `setup.c` in this folder had to be rewritten to match the hardware on the EVK1100. These changes include setting up the oscillators, SPI configuration, LED configuration, and running initialization functions for applicable hardware. The clock configuration is discussed in section 3.6.5.

The patch adding board support is listed in appendix D.29. A part of this patch adding a file with a generated default configuration (`defconfig`) for the board, is omitted due to its length, but is included in the digital appendix. This file contains the kernel configuration used when the kernel was compiled, and can be generated by invoking `make menuconfig` and configuring for this system.

3.6.3 UC3A support

In our project, during the fall of 2008, we began the process of taking the code for the AP7 microcontroller family and adapting it for the UC3A family. The `arch/avr32/mach-at32ap` folder was copied to `arch/avr32/mach-at32uc3a`, and changes were made to the code. These changes were sufficient to almost complete the boot process, but still some nessecary changes remained.

The patch that adds support for UC3A devices can be seen in appendix D.28.

The file `at32uc3a0xxx.c` defines on-chip devices in the microcontroller, and the memory locations and layouts of these. Most of these addresses had to be updated, since the memory layout of AP7 microcontrollers greatly differ from the UC3A series. Many features in the AP7 are not present in the UC3A series. Support for the following features had been removed from `at32uc3a0xxx.c` (copied from `at32ap700x.c`) during the previous project:

- MultiMedia Card Interface (MCI)
- IDE/CompactFlash interface
- NAND Flash/SmartMedia
- AC97 Controller

- Audio Bitstream Digital to Analog Converter (DAC)

3.6.4 Cache

Since the processor used in this project does not have the same caching facilities as AP7000, some function calls used in the that implementation had to be removed. A flag was added to make this conditional on whether the chip has cache or not. This was done by adding these functions as empty stubs in an architecture specific file. Moving these functions to a header an setting them to be inline would be more efficient, because then the compiler would be able to optimize them away. This was not done because optimization was not highly prioritized in this phase. Our changes can be seen in appendix D.12.

3.6.5 Clocks

There were two tasks that needed to be done for the clock setup. The first one was relatively simple, and was to configure what clocks were available on the EVK1100 board. This was done in `arch/avr32/boards/atevk1100/setup.c`, where we updated an array named `at32_board_osc_rates`. Up to three external clocks can be connected to the UC3A, so this array has three elements. One for the 32 kHz slow clock, one for `osc0`, and one for `osc1`. On the EVK1100, a 12MHz clock is connected to `osc0`, and `osc1` is not used.

The array thus became:

```
unsigned long at32_board_osc_rates[3] = {
        [0] = 32768,    /* 32.768 kHz on RTC osc */
        [1] = 12000000, /* 12 MHz on osc0 */
        [2] = 0,
};
```

The second task that needed to be done for clocks, was to update all the clock connections for the microcontroller. The AP7000 and the UC3A0512 share many of the same internal devices, but they are connected to different clock outputs. We therefore had to revise all the device definitions, and update the clock connections. For example, on the AP7000, the SDRAM controller is connected to clock output 14 (i.e. clock mask bit 14) on the peripheral bus B. On the UC3A0512, it is connected to output 5 on the same bus.

At runtime, the Linux kernel uses a list to keep track of which clocks are in use, and this list is used to assemble clock masks. The clock masks are used to disable clocks for inactive devices.

We also had to add clocks for devices that are present in the UC3A, but not in the AP7000. The clock for on-chip debug system is an example of such a clock that we had to add. Before the clock was added to the list, the debug system was turned off during startup. This prevented us from accessing the device over JTAG.

3.6.6 Limiting network device speed

The MACB driver in both U-Boot and Linux was compatible with the MACB in the UC3A0512 microcontroller. However, because of the combination of low clock speed and RMII-mode we had to force the driver initialize the macb to 10Mbit/s mode. This had been done to the MACB driver in U-Boot in previous work, and we had to make an equivalent and proper solution for the driver in Linux. The final solution can be seen in appendix D.1. This patch adds a few lines of code that checks whether the mode is set to RMII, and disables support for 100Mbit/s if the CPU speed is not high enough for this mode.

3.6.7 GPIO

The GPIO controller on the UC3A0512 microcontroller is different from the PIO controller found on the AP7000. Therefore, the PIO controller code had to be modified to work on the UC3A0512. We started by copying the file PIO controller files, and renaming all functions and variables from `pio` to `gpio`. We then updated the header file (`mach/at32uc3a/gpio.h`). This header file contains the register definitions for the GPIO controller.

We then went through the code in this file, and updated it to access the correct registers. Mostly the registers were present, but with a different name. For example, to enable pull ups, we had to set the `PUERS` register instead of setting the `PUER` register.

Some decisions were more difficult. For example, the AP7000 has support for something called multi-drive capability. When examining the schematics for a output pin in the data sheets for the AP7000 and UC3A, it was not immediately apparent that this did the same as the UC3A's open drain mode. In the end we concluded that it did the same.

3.6.8 LED device driver

The EVK1100 has 8 LEDs that can be controlled independently (four single and two double). The NGW100 board has 3 LEDs, and we could simply re-use and modify the definition of these in the code. Linux uses a generic driver to control the LEDs, and this driver utilizes the generic GPIO interface. Note that in the final code, LED3 is not enabled because it is connected to the EBI bus (see 3.8.3). Lines 104-117 in patch 29 (appendix D.29) adds the necessary setup configuration for the LEDs in `setup.c`. The LEDs can be controlled in Linux by writing to trigger files in folders that appear in `/sys/class/leds/`, e.g. the command `echo ‘‘heartbeat’’ > /sys/class/leds/led1/trigger` enables a heartbeat on LED1.

3.6.9 SPI with DMA support

In Linux, the most suitable and proper way to communicate with the SPI on AVR32 devices is to set up and use a DMA controller. Both the AP7 and UC3A series have PDC controllers that provide hardware support for DMA functionality. Peripheral DMA

Controller (PDC) is abbreviated as PDC in the AP7000 datasheet, and PDCA in the UC3A datasheet. We will use the same convention here to distinguish between the two.

In an attempt to enable the SPI bus to communicate with the LCD display and DataFlash, the Linux source was searched for existing compatible or similar code for this. Support for the Peripheral DMA Controller (PDC) in the AP7000 series microcontroller was found in the Linux source code, but it was incompatible with the PDCA implemented in the UC3A series. While the PDC configuration registers are located in a reserved memory area of each IO device, the Peripheral DMA Controller (PDCA) has one central memory area for its configuration registers.

Because of these structural differences, we decided to write a generic interface to abstract the difference. The development of this interface was aborted when we were informed by Atmel that the existing PDC code had been restructured. The patch that changes the SPI driver and introduces the abstraction layer can be seen in appendix E.

3.6.10 Interrupt bug workaround

Because of a bug in the CPU, any instruction masking interrupts through the system register must be followed by two No-Operation (NOP) instructions to avoid abnormal behavior (see [8] section 41.4.5.5). This workaround had already been implemented in U-Boot, but also needed to be introduced in the Linux kernel. A separate patch was made for this specific workaround, and can be seen in appendix D.27. As can be seen in the patch, two NOP instructions are also added in the `mask_exceptions` macro. This may be superfluous since the bug should only affect masking of interrupts, not exceptions. The performance penalty of two NOP instructions is very low, so we chose to include them just in case.

3.6.11 Memory to memory copying

The Linux kernel includes architecture specific implementation of memory-to-memory copying routines. The existing implementation for the AP7000 had to be modified because the UC3 is not capable of doing unaligned memory accesses. These routines are as usually optimized in assembly because they are used very often.

The patch in appendix D.16 add a memory copy routing which is based on a the version found in the AP7000 implementation. The changes were simple changes, with no attempts at optimization.

3.6.12 Memory copying with checksumming

In conjunction with TCP networking, when copying data from one place in memory to another, it is desirable to also checksum the data. It is most efficient to implement routines that perform these two operations at the same time. That way, one does not have to read the same data several times.

The `csum_partial_copy_generic` function implements this for the AP7000. Unfortunately, this function assumes that the architecture can do unaligned accesses, which

makes the code incompatible with the UC3A. The patch that fixes this incompatibility is listed in appendix D.14. It changes the code that calls `csum_partial_copy_generic` to check that the buffers are aligned first If the buffers are unaligned, it will first copy the data, and then checksum them.

The patch also updates a function named `csum_partial`. This function does the same checksumming, but without copying the data. We updated this function to handle unaligned accesses.

3.6.13 User space memory access

The Linux kernel will often need to read or write memory belonging to a user space program, usually in response to a system call. There are a number of functions for performing these operations:

- `access_ok`: Check whether a range of memory is valid user space memory.
- `clear_user`: Fill a block of memory with zeros.
- `copy_from_user`: Copy a block of data from user space to kernel space.
- `copy_to_user`: Copy a block of data from kernel space to user space.
- `strncpy_from_user`: Copy a string from user space.
- `strnlen_user`: Get length of a user space string.
- `get_user`: Read an integer from user space.
- `put_user`: Write an integer to user space.

These functions provide a generic interface to the architecture-specific methods for accessing memory. They are also responsible for preventing user space processes from reading or writing data they shouldn't have access to. This is done by making sure that the memory areas passed to the functions belong in the user space part of the memory.

Some of the functions also have versions with less checking. Those functions are named with a `__` prefix, e.g. `__get_user`. `access_ok` must be used to validate the block of memory before using the functions with less checking. Failure to do so may result in security vulnerabilities, where a program may access memory it isn't allowed to access.

The existing functions for accessing user space memory utilizes the built-in MMU in AP7 processors to handle access violations to memory. This is, for example, used to handle the case where a read-only segment of memory is passed as the destination of copy_to_user. To implement this, the function marks every address where an exception may occur with an operation which be done if an exception occurs at that point.

Support for unaligned accesses

The existing `copy_to_user` and `copy_from_user` in the kernel were originally written for the AP7000 microcontroller, and assumes that unaligned accesses can be performed. We needed to change these functions so that they would work without performing any unaligned memory accesses. Without a functional MPU, memory protection is a lost cause, so we could simply use the normal memory copy functions for the implementations. There are however some advantages of implementing these functions with error checking. Error checking enables us to catch errors when user-space programs pass invalid pointers to the kernel. Also, if Atmel creates an microcontroller that features an MMU, but doesn't allow unaligned access, our implementation should be reusable. It might also be possible to use this code with later revisions of the microcontroller where the MPU is functional.

The final implementation can be seen in appendix D.13 (patch 13). This patch introduces a new file, `copy_user-nounaligned.S`, to the `arch/avr32/lib` folder. This new file is a copy of the existing `copy_user.S`, modified so that the alignment of the input addresses are checked. If both addresses are aligned, the CPU can perform per-word copying. If not, simple per-byte copying is performed.

This could be optimized further, but we have not prioritized optimization. How this could be improved is discussed in section 6.2.5.

User space address ranges

On the AP7 implementation, all user space memory is located in the lower half of the virtual memory address space, while all kernel memory is located in the upper half. Functions which access user space memory validate that the memory they are accessing are located in the lower half of the address space.

On the the UC3A implementation, there is no separation between address spaces because it lacks MMU, and the kernel memory may be mixed with the user space memory. There is no fast way to determine whether a block belongs to user space or to the kernel. Without any memory protection, doing the check does not bring any extra security either. We therefore decided to disable these checks for in our implementation.

There were three places we decided that we needed to update:

`ret_if_privileged` is an assembler macro that is called by several other assembler functions, such as `copy_from_user`. It checks the memory area defined by the input parameters, and determines whether it overlaps with the kernel's memory. This check does not work without the layout of the virtual address space employed by an MMU, and we chose to simply disable this check at compile-time. Patch 19 listed in appendix D.19 shows how this was done.

`access_ok` does the same check as `ret_if_privileged` by invoking the `__range_ok`, but is accessible from C code instead of assembler. We decided to replace the `__range_ok`

macro at compile-time by a dummy macro. This change is shown in Linux patch number 23 in appendix D.23.

`strnlen_user` checks the length of a string residing in user space. It contained some checks for the string length, to make sure it did not extend into kernel memory. The check in this function and its helper function `adjust_length` was removed for systems without an MMU. This change is listed in appendix D.17.

3.6.14 Address space layout

There are some major differences in the address space layout of the AP7000 and the UC3A microcontrollers. Most of the differences are due to the AP7000 having an MMU while the UC3A has an MPU. There are also some differences in the physical layout of the memory.

Physical layout

One of the differences is the physical layout of different memory blocks. There are several separate blocks of memory addresses designated for accessing different devices, and embedded or external memories. The base addresses of these memory blocks differ between UC3A and AP7 microcontroller. Some blocks of memory are only available on either the AP7 or the UC3A. For example, the embedded flash memory in the UC3A is not present in the AP7 microcontroller.

Virtual memory layout

On the AVR32 processors with an MMU the virtual memory area is split into five segments:

- **P0/U0**: 2GB of memory with caching and paging.
- **P1**: 512 MB of memory with caching but without paging.
- **P2**: 512 MB of memory without caching and paging.
- **P3**: 512 MB of memory with caching and paging.
- **P4**: 512 MB of memory mapped to device registers and memory. No paging or caching.

Only the P0/U0 segment is accessible in unprivileged mode. The various Px segments are used for various low-level code. Both the P1 and P2 segments map the same physical memory.

The Linux kernel was loaded into the P1 segment. To change the caching property when accessing memory, various places in the kernel convert an address from the P1 segment to the P2 segment. The P3 segment is used when the kernel needs to use page

translated memory for some purpose, for example when it needs to map device memory with specific caching properties.

When we updated the Linux source code, we had to update all the code which assumed that the microcontroller used this segmented memory model.

Null pointer debugging

Because the internal SRAM is located on address 0, this is a perfectly valid address. In processors with an MMU, reading or writing to address 0 is usually caused by an error, and causes an exception. In our case, the CPU can read and write to all the SRAM memory and even execute code from it. Whenever a software error caused data to be read from address 0, whatever data residing on this address in SRAM would be fetched. When software errors cause a jump to an addresses in the SRAM, the CPU will interpret whatever data on that location as instructions and attempt to execute them. This may further instruct the CPU to perform any operations. In our case, the contents of the SRAM would be any data left behind by the execution of U-Boot.

To ensure that the CPU halts when it tries to execute instructions from the SRAM, a short routine was temporarily introduced in the `sram_init` function in `arch/avr32/mach-at32uc3a/at32uc3a0xxx.c`. This routine writes the breakpoint instruction to every address in the SRAM, enabling us to detect the error earlier, with any potential backtrace information guaranteed intact. The routine is shown in listing 3.3.

```
1 unsigned long i;
2 unsigned short *p;
3 p = 0;
4 for(i = 0; i < 64*1024; i+=2, p++) {
5       *p = 0xd673;
6 }
```

Listing 3.3: SRAM debug routine

3.6.15 Event handling entry points

This section describes the changes made to the code that handles interrupts, exceptions and system calls. A file for AVR32B (`arch/avr32/kernel/entry-avr32b.S`) was included in the Linux kernel, and we used this file as the basis for our code.

A significant part of the work was to get a clear understanding of all that happened in the assembler file. The file had a large number of labels without descriptive names, and many parts of the code needed commenting. We examined the file, analyzed the code flow, added a few comments, and changed many labels to more descriptive names.

Most of the actual code changes were due to the difference in the way events are handled on the AVR32 sub-architectures. The AVR32B sub-architecture will store return addresses and the status register in dedicated system registers, while the AVR32A sub-architecture saves them to the stack. We tried to optimize the stack layout based on this.

Original code

```
do_nmi_ll:
        sub     sp, 4
        stmts   --sp, r0-lr
        mfsr    r9, SYSREG_RSR_NMI
        mfsr    r8, SYSREG_RAR_NMI
        bfextu  r0, r9, MODE_SHIFT, 3
        brne    2f

1:      pushm   r8, r9  /* PC and SR */
        mfsr    r12, SYSREG_ECR
        mov     r11, sp
        rcall   do_nmi
        popm    r8-r9
        mtsr    SYSREG_RAR_NMI, r8
        tst     r0, r0
        mtsr    SYSREG_RSR_NMI, r9
        brne    3f

        ldmts   sp++, r0-lr
        sub     sp, -4          /* skip r12_orig */
        rete

2:      sub     r10, sp, -(FRAME_SIZE_FULL - REG_LR)
        stdsp   sp[4], r10      /* replace saved SP */
        rjmp    1b

3:      popm    lr
        sub     sp, -4          /* skip sp */
        popm    r0-r12
        sub     sp, -4          /* skip r12_orig */
        rete
```

Our code

```
do_nmi_ll:
        stmts   --sp, r0-lr
        sub     sp, 4           /* skip r12_orig */

        /* Check for kernel-mode. */
        lddsp   r9, sp[REG_SR]
        bfextu  r0, r9, MODE_SHIFT, 3
        brne    do_nmi_ll_kernel_fixup

do_nmi_ll_cont:
        mfsr    r12, SYSREG_ECR
        mov     r11, sp
        rcall   do_nmi
        tst     r0, r0
        brne    do_nmi_ll_kernel_exit

        sub     sp, -4          /* skip r12_orig */
        ldmts   sp++, r0-lr
        rete

        /* Kernel mode save */
do_nmi_ll_kernel_fixup:
        sub     r10, sp, -FRAME_SIZE_FULL
        stdsp   sp[REG_SP], r10 /* replace saved SP */
        rjmp    do_nmi_ll_cont

        /* Kernel mode restore */
do_nmi_ll_kernel_exit:
        sub     sp, -4          /* skip r12_orig */
        popm    lr
        sub     sp, -4          /* skip sp */
        popm    r0-r12
        rete
```

Figure 3.3: Example of entry point changes

Figure 3.3 shows the typical set of changes for an event handler. We can see that most of the `mfsr` (move from system register) and `mtsr` (move to system register) commands are gone. These were used to retrieve and set the return address and status register, which is unnecessary on the AVR32A architecture since they are already located on the stack. The stack layout changes can also be seen in the figure. The order of saves and restores from the stack is changed, and we no longer save the program counter and status register at all.

Stack layout changes

The kernel expects to be able to access the register data from when the exception, interrupt or system call occurred.

These registers should be saved in a structure named `pt_regs`. Therefore, the first that is done in the entry points is to save all registers to the stack in an order that matches the `pt_regs` structure.

When handling an exception or an interrupt, the AVR32B sub-architecture uses dedicated system registers to save the program counter and status register. These are automatically restored on exit. The AVR32B code will assemble the `pt_regs` structure from the current registers, and the program counter and status register from the system registers.

The AVR32A architecture pushes the program counter and the status register onto the stack. In addition, when handling interrupts, several extra registers are pushed onto the stack. The register layout is shown in figure 3.4.

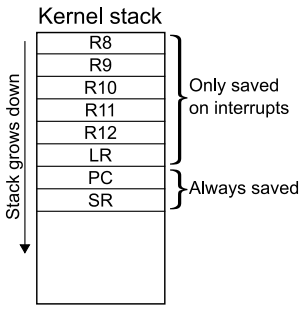


Figure 3.4: Kernel stack on interrupts/exceptions

We decided to reuse the program counter and status register which is already on the stack. Using the other registers that are automatically pushed during interrupts were also considered, but never implemented. The reason for this was that it would require pushing the program counter and status register on the stack when executing exceptions and system calls. This would add to the execution cost for all system calls and exceptions.

The entry-points will save `r0`-`r12`, the stack pointer and the link register to the stack. This, together with the program counter and status register already saved to the stack, forms most of the `pt_regs` structure. There is an additional element in the `pt_regs` structure, named `r12_orig`, used for system calls. This element is used to hold the original value of `r12` during system calls, but is unused in all other entry points. The final stack layout is shown in figure 3.5.

This change also meant that we had to change the `pt_regs` structure, so that it would match the order the registers were saved in the entry points. The `pt_regs` structure is part of the ptrace infrastructure in the kernel. The ptrace infrastructure is used for debugging applications, and the `pt_regs` structure is used for accessing registers of debugged programs from user space.

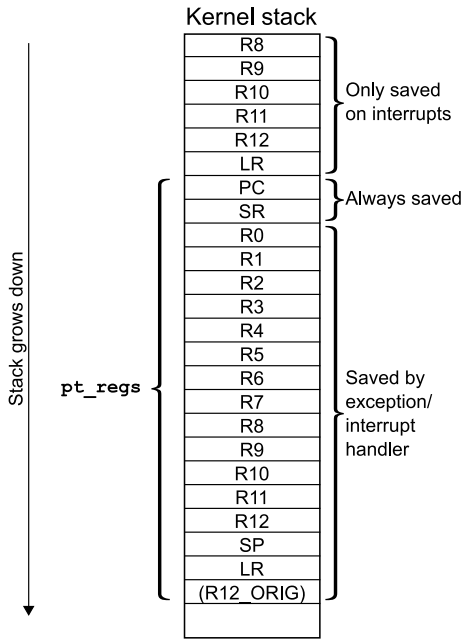


Figure 3.5: Kernel stack with `pt_regs`

The `pt_regs` structure is therefore part of the Application Binary Interface (ABI) interface exported to user space, and is located in `arch/avr32/include/asm/ptrace.h`. This file is installed by the kernel build infrastructure when `make headers_install` is executed.

This was a problem, because the kernel build infrastructure did not allow us to depend on configuration settings when installing header files. E.g., we could not install one set of header files for `CONFIG_SUBARCH_AVR32A` and one set of headers for `CONFIG_SUBARCH_AVR32B`.

Our original plan was to have the following layout of the `ptrace.h` file:

```
1 #ifdef CONFIG_SUBARCH_AVR32A
2 /* Our definition of pt_regs */
3 #else
4 /* Original definition of pt_regs */
5 #endif
```

Unfortunately, this did not work, since the `CONFIG_SUBARCH_AVR32A` option is not available outside the kernel build. Next, we tried to split the header file (`ptrace.h`) into two files, one of which was architecture dependent. The plan was to install `ptrace.h` and an additional architecture specific file – `ptrace-subarch.h`. Which file to be installed should depend on the kernel configuration options. This did not work either, because the configuration options are unavailable during `make headers_install`.

The final solution can be seen in appendix D.26. In this implementation, we rely on options set by the C compiler during compilation to select the correct version of `pt_regs`:

```
#ifdef __AVR32_AVR32A__
/* Our definition of pt_regs */
#else
/* Original definition of pt_regs */
#endif
```

This means that users of this code must select the correct architecture when compiling programs. This is something that must be done in any case, since various chips have support for different instructions. To compile a program for the UC3A0512ES, one can run: `avr32-uclinux-uclibc-gcc -march=ucr1 -mfdpic program.c -o program`

Debug entry point

The debug entry point is different from the others in that it has its return address and status register saved to a dedicated system register. As opposed to all other events on the AVR32A sub-architecture, nothing is saved to the stack automatically. This is similar to how all events are handled in the AVR32B sub-architecture.

We still need to have a complete `pt_regs` structure, and therefore need to save the return address and status register to the stack. Thus, the entry point for this event became slightly different. We leave a gap on the stack for the return address and status register, push all other registers. We then retrieve the status register and return address, and insert them on the correct location.

3.6.16 FDPIC ELF

There were several steps we did to add FDPIC ELF support to the Linux kernel. Since FDPIC ELF depends on architecture support, the configuration option contains a list of supported architectures. We added the AVR32 architecture to this list by appending `|| (AVR32 && !MMU)` to the end of this list. This change is shown on lines 55-56 of appendix D.5.

Next, we needed to add some extra fields to a data structure named `mm_context_t`. This structure contains information about each process' memory area. We added two variables to this structure – `exec_fdpic_loadmap` and `interp_fdpic_loadmap`. These are used to hold references to the load map for the executable and its interpreter. This change is contained outside of the FDPIC ELF patch because of the way we divided our patches, and can be seen on lines 39-40 of appendix D.11.

The FDPIC ELF loader code uses several functions and macros which the architecture is supposed to implement. We added these to `/arch/avr32/include/asm/elf.h`, and the changes can be seen on lines 15-46 of appendix D.5. The following was added to this file:

- `EF_AVR32_FDPIC`: A flag which we set in FDPIC ELF files to indicate that they are a FDPIC ELF file.
- `elf_check_fdpic`: A macro which checks that the `EF_AVR32_FDPIC` is set in a ELF file.

- `elf_check_const_displacement`: A macro which returns whether the file needs to be loaded contiguously in memory. We always return 0, since none of the files we generate has that requirement.
- `ELF_FDPIC_PLAT_INIT`: A macro which does architecture specific initialization when loading a FDPIC ELF file. We use this to load register `r0` with the pointer to the load map for the file. This enables the program to relocate itself.

We originally planned to depend only on the AVR32 architecture and add support for FDPIC ELF for AVR32 systems both with and without MMU. Unfortunately, the `mm_context_t` in the original code for AVR32 was an unsigned long. Changing this to a structure, so that the `exec_fdpic_loadmap` and `interp_fdpic_loadmap` elements could be added to the structure is possible. However, this would require many changes in various parts of the memory management code for the AVR32 systems with an MMU. We decided not to do this since we did not have the necessary time and hardware.

Register resetting

When the FDPIC ELF loader in Linux starts a new process, a reference to the load map is passed to the process via register `r0`. This reference passing was introduced with the patch for FDPIC ELF support listed in appendix D.5, inspired by the implementations for many other architectures. The existing code for AVR32 was not compatible with this convention, and set the value of every register to 0, overwriting the load map reference. The program then used this incorrect reference and tried relocate itself based on information found there. Because the relocation routine used invalid data, it ended up reading or writing to invalid addresses, which in turn caused an exception. The cause of the problem was discovered by inserting breakpoints and analyzing the processor registers during loading of the FDPIC file. We located, and removed the `memset` function call that cleared the registers, and made a separate patch for this. The patch is shown in appendix D.2. We were not sure about whether this was a good solution, but it was not denounced by anyone on the mailing list. A quick survey of implementations for other architectures suggested that it was common not to clear the registers. The mailing list discussion about this patch is shown in section 4.5.2.

3.6.17 Splitting of paging_init

During boot, the architecture specific initialization routine `setup_arch`, which is located in `arch/avr32/kernel/setup.c`, is called. This routine invokes a memory initialization function in `arch/avr32/mm/init.c` named `paging_init`. This function basically does three things: initialization of the MMU, pages and exceptions. Splitting this function would be a simple way to isolate the MMU initialization form the other two. Initialization of exceptions has no direct relation to memory management, so the code performing this was moved to a new function in the previously mentioned `setup.c`. Our final solution was to extract code from `paging_init` and create two new functions named `exceptions_init` and `mmu_init`. The `mmu_init` function and the call to it could then

be excluded whenever the `CONFIG_MMU` flag was unset. The patch for this modification can be seen in appendix D.3.

3.6.18 Use of existing macro

The patch listed in appendix D.4 changes code to utilize an existing macro. This macro returns a pointer to the register file for a procces. Using the same macro many places improves the structure, makes modification simpler, and ensures that the casting and calculation is done in the same way every place it is used. Of all the Linux patches submitted, this is the only one that is solely a structural change.

3.6.19 Patch summary

This section lists all the patches and describes those that perform small or uncomplicated changes not explicitly described in the previous sections.

0. **Cover letter:** This is not really a patch. It describes the purpose and scope the patches in the series.

1. **Network speed limiting:** Limits the network speed to 10 Mbit/s when the CPU is to slow for 100 Mbit/s. Described in section 3.6.6.

2. **Avoid register reset:** Disables zeroing of all register in `start_thread`. Described in section 3.6.16, “Register resetting”.

3. **Split paging function:** Split `paging_init` into separate functions. Described in section 3.6.17.

4. **Use task_pt_regs macro:** Simplifies some code by using an existing macro. Described in section 3.6.18.

5. **FDPIC ELF support:** Enables FDPIC ELF for AVR32. Described in section 3.6.16.

6. **Introduce cache and aligned flags:** This patch simply adds flags to `Kconfig` and `Makefile` that informs the compiler about the architecture and its features.

7. **Disable mm-tlb.c:** This patch disables the compilation of a file containing code not applicable for the UC3A.

8. **fault.c for !CONFIG_MMU:** This patch adds a new file is used instead of the file `fault.c` when compiling for MMU-less systems. The patch also adds the file to the appropriate make file.

9. **ioremap and iounmap for !CONFIG_MMU:** This patch adds a new file with dummy functions that replaces routines for mapping between physical and virtual memory.

10. **MMU dummy functions:** This patch introduces dummy functions to be used when an MMU is not available.

11. **mm_context_t for !CONFIG_MMU:** Described in section 3.6.16.

12. **Add cache function stubs:** This patch introduces dummy functions for CPUs without cache.

13. **copy_user.S for !CONFIG_NOUNALIGNED:** Descibed in section 3.6.13, in "Support for unaligned accesses".

14. **csum_partial: support for chips that cannot do unaligned accesses:** Described in section 3.6.12.

15. **Avoid unaligned access in uaccess.h:** This patch avoids an error occuring when an opcode-error is caused by an unaligned instructions. This patch was necessary because of a bug in the existing code, but became unnecessary after applying a patch from the mailing list posted by Håvard Skinnemoen. For the full discusson about this patch, see section 4.5.2.

16. **memcpy for !CONFIG_NOUNALIGNED:** Descibed in section 3.6.11.

17. **Mark AVR32B code with subarch flag:** Described in section 3.6.13, in "User space address ranges".

18. **mm-dma-coherent.c: ifdef AVR32B code:** This patch introduces a flag check that removes code only appropriate for CPUs with cache.

19. **Disable ret_if_privileged macro:** Described in section 3.6.13, in "User space address ranges".

20. **AVR32A-support in Kconfig:** This patch adds support for the AVR32A sub-architecture in the compilation configuration system.

21. **AVR32A address space support:** This patch introduces alternative version of macros that were not compatible with the address space layout of the UC3A.

22. **Change maximum task size for AVR32A:** Defining a upper boundary for a user space application does not serve any purpose without an MMU. This patch disables the boundary when compiling for UC3A, by setting the defined task size to `0xffffffff`.

23. **Fix ___range_ok for AVR32A in uaccess.h:** Described in section 3.6.13, in "User space address ranges".

24. **Support for AVR32A entry-avr32a.S:** Described in section 3.6.15.

25. **Change HIMEM_START for AVR32A:** The `HIMEM_START` address is used in relation with memory mapping. Without an MMU, mapping of physical memory is not possible. This patch therefore "disables" `HIMEM_START` in the same manner as described above for patch 22.

26. **New pt_regs layout for AVR32A:** Described in section 3.6.15, "Stack layout changes".

27. **UC3A0512ES interrupt bug workaround:** Described in section 3.6.10.

28. **UC3A0xxx support:** Described in section 3.6.3.

29. **Board support for ATEVK1100:** Described in section 3.6.2.

3.7 Toolchain adaptation

Initially, the toolchain did not support generating any type of relocatable executables for the AVR32 architecture. Since our system did not have an MMU, we needed the executables to be relocatable.

When we started on this task we had not yet chosen which binary format we should use. In an effort to understand the formats better, we did some initial testing with both formats. The testing we did with the Flat format is described in 3.7.4.

We followed another path which turned out to be a dead end. We tried to add a section with relocation information to normal executables. This is described in 3.7.5.

The rest of the section describes changes we made to add FDPIC ELF support to the toolchain – GNU GCC, GNU Binutils and uClibc.

When considering how to proceed, we quickly decided that it would be simplest to focus on static binaries. Shared libraries would introduce additional complexity, and we wanted to start simple.

3.7.1 GCC

The AVR32 specific GCC code is located under `gcc/config/avr32`, and all our changes are to files in that directory. We used the Blackfin and FR-V architectures as the base for our changes. These were located under `gcc/config/bfin` and `gcc/config/frv`.

`-mfdpic` flag

The first change we did to GCC was to add the `-mfdpic` flag. GCC has many target specific flags, and the convention is that the `-mfdpic` flag enables the FDPIC ELF target. For GCC to understand the `-mfdpic` flag, we had to add it to the `avr32.opt` file. This file combines option names, flags for options and the help text for options into a single file. Our changes to this file can be seen in the lines 31-33 of appendix F.2.

Options to linker and assembler

The whole point of adding the `-mfdpic`-flag is to be able to pass a different set of options to the assembler and linker when compiling FDPIC ELF files. The options passed to the linker and assembler are controlled by specifications in `linux-elf.h`. There were two options we changed – `ASM_SPEC` and `LINK_SPEC`.

For the assembler we only added the `-mfdpic` option when calling the assembler. This was done by adding `%{mfdpic}` to `ASM_SPEC`. The changes can be seen on lines 82-88 of F.2. For clarity, we also split the line into multiple lines.

We needed to use a different linker target when compiling FDPIC ELF files. To accomplish this, we added a single line to `LINK_SPEC`: `%{mfdpic:-mavr32linuxfdpic}`. This line will make GCC pass `-mavr32linuxfdpic` to the linker if `-mfdpic` is specified. The change can be seen on line 92 of F.2.

Options to self

Since FDPIC ELF files need to be relocatable, they should use position independent code. Normally, GCC creates code which isn't position independent for executables. To enable position independent code, one can pass one of `-fpic`, `-fPIC`, `-fpie` or `-fPIE` to GCC. So that the user should not have to specify this option, we can make GCC add the option to itself when `-mfdpic` is specified. This is done by adding `DRIVER_SELF_SPECS`. We set it to a line which basically says "If no other options enables or disables position independent code, set the `-fpie` option". This is done on lines 76-80 of F.2.

Later on we changed it so that the user would not have to specify the `-mno-init-got` option either. Normally GCC will create code which initializes the pointer to the GOT for each function call. Unfortunately, the code which initializes the pointer depends on the data segment being loaded at a constant offset from the code segment. This does not work when the FDPIC ELF file is fully relocatable, and it was therefore necessary to use this option. We added a line to `DRIVER_SELF_SPECS` which automatically sets the `-mno-init-got` option when `-mfdpic` is specified. This change can be seen in F.5, which is an unsubmitted patch for GCC.

`__AVR32_FDPIC__` define

To allow conditional compilation depending on whether a normal executable or a FDPIC ELF executable is created, we needed to add a preprocessor define. To be consistent with the Blackfin and FR-V architectures, we named it `__AVR32_FDPIC__`. This makes it possible to write code like:

```
#ifdef __AVR32_FDPIC__
/* Do something when creating FDPIC ELF files. */
#endif

#ifndef __AVR32_FDPIC__
/* Do something when not creating FDPIC ELF files. */
#endif
```

`crti.asm` GOT pointer

`crti.asm` is compiled during compilation of GCC, and the generated code is included in all compiled executables. This file initializes the GOT pointer unconditionally, including when the `-mno-init-got` option is specified. This overwrites the valid GOT pointer stored in the register. The code initializing the GOT pointer is the same as GCC uses elsewhere, and it will therefore fail in the same way when the program is not loaded contiguously into memory. Therefore we had to remove this code. This was done by adding an `#ifdef __AVR32_FDPIC__` around the code. The changes can be seen on line 34-67 in F.2.

For this `#ifdef` to work, we had to make GCC specify the `-mfdpic` flag when compiling `crti.asm`. This was done by adding `CFLAGS_FOR_TARGET=-mfdpic` when compiling GCC.

3.7.2 Binutils

Most of the changes necessary to produce FDPIC ELF files were to the GNU Binutils package. The patch we created for GNU Binutils can be found in appendix F.3. The changes done in GNU Binutils are inspired by the implementation done for the Blackfin and FR-V architectures, which can be found in `bfd/elf32-bfin.c` and `bfd/elf32-frv.c`. When looking at those two files, it was clear that they had a lot of the implementation in common, with a lot of code copied between those two files. We copied some code from those files into `bfd/elf32-avr32.c`, and used some of the code for inspiration. Since we implemented FDPIC ELF support without shared library support, a lot of the code we created became simpler.

New binary format

We needed to add a new binary format to the binary format library, located under the `bfd` directory. This was done by adding the following code at the end of `bfd/elf32-avr32.c`:

```
1 /* FDPIC target */
2 #undef TARGET_BIG_SYM
3 #define TARGET_BIG_SYM                  bfd_elf32_avr32fdpic_vec
4 #undef TARGET_BIG_NAME
5 #define TARGET_BIG_NAME                 "elf32-avr32fdpic"
6 #undef elf32_bed
7 #define elf32_bed                       elf32_avr32fdpic_bed
8
9 #include "elf32-target.h"
```

Later on we extended this code with some hooks to make it behave differently from the normal AVR32 target.

We also had to add this new target to the build files. To do this, we updated `bfd/config.bfd`, `bfd/configure`, `bfd/configure.in` and `targets.c`. These changes can be seen on lines 34-69 and 505-524 in appendix F.3.

New linker target

We needed to add a new linker target for generating FDPIC ELF binaries. The linker knows this as the "linker emulation", and the various targets are configured by shell scripts located in `ld/emulparams/`. We added a target named `avr32linuxfdpic` by creating a shell script named `avr32linuxfdpic.sh` in that directory.

Listing 3.4 is the script we ended up with. The script is heavily based on the `elf32bfinfd.sh` script and the `elf32frvfd.sh` script. Because the `avr32linuxfdpic` target is based on the `avr32linux` target, we begin the script by including the original `avr32linux.sh` script. We start by removing `STACK_ADDR` option since the stack is not mapped at a fixed address. Then we specify the output format by setting the `OUTPUT_FORMAT` option. The value is the internal name of the FDPIC ELF target, which is specified in the source code.

The `OTHER_READONLY_SECTIONS` extends the linker to understand the `rofixup` section. It specifies that the `rofixup` section should be included with the other read-only sections. It also creates two symbols which can be used in the program code: `__ROFIXUP_LIST__` and `__ROFIXUP_END__`. These are used in the assembler which handles the relocation.

```
1  . ${srcdir}/emulparams/avr32linux.sh
2
3  unset STACK_ADDR
4  OUTPUT_FORMAT="elf32-avr32fdpic"
5
6  OTHER_READONLY_SECTIONS="
7    .rofixup        : {
8      ${RELOCATING+__ROFIXUP_LIST__ = .;}
9      *(.rofixup
10     ${RELOCATING+__ROFIXUP_END__ = .;}
11   }
12 "
```

Listing 3.4: Linking configuration

We also had to add this new linker script to the various build files for the linker. These changes are located on line 584-644 of appendix F.3. They add the new `avr32linuxfdpic` target to the various files.

Stack size

The Linux kernel requires FDPIC ELF binaries to include its required stack size in one of the program headers, and will refuse to load a binary without that stack size. To support the new stack size, we needed to add a three new hooks for the `elf32-avr32fdpic` target. These were the following hooks:

- `elf_backend_always_size_sections`, which is called after all input files have been read, but before the linker has decided on the final size of the sections. This hook is handled by the `avr32_fdpic_always_size_sections` function.

- `elf_backend_modify_program_headers`, which is called just before the program headers are written to the output file. The `avr32_fdpic_modify_program_headers` function handles this hook.
- `bfd_elf32_bfd_copy_private_bfd_data`, which is used by tools which create copies on binary files. The `avr32_fdpic_copy_private_bfd_data` function handles this hook.

The program flow becomes:

1. The input files are read.
2. `avr32_fdpic_always_size_sections` is executed. If none of the input files has set the `__stacksize` symbol, this function will initialize it to 65536 bytes, which is default we have chosen. This function will also ensure that the `PT_GNU_STACK` segment is created and added to the program header.
3. `avr32_fdpic_modify_program_headers` is executed. This function will update the program header with the correct stack size from `__stacksize`.

The `avr32_fdpic_copy_private_bfd_data` function is only used by special tools, such as the `objcopy` command.

`objcopy` bug We had some problems setting the stack size to the right size when we compiled projects like BusyBox. When we compiled simple programs the stack size was correct, but when we compiled BusyBox the stack size became zero.

Our first theory was that error came during stripping of the binary, which is probably partly correct. As part of the debugging we tried to run `objcopy` standalone, and found that it did not retain the header correctly. After one pass through `objcopy` the stack size (`MemSiz`) was changed from 64KB to 44 byte, and after a second pass the stack size was zero.

```
1 Program Headers:
2   Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
3 - GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x10000 RWE 0x8
4 + GNU_STACK      0x000d44 0x00000000 0x00000000 0x00000 0x0002c RWE 0x8
```

Listing 3.5: Stack size after one pass through `objcopy`

```
1 Program Headers:
2   Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
3 - GNU_STACK      0x000d44 0x00000000 0x00000000 0x00000 0x0002c RWE 0x8
4 + GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4
```

Listing 3.6: Stack size after two passes through `objcopy`

The listings 3.5 and 3.6 show the difference between the output from `readelf` after the first and after the second run, formatted like an unified diff.

We assume that something goes wrong while reading the headers from the original file. After some debugging we found that if we specify the input format (listing 3.7) to `objcopy`, the `PT_GNU_STACK` header is retained correctly.

```
1 #avr32-uclinux-uclibc-objcopy -I elf32-avr32fdpic test.out
```

Listing 3.7: Command which correctly copies an object

To recreate the stack if it was removed during the build process a custom build script was used. This script inserted the stack size at pre-calculated offsets in the file. See section 3.10 for more information about this.

`rofixup` section

The FDPIC ELF binary format requires that statically linked executables are able to perform their own relocation. To accomplish this, they use a special section, named `rofixup`. This section is stored in the code segment, and contains a list of addresses that need to be updated. When the program is executed, it will look at the entries in that section to find addresses in the program which need to be updated.

The `rofixup` section is created by the function `avr32_rofixup_create` (line 87-119 in the patch in appendix F.3). This creates an empty section where relocation information is stored. The section is aligned on a word boundary by a call to the `bfd_set_section_alignment` function.

The function `avr32_check_relocs` iterates over all relocations and count any potential GOT and Procedure Linkage Table (PLT) reference. We extended this function to also count the number of potential addresses that should be in the `rofixup` section.

In the function `avr32_elf_size_dynamic_sections` we added code for calculating the size of the `rofixup` section. The size is the number of addresses we found during `avr32_check_relocs`, the number of GOT entries, and plus one for the terminator.

`avr32_rofixup_add_entry` is a helper function for adding entries to the `rofixup` section. `avr32_rofixup_add_relocation` uses the helper function to add relocations, and `avr32_rofixup_add_got` uses the helper function to add GOT entries to the section.

The code flow is as follows:

1. `avr32_check_relocs` calls `avr32_rofixup_create` to creates the `rofixup` section.

2. `avr32_check_relocs` iterates over all relocations. It counts the numer of relocations which should be included in the `rofixup` section, and saves the result in a counter.

3. `avr32_elf_size_dynamic_sections` calculates the final size of the `rofixup` section, and initializes it to that size.

4. `avr32_elf_relocate_section` iterates over all relocations, and adds relocations to the `rofixup` section by calling `avr32_rofixup_add_relocation`.

5. `avr32_elf_finish_dynamic_sections` adds all GOT entries to the `rofixup` section by calling `avr32_rofixup_add_got`. It then terminates the `rofixup` section by calling `avr32_rofixup_terminate`.

Assembler

The assembler is a part of the GNU Binutils package. There was only a minor change to the assembler – it had to handle the `-mfdpic` option correctly. When the assembler receives the `-mfdpic` option, it needs to set a flag in the output file, which indicates that the file is a FDPIC ELF file.

We added a new architecture specific ELF flag for AVR32 – the `EF_AVR32_FDPIC` flag. If this flag is set in an ELF file, the file is a FDPIC ELF file. The assembler was changed to set this flag when the `-mfdpic` option is set.

3.7.3 uClibc

The uClibc library needed several adaptions – some to support the UC3 family of microcontrollers, and some to support FDPIC ELF on the AVR32 architecture.

UC3 support

The first we did was to add an UC3 option to the build configuration, so that it was possible to select the UC3 family of processors in the configuration system. This option allows us to do conditional compilation of code depending on which processor is selected. It is also used to select the correct compiler flags – in our case `-march=ucr1` should be specified to generate code which is compatible with the UC3A0512ES. The -ES version of the UC3A0512 microcontroller only implements revision 1 of the AVR32 architecture, so we need to specify `-march=ucr1`. The changes can be seen on lines 34 and 54-56 of appendix F.4.

Unaligned memory accesses

The `libc/string/avr32` directory contains optimized variants of several standard C functions dealing with strings and blocks of memory: `bcopy`, `bzero`, `memcmp`, `memcpy`, `memmove`, `memset`, `strcmp`, `strlen`

We looked over these implementations, and identified three functions which perform unaligned accesses in some situations: `memcmp`, `memcpy` and `memmove`.

We added code to these functions to handle the unaligned case. This code is only activated when compiling for the UC3 family of microcontrollers.

FDPIC support

When adding FDPIC ELF support, we used the code from the Blackfin and FR-V architectures as inspiration. The first we added was support for doing relocation during program startup.

This code was added to `crt1.S` in the `libc/sysdeps/linux/avr32` directory. That file contains the entry point of the program, where the execution first starts when Linux passes control to the program. There were already two code paths in that file, depending on whether uClibc was compiled as a static or shared library.

We added a third code path to that file, which is enabled when the file built with FDPIC ELF support. This code path can be seen on line 176-226 of appendix F.4, and does the following:

1. Call the `__self_reloc` function with the following arguments:
   - The load map of the program – created by the Linux kernel.
   - The original (unrelocated) offset of the `rofixup` section, where relocation information is stored.
   - The original offset of the GOT.
2. The `__self_reloc` function uses the information in the `rofixup` section to update all pointers in the program.
3. The `__self_reloc` returns the relocated pointer to the global offset table. This offset is loaded into register `r6`, which is the register designated to hold a pointer to the GOT.
4. Control is passed to the `__uClibc_main` function, which is the main entry point for uClibc.

We also added a new C-file – `crtreloc.c`. This file can be seen on lines 258-348 of appendix F.4. It is this file that contains the `__self_reloc` function. It also contains a function named `__reloc_pointer`, which is used by the `__self_reloc` function. This function takes in a pointer, and returns the relocated pointer. The `__reloc_pointer` function is copied from the FR-V file `libc/sysdeps/linux/frv/bits/elf-fdpic.h`. An identical function can also be found in `libc/sysdeps/linux/bfin/bits/elf-fdpic.h`.

GOT pointer

Like GCC, uClibc also includes `crti.S`, which does the same thing as `crti.asm` in GCC (see section 3.7.1). We did the same change to this file as we did to the GCC file, and deactivated the initialization of the GOT pointer when FDPIC ELF is enabled. The same change also had to be done in two other assembler files – `syscalls.S` and `vfork.S`. The changes can be seen on lines 227-257 and 349-405 of appendix F.4.

3.7.4 `elf2flt`

During development we investigated the possibility of using the Flat binary format. `elf2flt` is the usual approach used generate Flat binaries. An attempt was made to identify the necessary changes to this tool in order to be able to produce Flat binaries. Our resulting code, with which we were able to produce some unstable results, is listed in appendix G. This evaluation was done at a time when the development of FDPIC ELF was stuck in some problem that we did not figure out right away.

In development of this patch the first milestone set was to add the AVR32 target skeleton. This would let `elf2flt` accept AVR32 as a target. When this was done it

was possible to add tracing information and add necessary code at places where it was necessary. A few relocation definitions were added to code which iterated the symbols. A few `printf`s used to trace the iteration still remains. If this code should be used, these would have to be removed.

Some changes were done to the Linux kernel as well, mostly in copying code from other architectures into the AVR32 architecture. These changes were reverted when we continued developing FDPIC ELF.

Even though we did not use the code developed in this exploration, the process gave us better knowledge about how the toolchain work and helped us to get further with FDPIC ELF.

3.7.5 PIE support

Another dead end that we investigated, was to create position independent executables via the `-fpie` flag to GCC. This was in an attempt to add the `DYNAMIC` section to normal executables, which we had assumed was required for FDPIC ELF support. What we discovered early on was that GNU Binutils for the AVR32 architecture did not include the linker script required for position independent executables. Our supervisor at Atmel, Håvard Skinnemoen, suggested that we added `GENERATE_PIE_SCRIPT=yes` to the `ld/emulparams/avr32linux.sh` script in GNU Binutils. This would make GNU Binutils generate the correct linker script.

With the correct linker script, we were able to generate position independent executables, which contained the `DYNAMIC` section. This section contained relocation information for the executable, so it could in theory be relocated by a loader. Unfortunately, the generated executable was basically a shared library, which was dependent on an external program for loading. This is unfortunate, as we could not generate static binaries with this method. We also discovered that FDPIC ELF executables did not require a `DYNAMIC` section when statically linked, and that they instead depended soley on the `rofixup` section for relocation.

3.8 SRAM optimization

The external SRAM severely limits the execution speed. We currently use three CPU clock cycles for each 16-bit read or write. According to the SRAM datasheet, it should be possible to accomplish this in a single clock cycle. The UC3A doesn't have any cache, so the processor has to use three cycles for every memory access. When code is executed from external SRAM, the throughput of the microcontroller will be reduced to at least three cycles per instruction. When executing code from internal SRAM, the throughput should be closer to one cycle per instruction.

We did a survey of the various SRAM signal lines, and identified three potential problems:

- Some of the signal lines are routed out of the UC3A0512 microcontroller in two locations.

- The joystick on the EVK1100 was connected to three of the address lines.
- One of the LEDs on the EVK1100 was connected to the chip-select line of the SRAM chips.

3.8.1 Routing of signals

All of the SRAM control signals and some of address lines can be routed out of the UC3A0512 microcontroller on several of the pins. When we originally added support for external SRAM to U-Boot, we routed the signals out on all available locations. This was done for simplicity as a temporary implementation.

We wanted to check whether routing the signals to multiple pins in this way, would degrade the output signals. To test this we made a simple change to U-Boot that disabled the control signals not in use. The change, shown in listing 3.8, comments out a part of one of the bitmasks that selects pin functionality. After this change, SRAM would still not work with increased speed settings.

```
diff --git a/cpu/at32uc/at32uc3a0xxx/portmux.c b/cpu/at32uc/at32uc3a0xxx/portmux.c
index a796f22..5b65ee0 100644
--- a/cpu/at32uc/at32uc3a0xxx/portmux.c
+++ b/cpu/at32uc/at32uc3a0xxx/portmux.c
@@ -48,7 +48,7 @@ void portmux_enable_ebi(unsigned int bus_width, unsigned int addr_
    width,
         */

        portmux_select_peripheral(PORTMUX_PORT(0),
-                                                 0x0003C000 |
+                                                 /* 0x0003C000 | */
                                                  0x1E000000, PORTMUX_FUNC_C, 0);
        portmux_select_peripheral(PORTMUX_PORT(1),
                                                  0x00000010 |
```

Listing 3.8: Disable SRAM signals

3.8.2 Joystick pull-up conflict

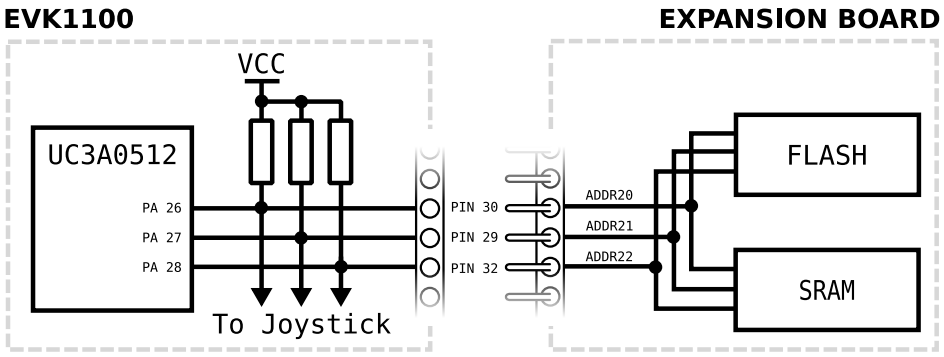


Figure 3.6: Joystick pull-up conflict

On the EVK1100, there are five signal lines between the joystick and the microcontroller. Three of these lines are also routed to the expansion header, and is used for EBI by the

memory expansion card. The joystick is accompanied by pull-up resistors connected to these lines, and could therefore potentially interfere with the memory bus. In an attempt to test this hypothesis, those three resistors were removed. Unfortunately, we were still unable to increase the speed of the memory.

3.8.3 LED resistor conflict

Of all of the 8 individually controllable LEDs, one (LED3[4]) shared a signal line with the memory expansion board. LEDs draw a significant amount of current (typically 20mA[3]), and a LED could considerably affect the rise and fall times of the memory bus.

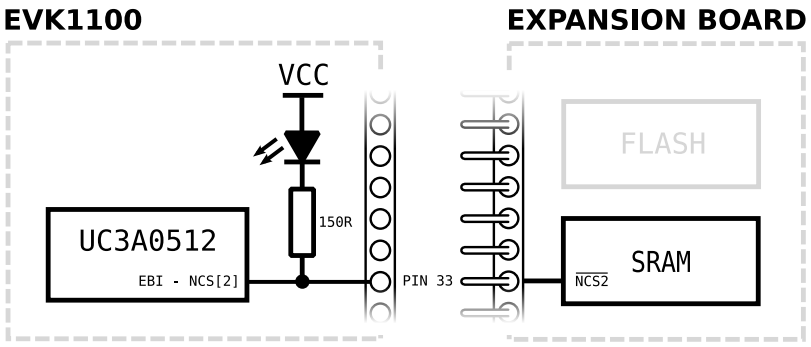


Figure 3.7: LED conflicting with the EBI bus

Figure 3.7 shows how the microcontroller, LED3 and the memory are interconnected. We tried to remove the LED, but this did not improve the performance of the memory.

3.9 SPI chip enable

While attempting to adapt and activate the SPI driver, unexpected crashes started to occur. Linux did not print a proper stack trace, and when single stepping, it was discovered that the processor jumped to the exception handler for illegal opcodes. After jumping to that handler, we could see that the executed code in the handler was not the same as the code that should be stored there.

We suspected that the memory had been corrupted by some software error. We tried to set memory breakpoints on the exception handler code, to check when it was written to. The breakpoint never triggered.

We then looked closer at the instructions which were executed right before the crash. These instructions dealt with activating a pin connected to a SPI chip enable line. Further inspection of the line showed that it was also connected to the chip enable pin for the flash chip on the memory expansion card.

[4]This LED is numbered LED2 in the schematics

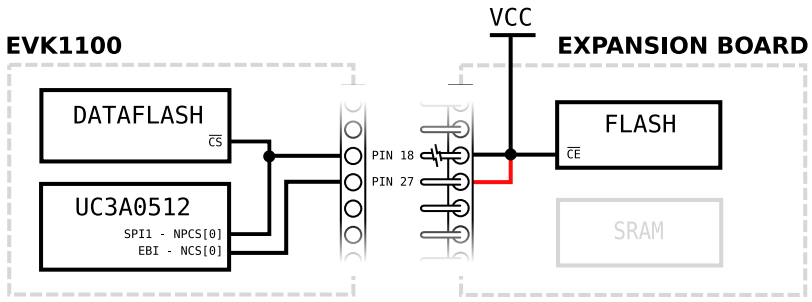


Figure 3.8: Chip enable pin conflict

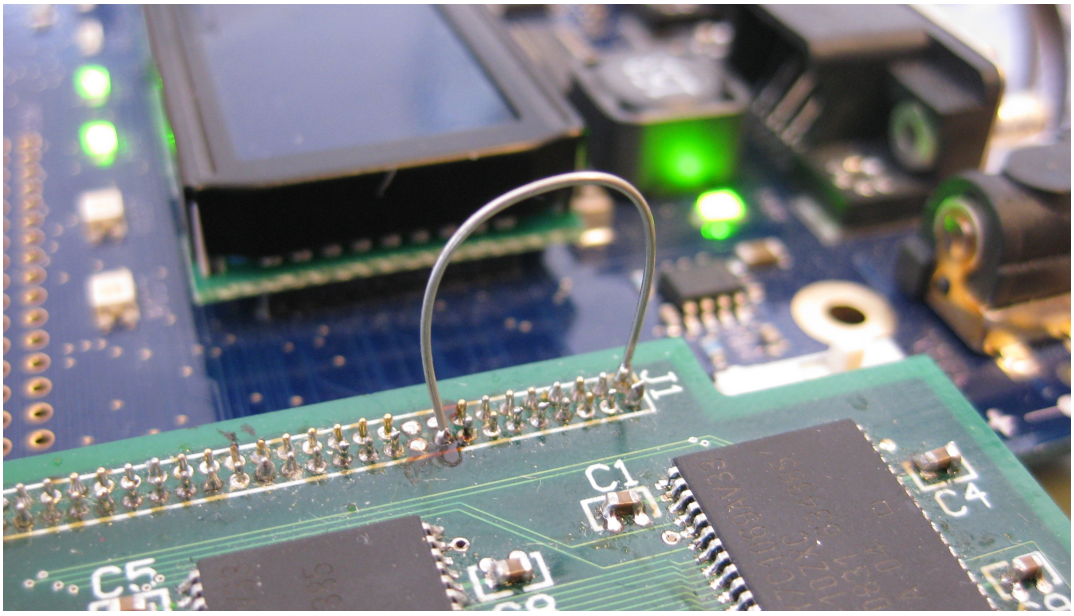


Figure 3.9: Wire soldered from chip enable pin to nearby VCC

The flash chip on the memory expansion card designed by Atmel (see 2.15) has one pin, chip enable, that is used for activating the chip. According to the schematics, this pin is routed to both pin 18 and 27 on the header. On the EVK1100, pin 18 and 27 on the header is routed to the pins PA14 and PA25, which are two possible physical pins that a chip enable signal can be routed to. Since the DataFlash on the EVK1100 already occupies pin PA14, activating SPI for this device also enables the flash chip on the expansion board. This ultimately leads to the flash chip interfering with the memory bus. This problem was solved by physically removing a pin from the expansion board header so that the chip enable signal could not reach the flash (illustrated in figure 3.8). It should then be possible to use the other pin on the expansion board to control the chip enable line, but after visual inspection and testing with a multimeter, it was found that the chip enable pin from the flash chip is only connected to pin 18, and not pin 27.

This missing connection is illustrated with a red line in figure 3.8. Since the path to pin 18 was cut, and pin 27 was unconnected, a short wire was soldered from `VCC` to pin 27 on the expansion board to avoid a floating chip enable input on the flash. This is shown schematically in figure 3.8, and figure 3.9 shows a picture of the soldered wire. The end result is that the flash chip is always deactivated.

3.10 BusyBox

One of the tasks (requirement 13) was to find a set of suitable applications for this platform. With a working toolchain it was possible to build BusyBox for this processor. BusyBox contained a lot of useful tools which could be used. More about BusyBox in section 2.14.

```
1 #!/bin/bash
2 INSTALLPATH=/tftpboot/evk1100
3 make V=1 CFLAGS="-mno-init-got -mfdpic" LDFLAGS="-static -mfdpic" install
4 echo -ne '\x00\x01\x00\x00' | dd of=$INSTALLPATH/bin/busybox seek=136 bs=1 conv=
      notrunc
```

Listing 3.9: Custom shell script for building BusyBox

Line 3-4 is a workaround for a bug which surfaced when an object was stripped for debug information. More about this bug in section 3.7.2. The magic number in the script, 136, is the location of the stack size (`MemSiz` of `GNU_STACK`). The formula for finding the offset to the stack size definition is simple. The necessary information can be found in listing 3.10, which is the output from the program `readelf` when given the compiled version of BusyBox as input. Start of program headers (52 byte) + Number of section headers before `GNU_STACK` (2) * Size of program headers (32) + Word size (4 byte) * Number of elements in the program header before `MemSiz` (5). The numbers used here is marked with a blue color, and the changed number is colored red. The stack size was 0x00000 before the provisional fix was applied and 0x10000 (64KB) afterwards.

```
1
2 # avr32-uclinux-uclibc-readelf  --file-header --program-headers busybox
3
4 ELF Header:
5   Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
6   Class:                             ELF32
7   Data:                              2's complement, big endian
8   Version:                           1 (current)
9   OS/ABI:                            UNIX - System V
10  ABI Version:                       0
11  Type:                              EXEC (Executable file)
12  Machine:                           Atmel AVR32
13  Version:                           0x1
14  Entry point address:               0x109c
15  Start of program headers:          52 (bytes into file)
16  Start of section headers:          218888 (bytes into file)
17  Flags:                             0x6
18  Size of this header:               52 (bytes)
19  Size of program headers:           32 (bytes)
20  Number of program headers:         3
21  Size of section headers:           40 (bytes)
```

```
22   Number of section headers:         12
23   Section header string table index: 11
24
25 Program Headers:
26   Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
27   LOAD           0x000000 0x00001000 0x00001000 0x3424c 0x3424c R E 0x1000
28   LOAD           0x03424c 0x0003624c 0x0003624c 0x01464 0x163b0 RW  0x1000
29   GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0xxxxxx RWE 0x4
30
31  Section to Segment mapping:
32   Segment Sections...
33    00     .init .text .fini .rodata .rofixup
34    01     .data.rel.ro .got .data .bss
35    02
```

Listing 3.10: Output from readelf

3.11 Obtaining and distributing source code

The five software units modified during this project were Linux, U-Boot, uClibc, GCC and GNU Binutils. We tracked all our modifications by using the Git revision control system. This section describes how the source code was obtained and distributed. Note that our current version of all the patches can be found in the digital appendices in this report.

3.11.1 Buildroot

Buildroot[5] is a tool for generating a cross-compilation toolchain and root file system for embedded systems. Some of the patches we used were extracted from Atmel's Buildroot release[6]. The most current version at the time of download was version 2.3.0.

3.11.2 GCC

When GCC was downloaded, version 4.2.2 was the most current version for which Atmel provided patches. The most current patch from Atmel at the time was version 1.1.3. Atmel's Buildroot package included their patch and several other patches that could be useful for us. We applied all patches from the Buildroot package to GCC 4.2.2, and used this as the base for our development.

Since there is no dedicated mailing list for the AVR32 toolchain, the patches against GCC were submitted to the AVR32 Buildroot list[7].

[5] http://buildroot.uclibc.org/
[6] http://www.atmel.no/buildroot/
[7] http://avr32linux.org/archives/buildroot/

3.11.3 GNU Binutils

When GNU Binutils was downloaded, version 2.18 was the newest version for which Atmel provided patches. The most current patch from Atmel were at the time version 1.0.1. Atmel's Buildroot package included their patch, which we applied to the official GNU Binutils source[8].

Since there is no dedicated mailing list for the AVR32 toolchain, the patches against Binutils were submitted to the AVR32 Buildroot list[7].

3.11.4 uClibc

When uClibc was downloaded 0.9.30, was the most current version. uClibc was downloaded from uClibc's download page[9].

Since there is no dedicated mailing list for the AVR32 toolchain, the patches against uClibc were submitted to the AVR32 Buildroot list[7].

3.11.5 `elf2flt`

`elf2flt` was downloaded from uClinux.org's official CVS repository[10] at the 6th of March 2009. We experimented with `elf2flt`, but no useful results were achieved, so no patches for `elf2flt` were submitted to the maintainers.

3.11.6 U-Boot

We had created a modified version of U-Boot during our project the fall of 2008, so the U-Boot source code was already in our possession. Håvard Skinnemoen at Atmel Norway maintains a repository of U-Boot for development and testing of AVR32-specific code. Skinnemoen can decide whether to add patches to his repository, and whether they eventually should be merged into the official U-Boot Git repository.

During this project, one revised patch series for U-Boot was prepared and submitted to the official U-Boot mailing list on the 23rd of January. This patch series is based on the earlier submitted patches, and addresses the feedback and criticism received on the mailing list. Only the modifications of U-Boot described in section 3.4.6 and 3.4.5 were done after the submission this patch series. These were therefore never organized into patches or submitted to any mailing list, but are listed in appendix C.

Some of the patches have already found their way into the current release of U-Boot[11], and some of them is pulled into the *next*[12] repository.

[8] http://ftp.gnu.org/gnu/binutils/
[9] http://www.uclibc.org/downloads/
[10] http://cvs.uclinux.org/cgi-bin/cvsweb.cgi/elf2flt/
[11] http://www.denx.de/wiki/U-Boot/UbootStat_2009_03
[12] *next* refers to the branch currently under development that is going to lead to the next release

3.11.7 Linux

The Linux kernel source code was obtained from Linus Torvalds' kernel tree on kernel.org during our project the fall of 2008, so the source code was already in our possession. The specific version that was originally downloaded was v2.6.27-rc6-99-g45e9c0d. In our case, merging was not highly prioritized and therefore only done once. We merged so that we used the most current stable at time (early January) as a basis, which was version 2.6.28.1[13].

We separated our changes into logical units, and ended with up 29 patches. These could be categorized into the following four categories:

- 19 patches to prepare for AVR32A support.
- 7 patches which add AVR32A support.
- 2 patches which add UC3A support.
- A patch to add support for the board we used (EVK1100).

These patches were submitted to the AVR32 kernel list[14] with a short description of the patch series. The patch series is listed as a whole with the cover letter in appendix D.

3.11.8 BusyBox

At the start of the work with this thesis, the BusyBox source code was downloaded from the official BusyBox website[15]. The latest version at the time was version 1.13.2. Since no changes were made to the BusyBox source code, there was no need to submit patches to the maintainers.

[13] http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.28.1.tar.bz2
[14] http://avr32linux.org/archives/kernel/
[15] http://www.busybox.net

Chapter 4

Testing and results

In this chapter, we present the tests used to determine which of the requirements defined in section 1.3.1 were met. The result of each test will also be given, and any abnormal results will be discussed. This chapter follows the structure of the requirements list. For each individual requirement, a corresponding test and result is listed.

All the tests assume that the hardware is connected as shown in figure 3.2. The U-Boot boot image is programmed to the internal flash of the microcontroller, and the computer is serving the Linux kernel image and the root file system to the board via the services shown in the figure.

Included in this chapter is also the feedback we received when submitting patches. This is organized as a list of each patch we received feedback on, and the feedback we received. Our responses to the feedback is also included.

4.1 U-Boot

4.1.1 SPI support, requirement 1

Result: Not implemented.

4.1.2 Loading from DataFlash or SD card, requirement 2

Result: Not implemented.

4.1.3 Patch cleanup, requirement 3

Result: Submitted.

A series of patches were submitted. Section 3.4 describes the changes done to U-Boot, and section 3.11.6 describes what has been submitted.

4.2 Linux

The Linux test project[1] could be used to test the robustness of the system, but this requires that the toolchain has reached sufficient maturity. Since that is not yet the case here, we were not able to locate any automatic testing software which could be used within our time frame. Only small parts of the system is tested here, and more comprehensive testing should be performed before putting the code into a production environment.

4.2.1 Booting Linux kernel, requirement 4

Testing of this requirement is done by letting U-Boot load and start the kernel. A init program must be placed on the appropriate place on the network file system. The kernel should then:

a. Give reasonable output to serial console.

   Verified by starting a terminal program and watching the output.

b. Bring up networking.

   Verified by running `ifconfig` when the system has booted and reading the output. If necessary `ifconfig` is used to set a static IP. If that gives reasonable output a ping to the server should be executed.

c. Receive network configuration using DHCP.

   Verified by running `ifconfig` after a reboot. The interface should now have received an IP-address from the DHCP server.

d. Mount necessary file systems:

   (i) NFS root file system.
   (ii) proc file system.
   (iii) sysfs file system.
   (iv) devpts file system.
   (v) devshm file system.

   This is checked by executing the command `mount`. The output should be a list containing the file systems listed above. On each file system it should be verified that files could be accessed.

e. Load and execute init application.

Result: Passed. All tests were successfully executed.

[1] http://ltp.sourceforge.net/

4.2.2 Running user space binaries, requirement 5

This can only be tested if test 4.2.1 passed.

A simple program should be compiled, copied to the root file system and executed. It should be verified that the program give the correct output.

Result: Passed. The binaries execute and give correct output.

4.2.3 Hardware support, requirement 6

LEDs, requirement 6a

The LEDs can be tested by writing to their trigger files. This can be tested by enabling the heartbeat function. First it must be checked that the LED is not already blinking a heartbeat, so that the test result is proper.

The heartbeat of LED1 is then enabled by writing:

```
1 echo 'heartbeat' > /sys/class/leds/led1/trigger
```

Listing 4.1: Enabling LED1

After executing that command LED1 should give a blinking heartbeat.

Result: Passed.

DataFlash, requirement 6b

This requirement was not implemented, and therefore no test was written.

Result: Not implemented.

LCD, requirement 6c

This requirement was not implemented, and therefore no test was written.

Result: Not implemented.

SD Card, requirement 6d

This requirement was not implemented, and therefore no test was written.

Result: Not implemented.

SPI, requirement 6e

This requirement was not implemented, and therefore no test was written.

Result: Not implemented.

DMA, requirement 6f

This requirement was not implemented, and therefore no test was written.

Result: Not implemented.

Network adapter, requirement 6g

Tested by assigning an IP address to the network adapter, and using `ping` to test connectivity to another computer connected to the same network.

Result: Passed.

4.2.4 Exceptions, requirement 7

We identified four exception entry points that should be possible to trigger from an user space application. These were:

- `handle_address_fault`, triggered by unaligned reads/writes.
- `do_bus_error_write`, triggered by writing to invalid addresses.
- `do_bus_error_read`, triggered by reading invalid addresses.
- `do_illegal_opcode_ll`, triggered by various invalid or illegal instructions.

Unaligned accesses

We created two tests for `handle_address_fault`. The first test, appendix I.1.1, tests unaligned read, while the second test, appendix I.1.2, tests unaligned writes. Both tests install an handler for the SIGBUS exception, and attempts to access an unaligned pointer.

Result: Both tests for unaligned accesses passed. The application received a SIGBUS exception from the kernel when attempting an unaligned access.

Invalid addresses

Two tests were created, one for testing of `do_bus_error_write` and one for testing of `do_bus_error_read`. Both tests attempt to access a memory area that does not exist on the UC3A0512 microcontroller. The first test, appendix I.1.3, tests invalid reads, while the second test, appendix I.1.4, tests invalid writes.

Result: The tests for reads and writes triggered an "oops" from the kernel when attempted. This is because the handlers used were the same as for the implementations with MMU support. The only way to receive this error with a processor with an MMU would be if the kernel made an error, and assigned invalid memory to the application. One could argue that we should pass the error to the application, instead of handling it in the kernel, but we decided not to do this. If MPU support was added, the error should be handled similarly as with an MMU, and until then the program is terminated.

Another problem we discovered was that the `do_bus_error_read` entry point was mislabeled in the original source code we based our work on. It turned out that `do_bus_error_read` was triggered by instruction reads, not data reads. The two exception handlers call the same function, with a parameter to show whether the access was a read or write. This is used to print a log line: `Bus error at physical address 0x00100000 (write access)`. The mislabeling caused both of our tests to be logged as a write access.

Invalid opcode

This handler is used by many exceptions. It is used when the opcode is unknown to the microcontroller, when the opcode is known but unsupported and when the application has insufficient privileges to execute the instruction. The logic when handling the different types is the same, so we decided to test only when the opcode was unknown to the microcontroller.

To test this, we attempt to use the `rsubeq` instruction. This instruction requires revision 2 or higher of the AVR32 architecture, while the UC3A0512ES only supports revision 1 of the AVR32 architecture. When an illegal opcode is found, the Linux kernel should deliver a SIGILL exception to the program.

We created two tests, to test two different cases of invalid operations. Appendix I.1.5 tests the case when the instruction is aligned on a four byte boundary. The other test, appendix I.1.5 tests the case when the instruction is aligned on a two byte boundary, but not on a four byte boundary. Both alignments are valid for all instructions, but the handler for invalid opcodes makes an assumption which did not hold for the UC3A0512 microcontroller. The handler code, which is shared between AVR32A and AVR32B assumes that it can read unaligned 4 byte words.

We had a patch which enabled reading of unaligned words (appendix D.15), but Håvard Skinnemoen commented that this patch should not be necessary. If that patch is dropped, then this code needs to be fixed. See the comments for patch 15 in section 4.5.2.

Result: Both tests pass with our patch applied, but if we remove our patch, the second test fails. When passing the test, the application receives a SIGILL exception from the kernel.

4.2.5 Code submission, requirement 8

Result: Almost everything was submitted. We did not submit the SPI changes, since these were incomplete, and largely irrelevant with the restructuring of the peripheral DMA code done by Atmel (see 3.6.9).

Section 4.5.2 summarizes the received feedback.

4.3 Toolchain

4.3.1 Select binary format, requirement 9

Result: FDPIC ELF was selected (see 3.5).

4.3.2 Produce binaries, requirement 10

Two simple example programs written in C (listing 4.2 and listing 4.3) were compiled with GCC.

The first program is a simple program which does not use large parts of the C library. It only invokes the `write` system call, to put "Hello!" on standard output.

```
1 #include <unistd.h>
2
3 int main(int argc, char *argv[])
4 {
5   write(1, "Hello!\n", 7);
6   return 0;
7 }
```

Listing 4.2: hello.c

The second program uses larger parts of the C library. It invokes `printf`, which uses the standard input/output part of the C library. This part will not work without correct relocations, because there are several data pointers used. For example, `printf` uses the `FILE *stdout` pointer, which only works with correct data relocation.

The `42` parameter to `printf` is mainly included to prevent optimization. If no arguments are given, the compiler will optimize the `printf` call to a `puts` call.

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5   printf("Hello world! %d\n", 42);
6   return 0;
7 }
```

Listing 4.3: helloworld.c

```
avr32-uclinux-uclibc-gcc -mfdpic hello.c -o hello
```

Listing 4.4: Compiling hello.c

The output should be copied to the root file system and executed. The output should be "Hello!" for the first program and "Hello world!" for the second.

Result: Passed.

4.3.3 Produce libraries, requirement 11

This is not implemented and therefore not tested.

Result: Not implemented.

4.3.4 Code submission, requirement 12

Result: Code submitted to the avr32linux.org's mailing list as discussed in section 3.11.2, 3.11.3 and 3.11.4.

4.4 Linux user space

4.4.1 BusyBox, requirement 13

BusyBox was compiled with the applets listed in requirement 13. Each applet was then in turn invoked from the Hush shell.

Result: Passed. Hush and all the other applets listed in the requirement executed, produced the expected output and terminated successfully. However, the applets occasionally caused the system to run out of free memory. Because the tasks that these applets perform are well known, we choose not to list the output of every applet and how they were invoked. When an application causes the system to run out of memory, the kernel fails to recover, and will crash if the system runs out of memory again.

We have looked at the error which occurs when out of memory, but have been unable to determine the cause. Due to our limited time frame, we have been unable to spend very much time on this bug.

4.5 Patch submission feedback

In this section we list the feedback to the patches we submitted, with one section for each patch series. For brevity, some of the feedback may be omitted, slightly shortened, rephrased or summarized.

4.5.1 U-Boot

The most relevant feedback to the submitted U-Boot patch series is presented in this section. All the e-mails discussing these patches can be found in the official U-Boot mailing list archive[2].

[2] http://lists.denx.de/pipermail/u-boot/2009-January/thread.html#45925

[PATCH v2 1/9] Fix IP alignement problem

The IP alignment patch was applied to the network branch by Ben Warren.

[PATCH v2 2/9] AVR32: Make cacheflush CPU-dependent

```
Applied to evk1100-prep and merged into next. I'll send it upstream
    as
soon as it's fine with Wolfgang.

Btw, I had to rebuild my next branch since it's become a bit stale.
Since all the non-merge commit IDs are the same, I hope it won't
    cause
any problems -- please let me know if you see any weird merge issues.
```

Reply from Håvard Skinnemoen

[PATCH v2 3/9] AVR32: Move addrspace.h to arch-directory, and move some functions from io.h to addrspace.h

```
Applied to evk1100-prep, thanks.
```

Reply from Håvard Skinnemoen

[PATCH v2 4/9] AVR32: Make GPIO implementation cpu dependent

```
Applied to evk1100-prep, thanks.
```

Reply from Håvard Skinnemoen

[PATCH v2 5/9] AVR32: macb - Disable 100mbps if clock is slow:

A white space unfortunately found it's way into our patch.

```
-       adv = ADVERTISE_CSMA | ADVERTISE_ALL;
+       adv = ADVERTISE_CSMA | ADVERTISE_ALL ;
??
```

Reply from Ben Warren

This patch sparked a debate on the mailing list. The most relevant replies are listed below. The rest of the e-mails in this discussion can be found on the web page listed at the beginning of this section.

```
> +#ifdef CONFIG_MACB_FORCE10M
> +       printf("%s: 100Mbps is not supported on this board - forcing
    10Mbps.n",
```

```
> +                netdev->name);
> +
> +        adv &= ~ADVERTISE_100FULL;
> +        adv &= ~ADVERTISE_100HALF;
> +        adv &= ~ADVERTISE_100BASE4;
> +#endif
        not a fan
        could you be more specific about the problem?
```

Reply from Ben Warren

```
On the EVK1100 board, the CPU (UC3A0512) is connected to the PHY via
an RMII bus. This requires the CPU clock to be at least 50 MHz.
Unfortunately, the chip on current EVK1100 boards may be unable to
    run
at more than 50 MHz, and with the oscillator on the board, the
    closest
frequency we can generate is 48 MHz.

This patch makes it possible to limit the macb to 10 MBit for this
case. We are open for suggestions for other solutions.
```

Our reply

```
How about using a PHY capability override CONFIG.
Something like this:

#if defined(CONFIG_MACB_PHY_CAPAB)     <-- insert better name here
adv = ADVERTISE_CSMA | CONFIG_MACB_PHY_CAPAB;
#else

adv = ADVERTISE_CSMA | ADVERTISE_ALL
#endif

Just an idea...
```

Reply from Ben Warren

[PATCH v2 6/9] AVR32: macb - Search for PHY id

This patch was added to the network branch of U-Boot by Ben Warren. We asked Ben if this code should be omitted in any newer versions of this patch series. Ben's answer is quoted below.

```
Correct. You've done a good job of making them orthogonal to the
    rest of your code, so the net repo is where they belong. I'll
    issue a pull request after doing a bit of testing.
```

Reply from Ben Warren

[PATCH v2 7/9] AVR32: Must add NOPs after disabling interrupts for AT32UC3A0512ES

```
Applied to evk1100-prep, thanks.
```

Reply from Håvard Skinnemoen

[PATCH v2 8/9] AVR32: CPU support for AT32UC3A0xxx CPUs

Regarding the reset procedure in the patch:

```
I read this as if you just reset the CPU "internal" stuff. Sorry for
asking stupid questions, I don't know this architecture at all, but:
Will external chips be reset this way, too? Or how do you make sure
that external peripherals get properly reset?
```

Reply from Wolfgang Denk

```
As most of the needed functionality is embedded in the
microcontroller, there are very few external peripherals
used by U-Boot. Apart from external memory, and oscillator,
and level-shifters for the serial-port, there is only the
Ethernet PHY, and that one shouldn't need a reset.
```

Our reply

```
Famous last words. What if exactly the PHY is stuck and needs a reset
    ?

Hmmm... "apart from external memory" ... does external memory also
include NORflash? Eventually the NOR flash you are booting from?
Assume the NOR flash is in query mode when you reset the board - how
does it get reset, then?
```

Reply from Wolfgang Denk

```
The only reset we can do on the PHY is a software reset, by sending a
reset command over the (R)MII bus, and I don't believe that the
    generic
chip code is the place to do that. If it should be done, I believe it
should be done by the macb-driver after the reset. This would allow
    it
to recover even if the microcontroller wasn't reset by the
reset-command, but for example by a watchdog timer.

External memory in this case would be SRAM or SDRAM.
```

Our reply

```
On other chips, it also covers the NOR flash you're booting from. So
    I
suppose we should look into this...maybe we need some sort of "
    notifier
chain" thing to give other drivers a chance to reset their
peripherals...
```

Reply from Håvard Skinnemoen

Comment regarding use of "magic hardcoded constants":

```
It would be nice if you used readable names instead of all these
magic hardcoded constants.
```

Comment from Wolfgang Denk

Denk also pointed out that we should use of structs instead of offsets. We used lists with offsets instead of C-structs because that is how it was done in the existing code we used as a basis for our work, and rewriting this code was not prioritized. The above comment about unreadable constants triggered a discussion about how this code should be written. Several e-mails are omitted here. For the full story, see the mailing list archive available at the URL listed at the beginning of this section. The discussion ended with the following comments from us and Skinnemoen.

```
But in this case, this is code which should never be changed without
looking at the datasheet, and probably schematics for the board in
question.
```

Our comment

```
Exactly. At some point, you need code which encapsulates the
definitions in the data sheet, and that's the whole purpose of these
functions.
```

Comment from Håvard Skinnemoen

[PATCH v2 9/9] AVR32: Board support for ATEVK1100

Wolfgang Denk commented on the presence of some code that was commented out in this patch. He found the code useless and wanted us to remove it. The code in question was the optimized memory timings that we never managed to fully optimize (described in section 3.8), but left in the code.

He also commented that we had set a configuration option that deactivates certain scripting functionality in U-Boot when it is compiled for the EVK1100 board. That option came from the code for the NGW100 board that we had used as a basis for our own configuration. On Denk's request, we chose to remove both the pieces of code mentioned above.

4.5.2 Linux

As mentioned in section 3.11.7, we submitted patches to the avr32linux.org's kernel mailing list. We received some feedback on these patches, mostly from Håvard Skinnemoen. In this section, we summarize the feedback we got on the submitted patches. Comments are included where appropriate.

The patches can be found in appendix D.

General approval

Håvard Skinnemoen gave comments like "Looks reasonable' to the following patches:

- [PATCH 01/29] macb: limit to 10 Mbit/s if the clock is too slow to handle 100 Mbit/s
- [PATCH 04/29] AVR32: use task_pt_regs in copy_thread.
- [PATCH 05/29] AVR32: FDPIC ELF support.
- [PATCH 06/29] AVR32: Introduce AVR32_CACHE and AVR32_UNALIGNED Kconfig options
- [PATCH 07/29] AVR32: mm/tlb.c should only be enabled with CONFIG_MMU.
- [PATCH 08/29] AVR32: mm/fault for !CONFIG_MMU.
- [PATCH 10/29] AVR32: MMU dummy functions for chips without MMU.
- [PATCH 11/29] AVR32: mm_context_t for !CONFIG_MMU
- [PATCH 13/29] AVR32: copy_user for chips that cannot do unaligned memory access.
- [PATCH 14/29] AVR32: csum_partial: Support chips that cannot do unaligned memory accesses.
- [PATCH 16/29] AVR32: memcpy implementation for chips that cannot do unaligned memory accesses.

Håvard wanted signoffs for patches that can be sent upstream.

[PATCH 02/29] AVR32: Don't clear registers when starting a new thread

From the our patch description:

```
Not certain about this patch, but we can't clear the registers here,
since the FDPIC ELF loader stores a pointer to the process' load map
in a register before this function is called.
```

Our patch description

```
Right.

Do you know how other architectures do this?
I'm a bit concerned about leaking information from one process to
    another
if we don't zero out the registers...
```

Håvard's comment

```
As far as we understand does neither x86, frv, SuperH 32, blackfin
    and
several other architectures do it in start_thread. A quick survey
    shows
that ARM and PowerPC are the only architectures who clear the
    registers
in start_thread

X86 and several other architectures clears the registers from an an
architecture dependent hook in the elf loader, ELF_PLAT_INIT, which
is called right before start_thread.
```

Our reply

Patch [PATCH 03/29] AVR32: split paging_init into mmu init, free memory init and exceptions init

```
You still export the zero page when !CONFIG_MMU, but you only
initialize it when CONFIG_MMU is set.
Is that a good idea?
```

Håvard's comment

```
When looking at this again, it turns out that the zero-page wasn't
    used
strictly for MMU-systems. We had assumed that it was only used when
    the
kernel needed to map a page full of zeros somewhere.

It turns out that it is also used by fs/direct-io.c:760. Therefore,
the zero page still needs to exist for MMU-less systems.
```

Our comment

```
But then it really should be initialized, no?
```

Håvard's reply

```
Yes, that was what we meant.
```

Our reply

[PATCH 09/29] AVR32: ioremap and iounmap for !CONFIG_MMU

```
Would probably be more efficient to do this inline.
But I can't see any serious problems with this code,
so it's fine with me.
```

Comment from Håvard

[PATCH 12/29] AVR32: Add cache-function stubs for chips without cache

```
Would be better to do this inline, I think.
But let's worry about optimization later.
```

Comment from Håvard

Regarding a copy_to_user_page stub:

```
Hmm...don't you need to do any copying at all here?
```

Comment from Håvard

```
Oops, seems we became a little carried away here, and missed the
    memcpy.

Regarding making these inline - most of them could be changed, and we
would agree that this would make the code simpler.

Btw.: There are a lot of static inline functions in
include/asm/cacheflush.h that are only called from mm/cache.c
```

Our comment

[PATCH 15/29] AVR32: avoid unaligned access in uaccess.h

```
The patch fixes __get_user_check by calling copy_from_user if the
pointer is unaligned. Note that there are three more macros that
    needs
to be changed: get_user_nocheck, put_user_check and put_user_nocheck.

This patch really needs a better solution that doesn't involve
    calling
copy_from_user or copy_to_user.
```

Patch description

```
I'm sort of wondering if this is really needed. AP7000 doesn't
    support
unaligned 16-bit access, and we don't do anything to avoid that. And
the worst thing that can happen is that some system calls may return
-EFAULT if user space passes a badly aligned pointer.
```

Håvard's comment

```
This patch was added because we hit an exception during the illegal
opcode handler. That function executes the following code:

  pc = (void __user *)instruction_pointer(regs);
  if (get_user(insn, (u32 __user *)pc))
          goto invalid_area;

If get_user isn't changed then this function should be changed.

Also: unaligned accesses in kernel mode doesn't cause an -EFAULT, but
instead an Oops. If the kernel is going to cause unaligned exceptions
    ,
I assume that this should be changed.
```

Our comment

```
Right...I guess that function should be changed. But it _should_ be
able to handle it gracefully in any case...
```

Håvard's reply - about the illegal opcode handler

```
Ah...that doesn't sound good. Looks like do_address_exception() doesn
    't
walk the fixup tables before crashing...that should probably be fixed
    .

Could you give the (untested) patch below a try?

diff --git a/arch/avr32/kernel/traps.c b/arch/avr32/kernel/traps.c
index d547c8d..69e9218 100644
--- a/arch/avr32/kernel/traps.c
+++ b/arch/avr32/kernel/traps.c
@@ -75,8 +75,15 @@ void _exception(long signr, struct pt_regs *regs,
    int code,
 {
         siginfo_t info;

-        if (!user_mode(regs))
+        if (!user_mode(regs)) {
+                /* Are we prepared to handle this kernel fault? */
+                fixup = search_exception_tables(regs->pc);
```

```
+               if (fixup) {
+                       regs->pc = fixup->fixup;
+                       return;
+               }
                die("Unhandled exception in kernel mode", regs, signr
                    );
+       }

        memset(&info, 0, sizeof(info));
        info.si_signo = signr;
```

Håvard's reply - about unaligned access not causing -EFAULT

```
Yes, it solves the problem.
Btw; we had to declare the fixup variable also.
```

Our reply

```
Ah yes...I did actually fix that, but I forgot to regenerate the diff
    .

The result should look something like the below.
```

Håvard's reply

[PATCH 17/29] AVR32: Mark AVR32B specific assumptions with CONFIG_ SUBARCH_AVR32B in strnlen

```
Please include a short description about which assumptions you're
talking about and why they're specific to AVR32B.
```

Comment from Håvard

```
The problem is that this code assumes that the address space is split
into two 2GB parts, with the lower half belonging to user space. This
assumption does not hold for AVR32A, where almost all memory is
    located
in the upper half of the address space, and there is no clear
separation between kernel space and user space memory areas.
```

Our comment

[PATCH 18/29] AVR32: mm/dma-coherent.c - ifdef AVR32B specific code

```
Actually, the whole thing should be a no-op on devices with no cache,
since there's no need to synchronize anything.
```

Comment from Håvard

```
That is true, but in this case we focused on the code that was AVR32B
specific. One could conceivably have an AVR32A microcontroller with
caches?

I assume that if the cache changes above were moved to inline
    functions
in a header file, this function would compile down to a no-op.
```

Our comment

Several patches got comments like:

```
I think this should depend on CONFIG_MMU, not AVR32B.
```

Comment from Håvard Skinnemoen

This applies for the patches listed below.

- [PATCH 18/29] AVR32: mm/dma-coherent.c - ifdef AVR32B specific code.
- [PATCH 19/29] AVR32: Disable ret_if_privileged macro for !CONFIG_SUBARCH_AVR32B.
- [PATCH 21/29] AVR32: AVR32A address space support.
- [PATCH 22/29] AVR32: Change maximum task size for AVR32A
- [PATCH 23/29] AVR32: Fix uaccess ___range_ok macro for AVR32A.

```
The main criteria we did for deciding whether something should depend
on AVR32B or if it should depend on MMU, was whether it depends on
AVR32B memory layout, or whether it depends on an MMU being present.
```

Our comment

```
But the virtual memory layout does not depend on the sub-architecture
(apart from the entry point, which makes the two somewhat related),
    it
depends on whether or not the chip has an MMU.

If the chip does not have an MMU, all virtual addresses are mapped 1:
    1
to physical addresses. If the mapping isn't 1:1, there must be
something in the chip doing the mapping, i.e. an MMU. This is
    confirmed
by the fact that the segmented memory model is defined in the MMU
chapter in the architecture manual.

So there's really no such thing as an AVR32B memory layout -- the
```

```
memory layout depends entirely on whether or not an MMU is present.

As for caches, I think adding caches without also adding an MMU would
be problematic since the caching properties of a given address is
determined by the MMU. So if you don't have an MMU, you won't be able
to bypass the cache for certain parts of the memory, which makes it
difficult to do DMA.

Sure, it might be possible to introduce some other mechanism for
specifying caching properties, but the current architecture document
does not specify any such mechanism apart from the MMU.
```

Håvard's reply

After Håvards comment, we realized that some of our decisions on which configuration flags we used was based on a misunderstanding. Earlier we had the misconception that AVR32A implied that an MMU was not present, and AVR32B meant that an MMU was present.

The changes we did were still correct, but the build criteria were wrong in many places. As mentioned in Håvard's comments, some code segments should be updated to depend on the `CONFIG_MMU` option and not the sub-architecture as our patches do.

[PATCH 20/29] AVR32: AVR32A support in Kconfig

```
Ok. I was thinking this could have been merged with some of the other
AVR32A patches, but then again, this makes it easier to reorder the
patches, so it's fine.
```

Comment from Håvard Skinnemoen

When the implementation were split into patches, we tried to keep this in mind and rather split into too many rather than too few. By doing this, it should hopefully be easier for other developers to pick up our work and continue development.

[PATCH 21/29] AVR32: AVR32A address space support

```
Haven't had a chance to have a good look over but:
On Fri, 2009-05-15 at 14:39 +0200, Gunnar Rangoy wrote:
> +#elif CONFIG_SUBARCH_AVR32B
#elif defined (CONFIG_SUBARCH_AVR32B)
??
```

Comment from Ben Nizette

```
Oops, it should indeed use defined(...). It will still work as long
    as
the only sub-architectures are AVR32A and AVR32B, which is why we
    missed it.
```

Our comment

[PATCH 24/29] AVR32: Support for AVR32A (entry-avr32a.c)

```
Ok, this part really does depend on AVR32A, so that part is fine.
Unfortunately, I haven't got the time to review this or the remaining
patches today, so I'll have to continue some other day (probably next
week).
```

Håvard Skinnemoen

4.5.3 Toolchain

Binutils support for FDPIC ELF on AVR32 UC3

```
This patch adds support for statically linked FDPIC ELF targets on
AVR32. It mostly works, but there is a lack of error checking on
    input
file types, which means that if the linker is invoked incorrectly,
it will fail in strange ways.

For example, if one fails to specify -I elf32-avr32fdpic to
strip/objcopy, it will pretend that the file is a normal
elf32-avr32 file, and "ruin" the PT_GNU_STACK program header.

Some functions are (almost) direct copies from elf32-bfin.c and
elf32-frv.c, which are two architectures with FDPIC support. The code
for creating the .rofixup-section is however mostly new.
```

Patch description

```
Without this error checking, it will be difficult to accept the patch
as-is. We can't in good faith expect our users accept that the linker
will fail "in strange ways" because of an incorrect invocation.
It needs to fail gracefully and in a known way.
```

Comment from Eric Weddington (Atmel)

uClibc: Some support for FDPIC ELF for AVR32

```
This patch enables uClibc to be linked statically into a FDPIC ELF
binary on AVR32. It doesn't update the parts necessary for dynamic
linking.

There are also a few simple changes to memcmp, memcpy and memmove,
```

```
which makes them work on the UC3 (which cannot access unaligned
memory.)
```

Patch description

```
If the change to the mem* functions have nothing to do with support
for FDPIC, then it is preferrable if the patches are separated.
The idea is that a patch file should have a single purpose only and
not to mix together changes with different purposes.
```

Comment from Eric Weddington (Atmel)

```
I will commit the uClibc stuff, and it will be broken down into
separate changes. Patches will go through review on the uClibc
list + Paul before committing.
```

Hans-Christian Egtvedt (Atmel)

Chapter 5

Conclusion

During this project we have created a modified version of Linux, capable of running on a microcontroller of the UC3A family. A toolchain has been extended with the capability of generating executables suitable for this platform. Our version of Linux is capable of loading and running these executables. Previously submitted patches for the U-Boot loader have been improved, significanly revised, and re-submitted.

Because custom hardware had to be used, the usefulness of the product of our work is currently somewhat limited. However, with newer chips without the SDRAM bug, it should be possible to run Linux on the EVK1100 without hardware modifications. With some software modifications, it should also be possible to use our work with other closely related microcontrollers in the AVR32 family.

Patches for the majority of all software modifications done in this project have been submitted to the appropriate maintainers. By publishing our work, we have significantly contributed to increase the useful assortment of software for the UC3A microcontroller family. If Atmel wants to, they can adopt the patches and finalize them to make sure that they ultimately become part of their respective official software distributions.

By undertaking this project, we have gained valuable knowledge about embedded development, the Linux kernel, the GNU Toolchain, and open source development in general. It has also been a valuable experience to communicate with other people in the open source communities.

Chapter 6

Future work

This chapter describes the tasks that currently remain undone. The three first sections in this chapter describes remaining work in U-Boot, Linux and the toolchain. The last section discusses the possibility of running programs created for the AVR32A on AVR32B chips and vice versa.

6.1 U-Boot

In many setups it would be more suitable to load the kernel from SD card or flash, because then it would not need to depend on external systems when booting.

There are a few changes that still is not submitted to the U-Boot mailing list. This mostly regards the MACB driver but it also includes some cleanup of unnecessary code. These could be included in an updated patch series, but this was not highly prioritized.

6.2 Linux

Ideally, Linux should support all the hardware in the UC3A controller and on the EVK1100. In this section we outline the most essential features that we would have tried to implement if we had the time and hardware available.

6.2.1 PDCA support

As described in section 3.6.9, support for the PDCA was never completed. Support for the PDCA would be very useful since it is a great feature for communicating with peripherals. The restructured code for the PDC should be obtained from Atmel and adapted for the UC3A0512.

6.2.2 SPI support

The proper way to use the SPI is in combination with the PDCA. Since the PDCA support never was completed, neither was the SPI support. On the EVK1100, the microcontroller is connected to several SPI devices. Therefore, Linux support for the SPI controller would be very useful.

6.2.3 MPU support

Linux does not currently support any use of the MPU, and no attempt has been made to implement this. Without the MPU enabled, any process can read and write to any memory location, and potentially obtain all information about, sabotage or modify the kernel or any processes. The MPU is dysfunctional in our chip, and it would be very hard to implement and test the software for it.

6.2.4 Support for on-chip devices

Linux support would also be desirable for the following on-chip features of the UC3A:

- USB interface
- Audio Bitstream DAC
- Synchronous Serial Controller
- Analog-to-Digital Converter

None of these features have been considered in the implementation phase of this project.

6.2.5 Memory copy optimization

The implementation of memory copying routines in Linux is not optimized for any unaligned or halfword copying. A good way to achieve this functionality would be to extract the already existing and optimized copying routines in newlib. Newlib is a standard C library implementation for embedded systems and does not require any operating system like uClibc does. For more information about newlib, see the newlib website[1].

6.2.6 Debug support

Support for debugging applications with a software debugger under Linux is not completed. We have made changes to the entry point, so debugging events should be handled. However, there are some code in `arch/avr32/kernel/ptrace.c` that is not changed for the AVR32A architecture.

The relevant piece of code is:

[1] http://sourceware.org/newlib/

```
1 ti->rar_saved = sysreg_read(RAR_EX);
2 ti->rsr_saved = sysreg_read(RSR_EX);
3 sysreg_write(RAR_EX, trampoline_addr);
4 sysreg_write(RSR_EX, (MODE_EXCEPTION | SR_EM | SR_GM));
```

This code sets up something called a "debug trampoline", to handle the case where a user space program single steps into an exception. In such cases, the exception should be executed at full speed and single stepping should resume after the exception. The code above changes the return address of the exception, so that that returns to a debug "trampoline" instead of the real return address. This trampoline will then reconfigure the debug system, so that single stepping can be resumed.

The problem with the code is that it changes two system registers that are unavailable on the AVR32A architecture. Instead of saving the return address and status register in dedicated system registers, the AVR32A architecture saves them on the stack. The equivalent of changing the two system registers would be to change the two registers as they are saved on the stack.

We made a design decision to try to reuse the registers saved automatically by the processor for the `pt_regs` structure. Thus, if we change the return address and status register on the stack, we will also change the return address and status register the exception handler would see. This would be troublesome, since the status register and return address is used to determine how the exception is handled.

A possible work around would be to insert a new stack frame when single stepping into an exception. The first stack frame would contain the correct `pt_regs` structure, while the next stack frame would contain what is needed to return to the debug "trampoline".

6.2.7 FDPIC ELF support for systems with an MMU

As mentioned in 3.6.16, we only added support for FDPIC ELF for systems with an MMU. It would be useful to also support the FDPIC ELF format in systems with MMU support. This would allow development of FDPIC ELF applications and the FDPIC ELF toolchain on platforms with an MMU.

To do this, one needs to create a structure of the `mm_context_t` used by MMU systems, and add the fields required by the FDPIC ELF loader to this structure.

6.3 Toolchain

The toolchain is not yet completed – it lacks support for dynamic linking, and it needs some error checking. In this section we will outline the remaining tasks for the toolchain.

6.3.1 Dynamic linking

The toolchain is currently only able to produce statically linked binaries. It should be able to support creating shared libraries and dynamically linked executables. At the very least, this requires changes to the linker, but it might be advantageous to also change the assembler and GCC.

The linker needs to be changed to handle a new relocation type for function calls. As mentioned in section 2.10.3, function calls across two different modules need to change the current GOT pointer to the new modules GOT pointer. The previous GOT pointer needs to be restored when the function call returns. The linker must therefore support this type of function call.

When saving the previous GOT pointer, it can also be advantageous to save it to one of the registers which is preserved across function calls. That would require changes to GCC and to the assembler. The assembler would need a new pseudo-instruction for function calls. It currently has a pseudo-instruction for function calls, which the linker replaces with the correct method for calling the function. The new pseudo-instruction should take in both the destination of the function call and a register which can be used to hold the previous GOT pointer.

After the new pseudo-instruction is added to the assembler, GCC would need to be changed to use it. GCC would need to select a suitable register and insert the new call instruction with that register as a parameter.

6.3.2 Error handling

The current changes to the linker doesn't properly check that all input files are FDPIC ELF files when being executed with the `-mavr32linuxfdpic` flag. It is this flag that tells the linker that it is processing FDPIC ELF files. This leads to errors, since it will not initialize everything correctly in those cases. What needs to be done is to check that every input file is FDPIC ELF files when the linker is executed with the `-mavr32linuxfdpic` flag. If it isn't executed with the `-mavr32linuxfdpic` flag, it should check that none of the input files are FDPIC ELF files.

6.4 AVR32B series compatibility

Since AVR32A and AVR32B are both implementations of the AVR32 architecture, it should, in theory, be possible to run the same binary Linux applications on both sub-architectures. A couple of requirements have to be fulfilled, though. The binary must only use instructions available in both sub-architectures, and can not use unaligned memory accesses. Note that different revisions of the AVR32 architecture exist, and the

instruction sets differ slightly. In the future, if support for dynamically linked libraries is implemented for both sub-architectures, unaligned access could be outsourced to the C library, thus eliminating the alignment issue in user space applications. To be able to work on all AVR32 variations, the binary must be compiled as a FDPIC ELF file, and FDPIC ELF files must be supported also on AVR32 systems with an MMU. See also section 6.2.7.

Chapter 7

Bibliography

[1] Avr32 linux kernel wiki. *http://avr32linux.org/twiki/bin/view/Main/LinuxKernel*, 2008.

[2] Erik Andersen. The official busybox website. *http://www.busybox.net*, 2009.

[3] Rob Arnold. Why do i need a resistor with an led? *http://led.linear1.org/why-do-i-need-a-resistor-with-an-led/*, 2006.

[4] Atmel. avr32 gcc. *http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=4118*.

[5] Atmel. AVR32 Architecture Manual. *http://www.atmel.com/dyn/resources/prod_documents/doc32000.pdf*, 2007.

[6] Atmel. AVR32 UC3 32-bit Flash Microcontrollers. *http://www.atmel.com/dyn/resources/prod_documents/doc7919.pdf*, 2007.

[7] Atmel. AT32AP7000 Datasheet. *http://www.atmel.com/dyn/resources/prod_documents/doc32003.pdf*, 2008.

[8] Atmel. AT32UC3A Series Datasheet, Revision F. *http://www.atmel.com/dyn/resources/prod_documents/doc32058.pdf*, 2008.

[9] Atmel. AVR32 UC3A Product Card. *http://www.atmel.com/dyn/products/product_card.asp?part_id=4117*, 2008.

[10] Eric J. Braude. *Software Engineering - An Object-Oriented Perspective*. John Wilsey & Sons, INC, 2000.

[11] Free Software Foundation. Gcc development mission statement. *http://gcc.gnu.org/gccmission.html*, 1999.

[12] Free Software Foundation. Gnu binutils. *http://www.gnu.org/software/binutils/*, 2007.

[13] CodeSourcery Inc. Joseph Myers. Draft sh uclinux fdpic abi. http://gcc.gnu.org/ml/gcc/2008-02/msg00619.html, 2008.

[14] Kernel.org. Linux Kernel Archives. http://kernel.org/, 2008.

[15] Amol Lad, Sriram Neelakandan, and Pichai Raghavan. *Embedded Linux system design and development*. CRC Press, 2005.

[16] Evan Leibovitch. The 86open project. http://www.telly.org/86open/, 1999.

[17] Kernel maintainers. Linux Kernel Documentation. http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=tree;f=Documentation.

[18] Arcturus Networks. What is uClinux? http://uclinux.org/description/, 2008.

[19] David A. Patterson and John L. Hennessy. *Computer Orgainzation and Design*. Morgan Kaufmann, 3 edition, 2007.

[20] Craig Peacock. uclinux - bflt binary flat format. *http://www.beyondlogic.org/uClinux/bflt.htm*, 2005.

[21] Inc The Santa Cruz Operation. System v, application binary interface. http://www.caldera.com/developers/devspecs/gabi41.pdf, 1997.

[22] uCdot.org. uclinux merged into main line linux kernel sources. http://www.ucdot.org/article.pl?sid=02/11/05/0324207.

[23] uClibc.org. About uclibc. http://www.uclibc.org/, 2008.

[24] usb.org. Introduction to usb on-the-go. http://www.usb.org/developers/onthego/USB_OTG_Intro.pdf, 2003.

[25] Inc VDC Research Group. THE EMBEDDED SOFTWARE MARKETING INTELLIGENCE PROGRAM: 2008 Service YearTrack 1: Operating Systems Used in Embedded Systems, Volume 1: Linux. http://www.vdcresearch.com/PurchasedDownloadFile.asp?type=executivebrief&id=2283, 2008.

Appendix A

Acronyms

ABI Application Binary Interface

BFD Binary Format Descriptor

CPU Central Processor Unit

CVS Concurrent Versions System

DAC Digital to Analog Converter

DHCP Dynamic Host Configuration Protocol

DMA Direct Memory Access

EBI External Bus Interface

EEPROM Electrically Erasable Programmable Read-Only Memory

ELF Executable and Linkable Format

EPROM Erasable Programmable Read-Only Memory

EEPROM Electrically Erasable Programmable Read-Only Memory

EPROM Erasable Programmable Read-Only Memory

FDPIC Function Descriptor Position Independent Code

FSF Free Software Foundation

GCC GNU Compiler Collection

GDB GNU Debugger

GNU GNU's Not Unix

GPIO General Purpose Input/Output

GPL General Public License

GOT Global Offset Table

HSB High Speed Bus

IO Input/Output

IP Internet Protocol

IDE Integrated Drive Electronics

JTAG Joint Test Action Group

LCD Liquid Crystal Display

LED Light Emitting Diode

MAC Media Access Controller

MCI MultiMedia Card Interface

MII Media Independent Interface

MMU Memory Management Unit

MPU Memory Protection Unit

NFS Network File System

NOP No-Operation

PDC Peripheral DMA Controller

PDCA Peripheral DMA Controller

PIC Position Independent Code

PIO Parallel Input/Output

PLC Programmable Logic Controller

PLT Procedure Linkage Table

POSIX Portable Operating System Interface for Unix

PROM Programmable Read-Only Memory

RAM Random Access Memory

RMII Reduced Media Independent Interface

SD Secure Digital

SDRAM Synchronous Dynamic Random Access Memory

SMC Static Memory Controller

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

SUS Single UNIX Specification

TCP Transmission Control Protocol

TFTP Trivial File Transfer Protocol

TLB Translation Lookaside Buffer

UDP User Datagram Protocol

URL Uniform Resource Locator

USB Universal Serial Bus

Appendix B

U-Boot patch cleanup

B.1 Network limiting reorganization

```
diff --git a/drivers/net/macb.c b/drivers/net/macb.c
index 561669b..31a4fbe 100644
--- a/drivers/net/macb.c
+++ b/drivers/net/macb.c
@@ -296,28 +296,17 @@ static void macb_phy_reset(struct macb_device *macb)
        struct eth_device *netdev = &macb->netdev;
        int i;
        u16 status, adv;
-       int rmii_mode;
-       unsigned min_hz;
-
-#ifdef CONFIG_RMII
-       rmii_mode = 1;
-       min_hz = 50000000;
-#else
-       rmii_mode = 0;
-       min_hz = 25000000;
-#endif

        adv = ADVERTISE_CSMA | ADVERTISE_ALL ;

-       if (get_hsb_clk_rate() < min_hz) {
-               printf("%s: HSB clock < %u MHz in %s mode - "
-                      "disabling 100mbit.\n", netdev->name, min_hz / 1000000,
-                      (rmii_mode ? "RMII" : "MII"));
+#ifdef CONFIG_MACB_FORCE10M
+       printf("%s: 100Mbps is not supported on this board - forcing 10Mbps.\n",
+              netdev->name);

-               adv &= ~ADVERTISE_100FULL;
-               adv &= ~ADVERTISE_100HALF;
-               adv &= ~ADVERTISE_100BASE4;
-       }
+       adv &= ~ADVERTISE_100FULL;
+       adv &= ~ADVERTISE_100HALF;
+       adv &= ~ADVERTISE_100BASE4;
+#endif

        macb_mdio_write(macb, MII_ADVERTISE, adv);
        printf("%s: Starting autonegotiation...\n", netdev->name);
@@ -345,7 +334,7 @@ static int macb_phy_find(struct macb_device *macb)

        /* Search for PHY... */
        for (i = 0; i < 32; i++) {
-               macb->phy_addr=i;
+               macb->phy_addr = i;
                phy_id = macb_mdio_read(macb, MII_PHYSID1);
                if (phy_id != 0xffff) {
                        printf("%s: PHY present at %d\n", macb->netdev.name, i);
```

B.2 Add board to lists

```
diff --git a/MAINTAINERS b/MAINTAINERS
index 9c0d6bf..d83b580 100644
--- a/MAINTAINERS
+++ b/MAINTAINERS
@@ -747,6 +747,7 @@ Haavard Skinnemoen <haavard.skinnemoen@atmel.com>
         ATSTK1004       AT32AP7002
         ATSTK1006       AT32AP7000
         ATNGW100        AT32AP7000
+        ATEVK1100       AT32UC3A0512

 ##############################################################################
 # SuperH Systems:                                                            #
diff --git a/MAKEALL b/MAKEALL
index 9ccb9ac..cd33214 100755
--- a/MAKEALL
+++ b/MAKEALL
@@ -720,6 +720,7 @@ LIST_coldfire="                                 \
 ##############################################################################

 LIST_avr32="            \
+        atevk1100       \
         atstk1002       \
         atstk1003       \
         atstk1004       \
```

B.3 Precedence safety fix

```
diff --git a/cpu/at32uc/smc.h b/cpu/at32uc/smc.h
index ea4d399..ae765ec 100644
--- a/cpu/at32uc/smc.h
+++ b/cpu/at32uc/smc.h
@@ -8,10 +8,10 @@
 #include <asm/io.h>

 /* SMC register offsets */
-#define SMC_SETUP(x)                    0x0000+(x)*0x10
-#define SMC_PULSE(x)                    0x0004+(x)*0x10
-#define SMC_CYCLE(x)                    0x0008+(x)*0x10
-#define SMC_MODE(x)                     0x000c+(x)*0x10
+#define SMC_SETUP(x)                    (0x0000+(x)*0x10)
+#define SMC_PULSE(x)                    (0x0004+(x)*0x10)
+#define SMC_CYCLE(x)                    (0x0008+(x)*0x10)
+#define SMC_MODE(x)                     (0x000c+(x)*0x10)

 /* Bitfields in SETUP0..3 */
 #define SMC_NWE_SETUP_OFFSET            0
```

B.4 Board configuration

```
diff --git a/include/configs/atevk1100.h b/include/configs/atevk1100.h
index 2a9d91b..ad134f8 100644
--- a/include/configs/atevk1100.h
+++ b/include/configs/atevk1100.h
@@ -72,7 +72,8 @@
   * Select the operating range for the PLL.
   * PLLOPT[0]: Select the VCO frequency range.
   * PLLOPT[1]: Enable the extra output divider.
- * PLLOPT[2]: Disable the Wide-Bandwidth mode (Wide-Bandwidth mode allows a faster startup time and out-of-lock time).
+ * PLLOPT[2]: Disable the Wide-Bandwidth mode (Wide-Bandwidth mode allows a
+ *            faster startup time and out-of-lock time).
   *
   * We want to run the cpu at 66 MHz, and the fVCO of the PLL at 132 MHz.
   */
@@ -93,7 +94,7 @@

 #define CONFIG_STACKSIZE                (2048)

-#define CONFIG_BAUDRATE                         9600
+#define CONFIG_BAUDRATE                         115200
 #define CONFIG_BOOTARGS                                                 \
```

```
        "console=ttyS0 ip=dhcp root=/dev/nfs rootwait=1"

@@ -143,6 +144,11 @@
 /* Ethernet - RMII mode */
 #define CONFIG_MACB                     1
 #define CONFIG_RMII                     1
+/*
+ * 100Mbps requires a CPU clock of at least 50MHz for RMII mode, and 25MHz for
+ * MII mode. Set CONFIG_MACB_FORCE10M flag if clock is too slow for 100Mbit.
+ */
+#define CONFIG_MACB_FORCE10M            1

 #define CONFIG_ATMEL_USART              1
 #define CONFIG_ATMEL_SPI                1
@@ -156,7 +162,7 @@

 #define CONFIG_NR_DRAM_BANKS            1

-/* Internal flash on the microcontroller (TODO?) (512kB)*/
+/* Internal flash on the microcontroller (512kB)*/
 #define CFG_FLASH_BASE                  0x80000000
 #define CFG_FLASH_SIZE                  0x80000
 #define CFG_MAX_FLASH_BANKS             1
@@ -171,14 +177,15 @@

 #define CONFIG_ENV_IS_IN_FLASH          1
 #define CONFIG_ENV_SIZE                         65536
-#define CONFIG_ENV_ADDR                         (CFG_FLASH_BASE + CFG_FLASH_SIZE - CONFIG_ENV_SIZE)
+#define CONFIG_ENV_ADDR                         (CFG_FLASH_BASE + CFG_FLASH_SIZE - \
+                                        CONFIG_ENV_SIZE)

 #define CFG_INIT_SP_ADDR                (CFG_INTRAM_BASE + CFG_INTRAM_SIZE)

 #define CFG_MALLOC_LEN                  (256*1024)
 #define CFG_DMA_ALLOC_LEN               (16384)

-/* Allow 3MB(TODO:update) for the kernel run-time image */
+/* Allow 2.5MB for the kernel run-time image */
 #define CFG_LOAD_ADDR                   (CFG_SDRAM_BASE + 0x00270000)
 #define CFG_BOOTPARAMS_LEN              (16 * 1024)
```

B.5 Keeping lists sorted

```
diff --git a/Makefile b/Makefile
index bfaa625..d9fbc6e 100644
--- a/Makefile
+++ b/Makefile
@@ -3047,6 +3047,9 @@ $(BFIN_BOARDS):
 # AVR32
 #========================================================================

+atevk1100_config       :       unconfig
+       @$(MKCONFIG) $(@:_config=) avr32 at32uc atevk1100 atmel at32uc3a0xxx
+
 atngw100_config        :       unconfig
        @$(MKCONFIG) $(@:_config=) avr32 at32ap atngw100 atmel at32ap700x

@@ -3071,9 +3074,6 @@ hammerhead_config :       unconfig
 mimc200_config         :       unconfig
        @$(MKCONFIG) $(@:_config=) avr32 at32ap mimc200 mimc at32ap700x

-atevk1100_config       :       unconfig
-       @$(MKCONFIG) $(@:_config=) avr32 at32uc atevk1100 atmel at32uc3a0xxx
-
 #========================================================================
 # SH3 (SuperH)
 #========================================================================
```

B.6 Removal of TODOs

```
diff --git a/cpu/at32uc/at32uc3a0xxx/sm.h b/cpu/at32uc/at32uc3a0xxx/sm.h
index d232f91..17bff39 100644
--- a/cpu/at32uc/at32uc3a0xxx/sm.h
+++ b/cpu/at32uc/at32uc3a0xxx/sm.h
```

```
@@ -30,7 +30,6 @@
 #define SM_PM_VREGCR                            (SM_PM_REGS_OFFSET + 0x00c8)
 #define SM_PM_BOD                               (SM_PM_REGS_OFFSET + 0x00d0)
 #define SM_PM_RCAUSE                            (SM_PM_REGS_OFFSET + 0x0140)
-#define SM_RC_RCAUSE                            SM_PM_RCAUSE /* TODO: remove */
 /* RTC starts at 0xFFFF0D00 */
 #define SM_RTC_REGS_OFFSET                      0x0d00
 #define SM_RTC_CTRL                             (SM_RTC_REGS_OFFSET + 0x0000)
@@ -45,25 +44,24 @@
 #define SM_WDT_REGS_OFFSET                      0x0d30
 #define SM_WDT_CTRL                             (SM_WDT_REGS_OFFSET + 0x0000)
 #define SM_WDT_CLR                              (SM_WDT_REGS_OFFSET + 0x0004)
-#define SM_WDT_EXT                              (SM_WDT_REGS_OFFSET + 0x0008) /* TODO: does not exist ?
    */
 /* EIC starts at offset 0xFFFF0D80 */
 /* TODO: change EIM to EIC */
 #define SM_EIC_REGS_OFFSET                      0x0d80
-#define SM_EIM_IER                              (SM_EIC_REGS_OFFSET + 0x0000)
-#define SM_EIM_IDR                              (SM_EIC_REGS_OFFSET + 0x0004)
-#define SM_EIM_IMR                              (SM_EIC_REGS_OFFSET + 0x0008)
-#define SM_EIM_ISR                              (SM_EIC_REGS_OFFSET + 0x000c)
-#define SM_EIM_ICR                              (SM_EIC_REGS_OFFSET + 0x0010)
-#define SM_EIM_MODE                             (SM_EIC_REGS_OFFSET + 0x0014)
-#define SM_EIM_EDGE                             (SM_EIC_REGS_OFFSET + 0x0018)
-#define SM_EIM_LEVEL                            (SM_EIC_REGS_OFFSET + 0x001c)
-#define SM_EIM_FILTER                           (SM_EIC_REGS_OFFSET + 0x0020)
-#define SM_EIM_TEST                             (SM_EIC_REGS_OFFSET + 0x0024)
-#define SM_EIM_ASYNC                            (SM_EIC_REGS_OFFSET + 0x0028)
-#define SM_EIM_SCAN                             (SM_EIC_REGS_OFFSET + 0x002c)
-#define SM_EIM_EN                               (SM_EIC_REGS_OFFSET + 0x0030)
-#define SM_EIM_DIS                              (SM_EIC_REGS_OFFSET + 0x0034)
-#define SM_EIM_CTRL                             (SM_EIC_REGS_OFFSET + 0x0038)
+#define SM_EIC_IER                              (SM_EIC_REGS_OFFSET + 0x0000)
+#define SM_EIC_IDR                              (SM_EIC_REGS_OFFSET + 0x0004)
+#define SM_EIC_IMR                              (SM_EIC_REGS_OFFSET + 0x0008)
+#define SM_EIC_ISR                              (SM_EIC_REGS_OFFSET + 0x000c)
+#define SM_EIC_ICR                              (SM_EIC_REGS_OFFSET + 0x0010)
+#define SM_EIC_MODE                             (SM_EIC_REGS_OFFSET + 0x0014)
+#define SM_EIC_EDGE                             (SM_EIC_REGS_OFFSET + 0x0018)
+#define SM_EIC_LEVEL                            (SM_EIC_REGS_OFFSET + 0x001c)
+#define SM_EIC_FILTER                           (SM_EIC_REGS_OFFSET + 0x0020)
+#define SM_EIC_TEST                             (SM_EIC_REGS_OFFSET + 0x0024)
+#define SM_EIC_ASYNC                            (SM_EIC_REGS_OFFSET + 0x0028)
+#define SM_EIC_SCAN                             (SM_EIC_REGS_OFFSET + 0x002c)
+#define SM_EIC_EN                               (SM_EIC_REGS_OFFSET + 0x0030)
+#define SM_EIC_DIS                              (SM_EIC_REGS_OFFSET + 0x0034)
+#define SM_EIC_CTRL                             (SM_EIC_REGS_OFFSET + 0x0038)

 /* Bitfields used in many registers */
 #define SM_EN_OFFSET                            0
@@ -110,8 +108,6 @@
 #define SM_PLLMUL_SIZE                          4
 #define SM_PLLCOUNT_OFFSET                      24
 #define SM_PLLCOUNT_SIZE                        6
-#define SM_PLLTEST_OFFSET                       31      /* TODO: remove */
-#define SM_PLLTEST_SIZE                                 1       /* TODO: remove */

 /* Bitfields in PM_OSCCTRL0,1 */
 #define SM_MODE_OFFSET                                  0
@@ -119,31 +115,12 @@
 #define SM_STARTUP_OFFSET                       8
 #define SM_STARTUP_SIZE                                 3

-/* Bitfields in PM_VCTRL */
-#define SM_VAUTO_OFFSET                                 0       /* TODO: remove */
-#define SM_VAUTO_SIZE                           1       /* TODO: remove */
-#define SM_PM_VCTRL_VAL_OFFSET                  8       /* TODO: remove */
-#define SM_PM_VCTRL_VAL_SIZE                    7       /* TODO: remove */
-
-/* Bitfields in PM_VMREF */
-#define SM_REFSEL_OFFSET                        0       /* TODO: remove */
-#define SM_REFSEL_SIZE                          4       /* TODO: remove */
-
-/* Bitfields in PM_VMV */
-#define SM_PM_VMV_VAL_OFFSET                    0       /* TODO: remove */
-#define SM_PM_VMV_VAL_SIZE                      8       /* TODO: remove */

 /* Bitfields in PM_IER/IDR/IMR/ISR/ICR, POSCSR */
 #define SM_LOCK0_OFFSET                                 0
 #define SM_LOCK0_SIZE                           1
 #define SM_LOCK1_OFFSET                                 1
 #define SM_LOCK1_SIZE                           1
-#define SM_WAKE_OFFSET                          2       /* TODO: remove */
-#define SM_WAKE_SIZE                            1       /* TODO: remove */
```

```
-#define SM_VOK_OFFSET                   3       /* TODO: remove */
-#define SM_VOK_SIZE                     1       /* TODO: remove */
-#define SM_VMRDY_OFFSET                         4       /* TODO: remove */
-#define SM_VMRDY_SIZE                   1       /* TODO: remove */
 #define SM_CKRDY_OFFSET                         5
 #define SM_CKRDY_SIZE                  1
 #define SM_MSKRDY_OFFSET               6
@@ -180,8 +157,6 @@
 #define SM_WDT_SIZE                    1
 #define SM_JTAG_OFFSET                 4
 #define SM_JTAG_SIZE                   1
-#define SM_SERP_OFFSET                 5       /* TODO: remove */
-#define SM_SERP_SIZE                   1       /* TODO: remove */
 #define SM_CPUERR_OFFSET               7
 #define SM_CPUERR_SIZE                 1
 #define SM_OCDRST_OFFSET               8
```

B.7 Coding style fixes

```
diff --git a/cpu/at32uc/at32uc3a0xxx/clk.c b/cpu/at32uc/at32uc3a0xxx/clk.c
index 7c66b94..7d4f813 100644
--- a/cpu/at32uc/at32uc3a0xxx/clk.c
+++ b/cpu/at32uc/at32uc3a0xxx/clk.c
@@ -43,7 +43,8 @@ void clk_init(void)
        sm_writel(PM_MCCTRL, SM_BIT(OSC0EN));

        /* wait for osc0 */
-       while (!(sm_readl(PM_POSCSR) & SM_BIT(OSC0RDY))) ;
+       while (!(sm_readl(PM_POSCSR) & SM_BIT(OSC0RDY)))
+               ;

        /* run from osc0 */
        sm_writel(PM_MCCTRL, SM_BF(MCSEL, 1) | SM_BIT(OSC0EN));
@@ -59,11 +60,13 @@ void clk_init(void)
                                | SM_BIT(ERRATA)));

        /* Wait for lock */
-       while (!(sm_readl(PM_POSCSR) & SM_BIT(LOCK0))) ;
+       while (!(sm_readl(PM_POSCSR) & SM_BIT(LOCK0)))
+               ;
 #endif

        /* We cannot write the CKSEL register before the ready-signal is set. */
-       while (!(sm_readl(PM_POSCSR) & SM_BIT(CKRDY))) ;
+       while (!(sm_readl(PM_POSCSR) & SM_BIT(CKRDY)))
+               ;

        /* Set up clocks for the CPU and all peripheral buses */
        cksel = 0;
diff --git a/cpu/at32uc/cpu.c b/cpu/at32uc/cpu.c
index 4a95427..d145e1d 100644
--- a/cpu/at32uc/cpu.c
+++ b/cpu/at32uc/cpu.c
@@ -55,7 +55,7 @@ int cpu_init(void)
        sysreg_write(EVBA, (unsigned long)&_evba);
        asm volatile("csrf      %0" : : "i"(SYSREG_EM_OFFSET));

-       if(gclk_init)
+       if (gclk_init)
                gclk_init();

        return 0;
diff --git a/cpu/at32uc/flashc.c b/cpu/at32uc/flashc.c
index e626e1f..2244b2e 100644
--- a/cpu/at32uc/flashc.c
+++ b/cpu/at32uc/flashc.c
@@ -56,7 +56,7 @@ unsigned long flash_init(void)
        /* Currently, all interflash have pages which are 128 words. */
        flash_info[0].sector_count = size / (128*4);

-       for(i=0; i<flash_info[0].sector_count; i++){
+       for (i = 0; i < flash_info[0].sector_count; i++) {
                flash_info[0].start[i] = i*128*4 + CFG_FLASH_BASE;
        }

@@ -73,19 +73,20 @@ void flash_print_info(flash_info_t *info)

 static void flash_wait_ready(void)
 {
```

```
-       while(! flashc_readl(FSR) & FLASHC_BIT(FRDY) );
+       while (!flashc_readl(FSR) & FLASHC_BIT(FRDY))
+               ;
 }

 int flash_erase(flash_info_t *info, int s_first, int s_last)
 {
        int page;

-       for(page=s_first;page<s_last; page++){
+       for (page = s_first; page < s_last; page++) {
                flash_wait_ready();
-               flashc_writel(
-                       FCMD,FLASHC_BF(CMD, FLASHC_EP)
-                       |FLASHC_BF(PAGEN, page)
-                       |FLASHC_BF(KEY, 0xa5));
+               flashc_writel(FCMD,
+                             FLASHC_BF(CMD, FLASHC_EP) |
+                             FLASHC_BF(PAGEN, page) |
+                             FLASHC_BF(KEY, 0xa5));
        }
        return ERR_OK;
 }
@@ -105,15 +106,15 @@ static void write_flash_page(unsigned int pagen, const u32 *data)

        /* fill page buffer*/
        flash_wait_ready();
-       for(i=0; i<128; i++){
-               dst[i]=data[i];
+       for (i = 0; i < 128; i++) {
+               dst[i] = data[i];
        }

        /* issue write command */
        flashc_writel(FCMD,
-               FLASHC_BF(CMD, FLASHC_WP)|
-               FLASHC_BF(PAGEN, pagen)|
-               FLASHC_BF(KEY, 0xa5));
+                     FLASHC_BF(CMD, FLASHC_WP) |
+                     FLASHC_BF(PAGEN, pagen) |
+                     FLASHC_BF(KEY, 0xa5));
 }

 int write_buff(flash_info_t *info, uchar *src, ulong addr, ulong count)
@@ -134,7 +135,7 @@ int write_buff(flash_info_t *info, uchar *src, ulong addr, ulong count)
        for (i = 0; i < count; i += 128*4) {
                unsigned int pagen;
                pagen = (addr-CFG_FLASH_BASE+i) / (128*4);
-               write_flash_page(pagen, (u32*) (src+i));
+               write_flash_page(pagen, (u32 *) (src+i));
        }


diff --git a/cpu/at32uc/smc.c b/cpu/at32uc/smc.c
index f4bb9fb..74c2947 100644
--- a/cpu/at32uc/smc.c
+++ b/cpu/at32uc/smc.c
@@ -26,13 +26,13 @@ unsigned long sram_init(const struct sram_config *config)

        switch (config->data_bits) {
        case 8:
-               dbw=0;
+               dbw = 0;
                break;
        case 16:
-               dbw=1;
+               dbw = 1;
                break;
        case 32:
-               dbw=2;
+               dbw = 2;
                break;
        default:
                panic("Invalid number of databits for SRAM");
@@ -52,7 +52,7 @@ unsigned long sram_init(const struct sram_config *config)


        smc_writel(config->chip_select, MODE, cfgreg);
-       sram_size= (1<<config->address_bits) * (config->data_bits/8);
+       sram_size = (1<<config->address_bits) * (config->data_bits/8);


        return sram_size;
```

```
diff --git a/include/asm-avr32/arch-at32uc3a0xxx/portmux.h b/include/asm-avr32/arch-at32uc3a0xxx/portmux
    .h
index 2877206..c9b17a8 100644
--- a/include/asm-avr32/arch-at32uc3a0xxx/portmux.h
+++ b/include/asm-avr32/arch-at32uc3a0xxx/portmux.h
@@ -25,7 +25,7 @@
 #include <asm/arch-common/portmux-gpio.h>
 #include <asm/arch/memory-map.h>

-#define PORTMUX_PORT(x)         ( (void *) (GPIO_BASE + (x) * 0x0100) )
+#define PORTMUX_PORT(x)         ((void *) (GPIO_BASE + (x) * 0x0100))
 #define PORTMUX_PORT_A          PORTMUX_PORT(0)
 #define PORTMUX_PORT_B          PORTMUX_PORT(1)
 #define PORTMUX_PORT_C          PORTMUX_PORT(2)
diff --git a/include/asm-avr32/arch-at32uc3a0xxx/clk.h b/include/asm-avr32/arch-at32uc3a0xxx/clk.h
index 1bfb721..eb94eaa 100644
--- a/include/asm-avr32/arch-at32uc3a0xxx/clk.h
+++ b/include/asm-avr32/arch-at32uc3a0xxx/clk.h
@@ -37,7 +37,6 @@ static inline unsigned long get_cpu_clk_rate(void)
 }
 static inline unsigned long get_hsb_clk_rate(void)
 {
-        //TODO HSB is always the same as cpu-rate
         return MAIN_CLK_RATE >> CFG_CLKDIV_CPU;
 }
 static inline unsigned long get_pba_clk_rate(void)
diff --git a/include/asm-avr32/arch-at32uc3a0xxx/memory-map.h b/include/asm-avr32/arch-at32uc3a0xxx/
    memory-map.h
index cef3807..3beaad9 100644
--- a/include/asm-avr32/arch-at32uc3a0xxx/memory-map.h
+++ b/include/asm-avr32/arch-at32uc3a0xxx/memory-map.h
@@ -70,6 +70,6 @@
 #define TC_BASE                                         0xFFFF3800
 #define ADC_BASE                                  0xFFFF3C00

-#define GPIO_PORT(x) ( (void *) (GPIO_BASE + (x) * 0x0100) )
+#define GPIO_PORT(x) ((void *) (GPIO_BASE + (x) * 0x0100))

 #endif /* __AT32UC3A0512_MEMORY_MAP_H__ */
diff --git a/include/asm-avr32/arch-at32uc3a0xxx/addrspace.h b/include/asm-avr32/arch-at32uc3a0xxx/
    addrspace.h
index 90feed7..0b8b3df 100644
--- a/include/asm-avr32/arch-at32uc3a0xxx/addrspace.h
+++ b/include/asm-avr32/arch-at32uc3a0xxx/addrspace.h
@@ -33,7 +33,7 @@ static __inline__ unsigned long virt_to_phys(volatile void *address)
         return PHYSADDR(address);
 }

-static __inline__ void * phys_to_virt(unsigned long address)
+static __inline__ void *phys_to_virt(unsigned long address)
 {
         return (void *)address;
 }
diff --git a/cpu/at32uc/cache.c b/cpu/at32uc/cache.c
index 06fa12c..d624e6f 100644
--- a/cpu/at32uc/cache.c
+++ b/cpu/at32uc/cache.c
@@ -28,7 +28,7 @@
  * RAM, and that the icache will look for it. Cleaning the dcache and
  * invalidating the icache will do the trick.
  */
-void  flush_cache (unsigned long start_addr, unsigned long size)
+void  flush_cache(unsigned long start_addr, unsigned long size)
 {
         /* No cache to clean in the at32uc3. */
 }
diff --git a/board/atmel/atevk1100/atevk1100.c b/board/atmel/atevk1100/atevk1100.c
index a85337e..e9c5452 100644
--- a/board/atmel/atevk1100/atevk1100.c
+++ b/board/atmel/atevk1100/atevk1100.c
@@ -88,7 +88,8 @@ phys_size_t initdram(int board_type)
         unsigned long actual_size;
         void *sram_base;

-        sram_base = map_physmem(EBI_SRAM_CS2_BASE, EBI_SRAM_CS2_SIZE, MAP_NOCACHE);
+        sram_base = map_physmem(EBI_SRAM_CS2_BASE, EBI_SRAM_CS2_SIZE,
+                                MAP_NOCACHE);

         expected_size = sram_init(&sram_config);
         actual_size = get_ram_size(sram_base, expected_size);
```

Appendix C

Unsubmitted U-Boot changes

```
diff --git a/board/atmel/atevk1100/atevk1100.c b/board/atmel/atevk1100/atevk1100.c
index e9c5452..d2d7893 100644
--- a/board/atmel/atevk1100/atevk1100.c
+++ b/board/atmel/atevk1100/atevk1100.c
@@ -105,7 +105,10 @@ phys_size_t initdram(int board_type)

 int board_early_init_r(void)
 {
-       /* Physical address of phy (0xff = auto-detect) */
+       /*
+        * Physical address of phy. This is not used when the address is
+        * autodetected. See CONFIG_MACB_SEARCH_PHY.
+        */
        gd->bd->bi_phy_id[0] = 0xff;
        return 0;
 }
diff --git a/drivers/net/macb.c b/drivers/net/macb.c
index 31a4fbe..c8beb82 100644
--- a/drivers/net/macb.c
+++ b/drivers/net/macb.c
@@ -327,6 +327,7 @@ static void macb_phy_reset(struct macb_device *macb)
                        netdev->name, status);
 }

+#ifdef CONFIG_MACB_SEARCH_PHY
 static int macb_phy_find(struct macb_device *macb)
 {
        int i;
@@ -347,6 +348,8 @@ static int macb_phy_find(struct macb_device *macb)

        return 0;
 }
+#endif /* CONFIG_MACB_SEARCH_PHY */
+

 static int macb_phy_init(struct macb_device *macb)
 {
@@ -356,12 +359,12 @@ static int macb_phy_init(struct macb_device *macb)
        int media, speed, duplex;
        int i;

-       if (macb->phy_addr == 0xff) {
-               /* Auto-detect phy_addr */
-               if (!macb_phy_find(macb)) {
-                       return 0;
-               }
+#ifdef CONFIG_MACB_SEARCH_PHY
+       /* Auto-detect phy_addr */
+       if (!macb_phy_find(macb)) {
+               return 0;
        }
+#endif /* CONFIG_MACB_SEARCH_PHY */

        /* Check if the PHY is up to snuff... */
        phy_id = macb_mdio_read(macb, MII_PHYSID1);
diff --git a/drivers/serial/atmel_usart.c b/drivers/serial/atmel_usart.c
index a358871..f3b146c 100644
--- a/drivers/serial/atmel_usart.c
+++ b/drivers/serial/atmel_usart.c
@@ -58,9 +58,6 @@ int serial_init(void)
```

```
 {
        usart3_writel(CR, USART3_BIT(RSTRX) | USART3_BIT(RSTTX));

-       /* Make sure that all interrupts are disabled during startup. */
-       usart3_writel(IDR, 0xffffffff);
-
        serial_setbrg();

        usart3_writel(CR, USART3_BIT(RXEN) | USART3_BIT(TXEN));
diff --git a/include/configs/atevk1100.h b/include/configs/atevk1100.h
index ad134f8..e6e4746 100644
--- a/include/configs/atevk1100.h
+++ b/include/configs/atevk1100.h
@@ -149,6 +149,10 @@
  * MII mode. Set CONFIG_MACB_FORCE10M flag if clock is too slow for 100Mbit.
  */
 #define CONFIG_MACB_FORCE10M           1
+/*
+ * On this board, the PHY can be found at different addresses (eiter 1 or 7).
+ */
+#define CONFIG_MACB_SEARCH_PHY         1

 #define CONFIG_ATMEL_USART             1
 #define CONFIG_ATMEL_SPI               1
```

Appendix D

Linux kernel patches

D.0 Cover letter

```
From 6677f489f529d76a17c5ab6900f81dbcfbc8b5d1 Mon Sep 17 00:00:00 2001
Content-Type: text/plain; charset=UTF-8
Message-Id: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <rangoy@mnops.(none)>
Date: Fri, 15 May 2009 13:59:33 +0200
Subject: [PATCH 00/29] AVR32: Support for EVK1100

These patches are the changes we made to Linux to make it possible to
run on the ATEVK1100 evaluation kit (with the UC3A0512ES
microcontroller).

We run Linux from 4 MB of SRAM added to the EVK1100. SDRAM hasn't been
tested.

What works:
- Booting linux
- Serial console
- Networking
- Root filesystem on NFS
- Loading FDPIC ELF files (only statically linked files).
- LEDs
- Booting busybox, running telnet server, ++++

What is known not to work:
- Debuging applications (ptrace)
- Shared libraries
- SPI, USB
- Has a tendency to crash when out of memory (which happens quite
  frequently with only 4 MB of RAM)


Patches in this series are in a somewhat random order, but the overall
pattern is:
1-19  Some changes making it easier to add AVR32A support.
20-26 AVR32A support
```

```
27-28 UC3A support
29    EVK1100 support

The line between the patches which add AVR32A support and the patches
which prepare for AVR32A support is somewhat fuzzy.

Note that these patches still needs a lot of work to be suited for
inclusion in the Linux kernel.

The changes we made to GCC and binutils will be posted later.

This patch series is coauthored by:
- Olav Morken <olavmrk@gmail.com>
- Gunnar Rangoy <gunnar@rangoy.com>
- Paul Driveklepp <pauldriveklepp@gmail.com>

Gunnar Rangoy (29):
  macb: limit to 10 Mbit/s if the clock is too slow to handle 100
    Mbit/s
  AVR32: Don't clear registers when starting a new thread.
  AVR32: split paging_init into mmu init, free memory init and
    exceptions init.
  AVR32: use task_pt_regs in copy_thread.
  AVR32: FDPIC ELF support.
  AVR32: Introduce AVR32_CACHE and AVR32_UNALIGNED Kconfig options
  AVR32: mm/tlb.c should only be enabled with CONFIG_MMU.
  AVR32: mm/fault for !CONFIG_MMU.
  AVR32: ioremap and iounmap for !CONFIG_MMU.
  AVR32: MMU dummy functions for chips without MMU.
  AVR32: mm_context_t for !CONFIG_MMU
  AVR32: Add cache-function stubs for chips without cache.
  AVR32: copy_user for chips that cannot do unaligned memory access.
  AVR32: csum_partial: Support chips that cannot do unaligned memory
    accesses.
  AVR32: avoid unaligned access in uaccess.h
  AVR32: memcpy implementation for chips that cannot do unaligned
    memory accesses.
  AVR32: Mark AVR32B specific assumptions with CONFIG_SUBARCH_AVR32B in
    strnlen.
  AVR32: mm/dma-coherent.c - ifdef AVR32B specific code.
  AVR32: Disable ret_if_privileged macro for !CONFIG_SUBARCH_AVR32B.
  AVR32: AVR32A support in Kconfig
  AVR32: AVR32A address space support.
  AVR32: Change maximum task size for AVR32A
  AVR32: Fix uaccess __range_ok macro for AVR32A.
  AVR32: Support for AVR32A (entry-avr32a.c)
  AVR32: Change HIGHMEM_START for AVR32A.
  AVR32: New pt_regs layout for AVR32A.
  AVR32: UC3A0512ES Interrupt bug workaround
  AVR32: UC3A0xxx-support
  AVR32: Board support for ATEVK1100

 arch/avr32/Kconfig                                 |   40 +-
```

```
 arch/avr32/Makefile                                |   17 +
 arch/avr32/boards/atevk1100/Makefile               |    1 +
 arch/avr32/boards/atevk1100/setup.c                |  121 ++
 arch/avr32/configs/atevk1100_defconfig             |  778 ++++++++++++
 arch/avr32/include/asm/addrspace.h                 |   12 +-
 arch/avr32/include/asm/asm.h                       |   28 +-
 arch/avr32/include/asm/checksum.h                  |   28 +
 arch/avr32/include/asm/elf.h                       |   10 +
 arch/avr32/include/asm/io.h                        |   29 +
 arch/avr32/include/asm/irqflags.h                  |    8 +
 arch/avr32/include/asm/mmu.h                       |   16 +
 arch/avr32/include/asm/mmu_context.h               |   40 +
 arch/avr32/include/asm/page.h                      |   13 +
 arch/avr32/include/asm/processor.h                 |    5 +-
 arch/avr32/include/asm/ptrace.h                    |   79 ++
 arch/avr32/include/asm/uaccess.h                   |   31 +-
 arch/avr32/kernel/Makefile                         |    1 +
 arch/avr32/kernel/cpu.c                            |    1 +
 arch/avr32/kernel/entry-avr32a.S                   |  705 +++++++++++
 arch/avr32/kernel/process.c                        |    2 +-
 arch/avr32/kernel/setup.c                          |   22 +
 arch/avr32/lib/Makefile                            |    7 +-
 arch/avr32/lib/copy_user-nounaligned.S             |  124 ++
 arch/avr32/lib/csum_partial.S                      |   31 +
 arch/avr32/lib/memcpy-nounaligned.S                |   86 ++
 arch/avr32/lib/strnlen_user.S                      |    4 +
 arch/avr32/mach-at32uc3a/Kconfig                   |   28 +
 arch/avr32/mach-at32uc3a/Makefile                  |    9 +
 arch/avr32/mach-at32uc3a/at32uc3a0xxx.c            | 1453 ++++++++++++++++++++++
 arch/avr32/mach-at32uc3a/clock.c                   |  270 ++++
 arch/avr32/mach-at32uc3a/clock.h                   |   30 +
 arch/avr32/mach-at32uc3a/cpufreq.c                 |  111 ++
 arch/avr32/mach-at32uc3a/extint.c                  |  279 ++++
 arch/avr32/mach-at32uc3a/gpio.c                    |  453 ++++++
 arch/avr32/mach-at32uc3a/gpio.h                    |   77 +
 arch/avr32/mach-at32uc3a/hmatrix.c                 |   88 ++
 arch/avr32/mach-at32uc3a/hsmc.c                    |  281 ++++
 arch/avr32/mach-at32uc3a/hsmc.h                    |  127 ++
 .../mach-at32uc3a/include/mach/at32uc3a0xxx.h      |   78 ++
 arch/avr32/mach-at32uc3a/include/mach/board.h      |  121 ++
 arch/avr32/mach-at32uc3a/include/mach/chip.h       |   21 +
 arch/avr32/mach-at32uc3a/include/mach/cpu.h        |   35 +
 arch/avr32/mach-at32uc3a/include/mach/gpio.h       |   45 +
 arch/avr32/mach-at32uc3a/include/mach/hmatrix.h    |   55 +
 arch/avr32/mach-at32uc3a/include/mach/init.h       |   18 +
 arch/avr32/mach-at32uc3a/include/mach/io.h         |   38 +
 arch/avr32/mach-at32uc3a/include/mach/irq.h        |   14 +
 arch/avr32/mach-at32uc3a/include/mach/pm.h         |   51 +
 arch/avr32/mach-at32uc3a/include/mach/portmux.h    |   29 +
 arch/avr32/mach-at32uc3a/include/mach/smc.h        |  113 ++
 arch/avr32/mach-at32uc3a/include/mach/sram.h       |   30 +
 arch/avr32/mach-at32uc3a/intc.c                    |  217 +++
 arch/avr32/mach-at32uc3a/intc.h                    |  329 +++++
```

```
 arch/avr32/mach-at32uc3a/pdca.c                    |   48 +
 arch/avr32/mach-at32uc3a/pm-at32uc3a0xxx.S         |  174 +++
 arch/avr32/mach-at32uc3a/pm.c                      |  243 ++++
 arch/avr32/mach-at32uc3a/pm.h                      |  112 ++
 arch/avr32/mach-at32uc3a/sdramc.h                  |   76 +
 arch/avr32/mm/Makefile                             |    3 +-
 arch/avr32/mm/cache-nocache.c                      |   36 +
 arch/avr32/mm/dma-coherent.c                       |    2 +
 arch/avr32/mm/fault-nommu.c                        |   19 +
 arch/avr32/mm/init.c                               |   48 +-
 arch/avr32/mm/ioremap-nommu.c                      |   31 +
 drivers/net/macb.c                                 |    7 +
 fs/Kconfig.binfmt                                  |    2 +-
 67 files changed, 7400 insertions(+), 40 deletions(-)
 create mode 100644 arch/avr32/boards/atevk1100/Makefile
 create mode 100644 arch/avr32/boards/atevk1100/setup.c
 create mode 100644 arch/avr32/configs/atevk1100_defconfig
 create mode 100644 arch/avr32/kernel/entry-avr32a.S
 create mode 100644 arch/avr32/lib/copy_user-nounaligned.S
 create mode 100644 arch/avr32/lib/memcpy-nounaligned.S
 create mode 100644 arch/avr32/mach-at32uc3a/Kconfig
 create mode 100644 arch/avr32/mach-at32uc3a/Makefile
 create mode 100644 arch/avr32/mach-at32uc3a/at32uc3a0xxx.c
 create mode 100644 arch/avr32/mach-at32uc3a/clock.c
 create mode 100644 arch/avr32/mach-at32uc3a/clock.h
 create mode 100644 arch/avr32/mach-at32uc3a/cpufreq.c
 create mode 100644 arch/avr32/mach-at32uc3a/extint.c
 create mode 100644 arch/avr32/mach-at32uc3a/gpio.c
 create mode 100644 arch/avr32/mach-at32uc3a/gpio.h
 create mode 100644 arch/avr32/mach-at32uc3a/hmatrix.c
 create mode 100644 arch/avr32/mach-at32uc3a/hsmc.c
 create mode 100644 arch/avr32/mach-at32uc3a/hsmc.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/at32uc3a0xxx.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/board.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/chip.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/cpu.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/gpio.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/hmatrix.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/init.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/io.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/irq.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/pm.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/portmux.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/smc.h
 create mode 100644 arch/avr32/mach-at32uc3a/include/mach/sram.h
 create mode 100644 arch/avr32/mach-at32uc3a/intc.c
 create mode 100644 arch/avr32/mach-at32uc3a/intc.h
 create mode 100644 arch/avr32/mach-at32uc3a/pdca.c
 create mode 100644 arch/avr32/mach-at32uc3a/pm-at32uc3a0xxx.S
 create mode 100644 arch/avr32/mach-at32uc3a/pm.c
 create mode 100644 arch/avr32/mach-at32uc3a/pm.h
 create mode 100644 arch/avr32/mach-at32uc3a/sdramc.h
 create mode 100644 arch/avr32/mm/cache-nocache.c
```

```
create mode 100644 arch/avr32/mm/fault-nommu.c
create mode 100644 arch/avr32/mm/ioremap-nommu.c
```

D.1 Network speed limiting

```
From 7e770576ff6338cdc7bf78c091229db93791aa54 Mon Sep 17 00:00:00 2001
Message-Id: <7e770576ff6338cdc7bf78c091229db93791aa54.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 13:03:30 +0200
Subject: [PATCH 01/29] macb: limit to 10 Mbit/s if the clock is too slow to handle 100 Mbit/s

The macb requires a 50 MHz clock to handle 100 Mbit/s in RMII mode, and
a 25 MHz clock to handle 100 Mbit/s in MII mode. This patch checks the
clock speed, and limits the PHY to 10 Mbit/s if the clock is too slow.
---
 drivers/net/macb.c |    7 +++++++
 1 files changed, 7 insertions(+), 0 deletions(-)

diff --git a/drivers/net/macb.c b/drivers/net/macb.c
index 01f7a31..9900dea 100644
--- a/drivers/net/macb.c
+++ b/drivers/net/macb.c
@@ -192,6 +192,7 @@ static int macb_mii_probe(struct net_device *dev)
        struct phy_device *phydev = NULL;
        struct eth_platform_data *pdata;
        int phy_addr;
+       unsigned long pclk_hz;

        /* find the first phy */
        for (phy_addr = 0; phy_addr < PHY_MAX_ADDR; phy_addr++) {
@@ -226,6 +227,12 @@ static int macb_mii_probe(struct net_device *dev)
        /* mask with MAC supported features */
        phydev->supported &= PHY_BASIC_FEATURES;

+       /* disable 100 Mbit if clock is too slow */
+       pclk_hz = clk_get_rate(bp->pclk);
+       if (pclk_hz < 25000000 ||
+           (pclk_hz < 50000000 && pdata && pdata->is_rmii))
+               phydev->supported &= ~SUPPORTED_100baseT_Half & ~SUPPORTED_100baseT_Full;
+
        phydev->advertising = phydev->supported;

        bp->link = 0;
--
1.6.2.2
```

D.2 Avoid register reset

```
From e38e8cf17d680df5b8c88be6fb5bdfdb90fd205c Mon Sep 17 00:00:00 2001
Message-Id: <e38e8cf17d680df5b8c88be6fb5bdfdb90fd205c.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 15:02:21 +0200
Subject: [PATCH 02/29] AVR32: Don't clear registers when starting a new thread.

Not certain about this patch, but we can't clear the registers here,
since the FDPIC ELF loader stores a pointer to the process' load map
in a register before this function is called.
---
 arch/avr32/include/asm/processor.h |    1 -
 1 files changed, 0 insertions(+), 1 deletions(-)

diff --git a/arch/avr32/include/asm/processor.h b/arch/avr32/include/asm/processor.h
index 49a88f5..3fb964d 100644
--- a/arch/avr32/include/asm/processor.h
+++ b/arch/avr32/include/asm/processor.h
@@ -132,7 +132,6 @@ struct thread_struct {
 #define start_thread(regs, new_pc, new_sp)      \
        do {                                    \
                set_fs(USER_DS);                \
```

```
-               memset(regs, 0, sizeof(*regs));  \
                regs->sr = MODE_USER;            \
                regs->pc = new_pc & ~1;          \
                regs->sp = new_sp;               \
--
1.6.2.2
```

D.3 Split paging function

```
From a55ab1e0c9cd149ae4e55f05ffc368aa1807a80f Mon Sep 17 00:00:00 2001
Message-Id: <a55ab1e0c9cd149ae4e55f05ffc368aa1807a80f.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 12:55:23 +0200
Subject: [PATCH 03/29] AVR32: split paging_init into mmu init, free memory init and exceptions init.

This change is necessary to allow AVR32A to initialize free memory
and exceptions without having an MMU.
---
 arch/avr32/kernel/setup.c |   22 ++++++++++++++++++++
 arch/avr32/mm/init.c      |   48 ++++++++++++++++++++++++++++++++-------------
 2 files changed, 45 insertions(+), 25 deletions(-)

diff --git a/arch/avr32/kernel/setup.c b/arch/avr32/kernel/setup.c
index 5c70839..f7e734a 100644
--- a/arch/avr32/kernel/setup.c
+++ b/arch/avr32/kernel/setup.c
@@ -536,6 +536,26 @@ static void __init setup_bootmem(void)
        }
 }

+/*
+ * exceptions_init() initializes exception handling.
+ *
+ * This function sets the exception handler vector and enables
+ * exceptions.
+ */
+void __init exceptions_init(void)
+{
+       extern unsigned long _evba;
+
+       printk("Exception vectors start at %p\n", &_evba);
+       sysreg_write(EVBA, (unsigned long)&_evba);
+
+       /*
+        * Since we are ready to handle exceptions now, we should let
+        * the CPU generate them...
+        */
+       __asm__ __volatile__ ("csrf %0" : : "i"(SR_EM_BIT));
+}
+
 void __init setup_arch (char **cmdline_p)
 {
        struct clk *cpu_clk;
@@ -589,6 +609,8 @@ void __init setup_arch (char **cmdline_p)
        conswitchp = &dummy_con;
 #endif

+       exceptions_init();
+
        paging_init();
        resource_init();
 }
diff --git a/arch/avr32/mm/init.c b/arch/avr32/mm/init.c
index fa92ff6..646f935 100644
--- a/arch/avr32/mm/init.c
+++ b/arch/avr32/mm/init.c
@@ -33,36 +33,21 @@ pgd_t swapper_pg_dir[PTRS_PER_PGD] __page_aligned;
 struct page *empty_zero_page;
 EXPORT_SYMBOL(empty_zero_page);

+#ifdef CONFIG_MMU
 /*
  * Cache of MMU context last used.
  */
 unsigned long mmu_context_cache = NO_CONTEXT;

-/*
```

```
- * paging_init() sets up the page tables
+/**
+ * Initialize the MMU.
  *
- * This routine also unmaps the page at virtual kernel address 0, so
- * that we can trap those pesky NULL-reference errors in the kernel.
+ * This function also reserves the zero-page, so that we can trap
+ * NULL-references
  */
-void __init paging_init(void)
+static void mmu_init(void)
 {
-       extern unsigned long _evba;
        void *zero_page;
-       int nid;
-
-       /*
-        * Make sure we can handle exceptions before enabling
-        * paging. Not that we should ever _get_ any exceptions this
-        * early, but you never know...
-        */
-       printk("Exception vectors start at %p\n", &_evba);
-       sysreg_write(EVBA, (unsigned long)&_evba);
-
-       /*
-        * Since we are ready to handle exceptions now, we should let
-        * the CPU generate them...
-        */
-       __asm__ __volatile__ ("csrf %0" : : "i"(SR_EM_BIT));

        /*
         * Allocate the zero page. The allocator will panic if it
@@ -75,6 +60,23 @@ void __init paging_init(void)
        enable_mmu();
        printk ("CPU: Paging enabled\n");

+       memset(zero_page, 0, PAGE_SIZE);
+       empty_zero_page = virt_to_page(zero_page);
+       flush_dcache_page(empty_zero_page);
+}
+#endif /* CONFIG_MMU */
+
+/**
+ * Initializes the MMU, and configures available memory.
+ */
+void __init paging_init(void)
+{
+       int nid;
+
+#ifdef CONFIG_MMU
+       mmu_init();
+#endif /* CONFIG_MMU */
+
        for_each_online_node(nid) {
                pg_data_t *pgdat = NODE_DATA(nid);
                unsigned long zones_size[MAX_NR_ZONES];
@@ -96,10 +98,6 @@ void __init paging_init(void)
        }

        mem_map = NODE_DATA(0)->node_mem_map;
-
-       memset(zero_page, 0, PAGE_SIZE);
-       empty_zero_page = virt_to_page(zero_page);
-       flush_dcache_page(empty_zero_page);
 }

 void __init mem_init(void)
--
1.6.2.2
```

D.4 Use task_pt_regs macro

```
From 12a2a1a6382f327843b4d637d12266366dd858ff Mon Sep 17 00:00:00 2001
Message-Id: <12a2a1a6382f327843b4d637d12266366dd858ff.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 15:13:37 +0200
Subject: [PATCH 04/29] AVR32: use task_pt_regs in copy_thread.
```

```
8
9  We already have the task_pt_regs macro, so we might as well use it.
10 ---
11  arch/avr32/kernel/process.c |    2 +-
12  1 files changed, 1 insertions(+), 1 deletions(-)
13
14 diff --git a/arch/avr32/kernel/process.c b/arch/avr32/kernel/process.c
15 index 134d530..fd37fcf 100644
16 --- a/arch/avr32/kernel/process.c
17 +++ b/arch/avr32/kernel/process.c
18 @@ -337,7 +337,7 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long usp,
19  {
20          struct pt_regs *childregs;
21
22 -        childregs = ((struct pt_regs *)(THREAD_SIZE + (unsigned long)task_stack_page(p))) - 1;
23 +        childregs = task_pt_regs(p);
24          *childregs = *regs;
25
26          if (user_mode(regs))
27 --
28 1.6.2.2
```

D.5 FDPIC ELF support

```
1  From 81a39fa959dffb95fec1634018f4137840e4c2a2 Mon Sep 17 00:00:00 2001
2  Message-Id: <81a39fa959dffb95fec1634018f4137840e4c2a2.1242388774.git.rangoy@mnops.(none)>
3  In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
4  References: <cover.1242388773.git.rangoy@mnops.(none)>
5  From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
6  Date: Fri, 24 Apr 2009 15:19:09 +0200
7  Subject: [PATCH 05/29] AVR32: FDPIC ELF support.
8
9  This patch introduces code necessary to load FDPIC files on AVR32.
10 ---
11  arch/avr32/include/asm/elf.h |   10 ++++++++++
12  fs/Kconfig.binfmt            |    2 +-
13  2 files changed, 11 insertions(+), 1 deletions(-)
14
15 diff --git a/arch/avr32/include/asm/elf.h b/arch/avr32/include/asm/elf.h
16 index d5d1d41..93ead6f 100644
17 --- a/arch/avr32/include/asm/elf.h
18 +++ b/arch/avr32/include/asm/elf.h
19 @@ -61,10 +61,15 @@ typedef elf_greg_t elf_gregset_t[ELF_NGREG];
20
21  typedef struct user_fpu_struct elf_fpregset_t;
22
23 +/* CPU specific flag for FDPIC. */
24 +#define EF_AVR32_FDPIC          0x04
25 +
26  /*
27   * This is used to ensure we don't load something for the wrong architecture.
28   */
29  #define elf_check_arch(x) ( (x)->e_machine == EM_AVR32 )
30 +#define elf_check_fdpic(x) ( (x)->e_flags & EF_AVR32_FDPIC )
31 +#define elf_check_const_displacement(x) 0
32
33  /*
34   * These are used to set parameters in the core dumps.
35 @@ -77,6 +82,11 @@ typedef struct user_fpu_struct elf_fpregset_t;
36  #endif
37  #define ELF_ARCH        EM_AVR32
38
39 +#define ELF_FDPIC_PLAT_INIT(_regs, _exec_map_addr, _interp_map_addr, _dynamic_addr)     \
40 +do {                                                                                    \
41 +        _regs->r0       = _exec_map_addr;                                               \
42 +} while(0)
43 +
44  #define USE_ELF_CORE_DUMP
45  #define ELF_EXEC_PAGESIZE       4096
46
47 diff --git a/fs/Kconfig.binfmt b/fs/Kconfig.binfmt
48 index ce9fb3f..9dabb33 100644
49 --- a/fs/Kconfig.binfmt
50 +++ b/fs/Kconfig.binfmt
51 @@ -30,7 +30,7 @@ config COMPAT_BINFMT_ELF
52  config BINFMT_ELF_FDPIC
53          bool "Kernel support for FDPIC ELF binaries"
54          default y
55 -        depends on (FRV || BLACKFIN || (SUPERH32 && !MMU))
```

```
+       depends on (FRV || BLACKFIN || (SUPERH32 && !MMU) || (AVR32 && !MMU))
        help
          ELF FDPIC binaries are based on ELF, but allow the individual load
          segments of a binary to be located in memory independently of each
--
1.6.2.2
```

D.6 Introduce cache and aligned flags

```
From f761e27a785a2dcd94ebee6b6f1f60f09ba6d703 Mon Sep 17 00:00:00 2001
Message-Id: <f761e27a785a2dcd94ebee6b6f1f60f09ba6d703.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 13:25:41 +0200
Subject: [PATCH 06/29] AVR32: Introduce AVR32_CACHE and AVR32_UNALIGNED Kconfig options

Reorganizes Kconfig a bit, and adds AVR32_CACHE (for CPUs with cache),
and AVR32_UNALIGNED (for CPUs that can do unaligned accesses). Also adds
three Makefile-variables: $(MMUEXT), $(CACHEEXT) and $(ALIGNEXT). These
are set to '-nommu' for !MMU, '-nocache' for !AVR32_CACHE and
'-nounaligned' for !AVR32_UNALIGNED.
---
 arch/avr32/Kconfig  |   17 +++++++++++++---
 arch/avr32/Makefile |   14 ++++++++++++
 2 files changed, 28 insertions(+), 3 deletions(-)

diff --git a/arch/avr32/Kconfig b/arch/avr32/Kconfig
index 26eca87..9e984b0 100644
--- a/arch/avr32/Kconfig
+++ b/arch/avr32/Kconfig
@@ -78,20 +78,31 @@ menu "System Type and features"

 source "kernel/time/Kconfig"

-config SUBARCH_AVR32B
-       bool
 config MMU
        bool
+
+config SUBARCH_AVR32B
+       bool
+       select MMU
+
 config PERFORMANCE_COUNTERS
        bool

+config AVR32_CACHE
+       bool
+
+config AVR32_UNALIGNED
+       bool
+
+
 config PLATFORM_AT32AP
        bool
        select SUBARCH_AVR32B
-       select MMU
        select PERFORMANCE_COUNTERS
        select ARCH_REQUIRE_GPIOLIB
        select GENERIC_ALLOCATOR
+       select AVR32_CACHE
+       select AVR32_UNALIGNED

 #
 # CPU types
diff --git a/arch/avr32/Makefile b/arch/avr32/Makefile
index b088e10..4864cb1 100644
--- a/arch/avr32/Makefile
+++ b/arch/avr32/Makefile
@@ -9,6 +9,20 @@
 .PHONY: all
 all: uImage vmlinux.elf

+ifeq ($(CONFIG_MMU),)
+MMUEXT=-nommu
+endif
+
+ifeq ($(CONFIG_AVR32_CACHE),)
```

```
+CACHEEXT=-nocache
+endif
+
+ifeq ($(CONFIG_AVR32_UNALIGNED),)
+ALIGNEXT=-nounaligned
+endif
+
+export MMUEXT CACHEEXT ALIGNEXT
+
 KBUILD_DEFCONFIG        := atstk1002_defconfig

 KBUILD_CFLAGS  += -pipe -fno-builtin -mno-pic
--
1.6.2.2
```

D.7 Disable mm-tlb.c

```
From 53cafff9ec5f54beb17130137ad46bd3d70dc781 Mon Sep 17 00:00:00 2001
Message-Id: <53cafff9ec5f54beb17130137ad46bd3d70dc781.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 12:58:23 +0200
Subject: [PATCH 07/29] AVR32: mm/tlb.c should only be enabled with CONFIG_MMU.

---
 arch/avr32/mm/Makefile |    3 ++-
 1 files changed, 2 insertions(+), 1 deletions(-)

diff --git a/arch/avr32/mm/Makefile b/arch/avr32/mm/Makefile
index 0066491..7d61b2c 100644
--- a/arch/avr32/mm/Makefile
+++ b/arch/avr32/mm/Makefile
@@ -3,4 +3,5 @@
 #

 obj-y                           += init.o clear_page.o copy_page.o dma-coherent.o
-obj-y                           += ioremap.o cache.o fault.o tlb.o
+obj-y                           += ioremap.o cache.o fault.o
+obj-$(CONFIG_MMU)               += tlb.o
--
1.6.2.2
```

D.8 fault.c for !CONFIG_MMU

```
From c3d37edc9dea393d59ca2186e41e8ec136cea69d Mon Sep 17 00:00:00 2001
Message-Id: <c3d37edc9dea393d59ca2186e41e8ec136cea69d.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 13:01:42 +0200
Subject: [PATCH 08/29] AVR32: mm/fault for !CONFIG_MMU.

This patch adds do_page_fault and do_bus_error for chips without MMU.
---
 arch/avr32/mm/Makefile      |    2 +-
 arch/avr32/mm/fault-nommu.c |   19 +++++++++++++++++++
 2 files changed, 20 insertions(+), 1 deletions(-)
 create mode 100644 arch/avr32/mm/fault-nommu.c

diff --git a/arch/avr32/mm/Makefile b/arch/avr32/mm/Makefile
index 7d61b2c..7dbd5a6 100644
--- a/arch/avr32/mm/Makefile
+++ b/arch/avr32/mm/Makefile
@@ -3,5 +3,5 @@
 #

 obj-y                           += init.o clear_page.o copy_page.o dma-coherent.o
-obj-y                           += ioremap.o cache.o fault.o
+obj-y                           += ioremap.o cache.o fault$(MMUEXT).o
 obj-$(CONFIG_MMU)               += tlb.o
diff --git a/arch/avr32/mm/fault-nommu.c b/arch/avr32/mm/fault-nommu.c
new file mode 100644
index 0000000..a3ebd4f
```

```
--- /dev/null
+++ b/arch/avr32/mm/fault-nommu.c
@@ -0,0 +1,19 @@
+#include <linux/mm.h>
+#include <linux/kdebug.h>
+
+#include <asm/sysreg.h>
+
+asmlinkage void do_page_fault(unsigned long ecr, struct pt_regs *regs)
+{
+       /* As we don't enable the MPU, a page fault should never occur. */
+       panic("Impossible page fault");
+}
+
+asmlinkage void do_bus_error(unsigned long addr, int write_access,
+                            struct pt_regs *regs)
+{
+       printk(KERN_ALERT
+              "Bus error at physical address 0x%08lx (%s access)\n",
+              addr, write_access ? "write" : "read");
+       die("Bus Error", regs, SIGKILL);
+}
--
1.6.2.2
```

D.9 ioremap and iounmap for !CONFIG_MMU

```
From 3b734d9d1e08ebc1776de70dd47e7e6fc480f29b Mon Sep 17 00:00:00 2001
Message-Id: <3b734d9d1e08ebc1776de70dd47e7e6fc480f29b.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 13:02:18 +0200
Subject: [PATCH 09/29] AVR32: ioremap and iounmap for !CONFIG_MMU.

---
 arch/avr32/mm/Makefile        |    2 +-
 arch/avr32/mm/ioremap-nommu.c |   31 +++++++++++++++++++++++++++++++
 2 files changed, 32 insertions(+), 1 deletions(-)
 create mode 100644 arch/avr32/mm/ioremap-nommu.c

diff --git a/arch/avr32/mm/Makefile b/arch/avr32/mm/Makefile
index 7dbd5a6..be29aee 100644
--- a/arch/avr32/mm/Makefile
+++ b/arch/avr32/mm/Makefile
@@ -3,5 +3,5 @@
 #

 obj-y                          += init.o clear_page.o copy_page.o dma-coherent.o
-obj-y                          += ioremap.o cache.o fault$(MMUEXT).o
+obj-y                          += ioremap$(MMUEXT).o cache.o fault$(MMUEXT).o
 obj-$(CONFIG_MMU)              += tlb.o
diff --git a/arch/avr32/mm/ioremap-nommu.c b/arch/avr32/mm/ioremap-nommu.c
new file mode 100644
index 0000000..52e6fe2
--- /dev/null
+++ b/arch/avr32/mm/ioremap-nommu.c
@@ -0,0 +1,31 @@
+/*
+ * Copyright (C) 2004-2006 Atmel Corporation
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the Free Software Foundation.
+ */
+#include <linux/vmalloc.h>
+#include <linux/mm.h>
+#include <linux/module.h>
+#include <linux/io.h>
+
+#include <asm/pgtable.h>
+#include <asm/addrspace.h>
+
+/*
+ * Re-map an arbitrary physical address space into the kernel virtual
+ * address space. Needed when the kernel wants to access physical
+ * memory directly.
+ */
+void __iomem *__ioremap(unsigned long phys_addr, size_t size,
```

```
+                        unsigned long flags)
+{
+       return (void __iomem *)(phys_addr);
+}
+EXPORT_SYMBOL(__ioremap);
+
+void __iounmap(void __iomem *addr)
+{
+}
+EXPORT_SYMBOL(__iounmap);
--
1.6.2.2
```

D.10 MMU dummy functions

```
From c7e41c4af45d365ad83ec58f67c64226244531cc Mon Sep 17 00:00:00 2001
Message-Id: <c7e41c4af45d365ad83ec58f67c64226244531cc.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 14:37:03 +0200
Subject: [PATCH 10/29] AVR32: MMU dummy functions for chips without MMU.

---
 arch/avr32/include/asm/mmu_context.h |   40 ++++++++++++++++++++++++++++++++++
 1 files changed, 40 insertions(+), 0 deletions(-)

diff --git a/arch/avr32/include/asm/mmu_context.h b/arch/avr32/include/asm/mmu_context.h
index 27ff234..edf97bf 100644
--- a/arch/avr32/include/asm/mmu_context.h
+++ b/arch/avr32/include/asm/mmu_context.h
@@ -12,6 +12,8 @@
 #ifndef __ASM_AVR32_MMU_CONTEXT_H
 #define __ASM_AVR32_MMU_CONTEXT_H

+#ifdef CONFIG_MMU
+
 #include <asm/tlbflush.h>
 #include <asm/sysreg.h>
 #include <asm-generic/mm_hooks.h>
@@ -145,4 +147,42 @@ static inline void disable_mmu(void)
        sysreg_write(MMUCR, SYSREG_BIT(MMUCR_S));
 }

+#else /* CONFIG_MMU */
+
+static inline void enter_lazy_tlb(struct mm_struct *mm, struct task_struct *tsk)
+{
+}
+
+static inline void switch_mm(struct mm_struct *prev,
+                             struct mm_struct *next,
+                             struct task_struct *tsk)
+{
+       /* Nothing to do when we don't have an MMU. */
+}
+
+/*
+ * Initialize the context related info for a new mm_struct
+ * instance.
+ */
+static inline int init_new_context(struct task_struct *tsk,
+                                   struct mm_struct *mm)
+{
+       return 0;
+}
+
+/*
+ * Destroy context related info for an mm_struct that is about
+ * to be put to rest.
+ */
+static inline void destroy_context(struct mm_struct *mm)
+{
+       /* Do nothing */
+}
+
+#define deactivate_mm(tsk,mm) do { } while(0)
+
+#define activate_mm(prev, next) switch_mm((prev), (next), NULL)
```

```
+
+#endif /* CONFIG_MMU */
+
 #endif /* __ASM_AVR32_MMU_CONTEXT_H */
--
1.6.2.2
```

D.11 mm_context_t for !CONFIG_MMU

```
From 939407126d16c2476ac32d114e4ccd5fee26dec6 Mon Sep 17 00:00:00 2001
Message-Id: <939407126d16c2476ac32d114e4ccd5fee26dec6.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 15:19:51 +0200
Subject: [PATCH 11/29] AVR32: mm_context_t for !CONFIG_MMU

This patch adds a struct for the mm_context_t for architectures without
an MMU. This needs to be a struct because it needs to contain some
specific "variables".
---
 arch/avr32/include/asm/mmu.h |   16 ++++++++++++++++
 1 files changed, 16 insertions(+), 0 deletions(-)

diff --git a/arch/avr32/include/asm/mmu.h b/arch/avr32/include/asm/mmu.h
index 60c2d26..f02a409 100644
--- a/arch/avr32/include/asm/mmu.h
+++ b/arch/avr32/include/asm/mmu.h
@@ -1,10 +1,26 @@
 #ifndef __ASM_AVR32_MMU_H
 #define __ASM_AVR32_MMU_H

+#ifdef CONFIG_MMU
+
 /* Default "unsigned long" context */
 typedef unsigned long mm_context_t;

 #define MMU_ITLB_ENTRIES       64
 #define MMU_DTLB_ENTRIES       64

+#else /* CONFIG_MMU */
+
+typedef struct {
+       struct vm_list_struct   *vmlist;
+       unsigned long           end_brk;
+
+#ifdef CONFIG_BINFMT_ELF_FDPIC
+       unsigned long   exec_fdpic_loadmap;
+       unsigned long   interp_fdpic_loadmap;
+#endif
+} mm_context_t;
+
+#endif /* CONFIG_MMU */
+
 #endif /* __ASM_AVR32_MMU_H */
--
1.6.2.2
```

D.12 Add cache function stubs

```
From 7f8c979a06b515f65499763692cfc6444ca6c8d2 Mon Sep 17 00:00:00 2001
Message-Id: <7f8c979a06b515f65499763692cfc6444ca6c8d2.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 12:59:23 +0200
Subject: [PATCH 12/29] AVR32: Add cache-function stubs for chips without cache.

---
 arch/avr32/mm/Makefile        |    2 +-
 arch/avr32/mm/cache-nocache.c |   36 ++++++++++++++++++++++++++++++++++++
 2 files changed, 37 insertions(+), 1 deletions(-)
 create mode 100644 arch/avr32/mm/cache-nocache.c

```

```
diff --git a/arch/avr32/mm/Makefile b/arch/avr32/mm/Makefile
index be29aee..52c751c 100644
--- a/arch/avr32/mm/Makefile
+++ b/arch/avr32/mm/Makefile
@@ -3,5 +3,5 @@
 #

 obj-y                           += init.o clear_page.o copy_page.o dma-coherent.o
-obj-y                           += ioremap$(MMUEXT).o cache.o fault$(MMUEXT).o
+obj-y                           += ioremap$(MMUEXT).o cache$(CACHEEXT).o fault$(MMUEXT).o
 obj-$(CONFIG_MMU)               += tlb.o
diff --git a/arch/avr32/mm/cache-nocache.c b/arch/avr32/mm/cache-nocache.c
new file mode 100644
index 0000000..ec6198d
--- /dev/null
+++ b/arch/avr32/mm/cache-nocache.c
@@ -0,0 +1,36 @@
+#include <asm/cacheflush.h>
+
+void invalidate_dcache_region(void *start, size_t size)
+{
+}
+
+void clean_dcache_region(void *start, size_t size)
+{
+}
+
+void flush_dcache_region(void *start, size_t size)
+{
+}
+
+void invalidate_icache_region(void *start, size_t size)
+{
+}
+
+void flush_icache_range(unsigned long start, unsigned long end)
+{
+}
+
+void flush_icache_page(struct vm_area_struct *vma, struct page *page)
+{
+}
+
+asmlinkage int sys_cacheflush(int operation, void __user *addr, size_t len)
+{
+       return 0;
+}
+
+void copy_to_user_page(struct vm_area_struct *vma, struct page *page,
+               unsigned long vaddr, void *dst, const void *src,
+               unsigned long len)
+{
+}
--
1.6.2.2
```

D.13 copy_user.S for !CONFIG_NOUNALIGNED

```
From 3222aad2fd0652f62e5d7fdfe61a7740e38a7b51 Mon Sep 17 00:00:00 2001
Message-Id: <3222aad2fd0652f62e5d7fdfe61a7740e38a7b51.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Thu, 30 Apr 2009 14:51:12 +0200
Subject: [PATCH 13/29] AVR32: copy_user for chips that cannot do unaligned memory access.

---
 arch/avr32/lib/Makefile                |    4 +-
 arch/avr32/lib/copy_user-nounaligned.S |  124 ++++++++++++++++++++++++++++++++
 2 files changed, 127 insertions(+), 1 deletions(-)
 create mode 100644 arch/avr32/lib/copy_user-nounaligned.S

diff --git a/arch/avr32/lib/Makefile b/arch/avr32/lib/Makefile
index 084d95b..be35b6a 100644
--- a/arch/avr32/lib/Makefile
+++ b/arch/avr32/lib/Makefile
@@ -2,10 +2,12 @@
 # Makefile for AVR32-specific library files
 #
```

```
22
23 -lib-y  := copy_user.o clear_user.o
24 +lib-y  := clear_user.o
25  lib-y  += strncpy_from_user.o strnlen_user.o
26  lib-y  += delay.o memset.o memcpy.o findbit.o
27  lib-y  += csum_partial.o csum_partial_copy_generic.o
28  lib-y  += io-readsw.o io-readsl.o io-writesw.o io-writesl.o
29  lib-y  += io-readsb.o io-writesb.o
30  lib-y  += __avr32_lsl64.o __avr32_lsr64.o __avr32_asr64.o
31 +lib-y  += copy_user$(ALIGNEXT).o
32 +
33 diff --git a/arch/avr32/lib/copy_user-nounaligned.S b/arch/avr32/lib/copy_user-nounaligned.S
34 new file mode 100644
35 index 0000000..1bf9d8d
36 --- /dev/null
37 +++ b/arch/avr32/lib/copy_user-nounaligned.S
38 @@ -0,0 +1,124 @@
39 +/*
40 + * Copy to/from userspace with optional address space checking.
41 + *
42 + * Copyright 2004-2006 Atmel Corporation
43 + *
44 + * This program is free software; you can redistribute it and/or modify
45 + * it under the terms of the GNU General Public License version 2 as
46 + * published by the Free Software Foundation.
47 + */
48 +#include <asm/page.h>
49 +#include <asm/thread_info.h>
50 +#include <asm/asm.h>
51 +
52 +       /*
53 +        * __kernel_size_t
54 +        * __copy_user(void *to, const void *from, __kernel_size_t n)
55 +        *
56 +        * Returns the number of bytes not copied. Might be off by
57 +        * max 3 bytes if we get a fault in the main loop.
58 +        *
59 +        * The address-space checking functions simply fall through to
60 +        * the non-checking version.
61 +        */
62 +       .text
63 +       .align  1
64 +       .global copy_from_user
65 +       .type   copy_from_user, @function
66 +copy_from_user:
67 +       branch_if_kernel r8, __copy_user
68 +       ret_if_privileged r8, r11, r10, r10
69 +       rjmp    __copy_user
70 +       .size   copy_from_user, . - copy_from_user
71 +
72 +       .global copy_to_user
73 +       .type   copy_to_user, @function
74 +copy_to_user:
75 +       branch_if_kernel r8, __copy_user
76 +       ret_if_privileged r8, r12, r10, r10
77 +       .size   copy_to_user, . - copy_to_user
78 +
79 +       .global __copy_user
80 +       .type   __copy_user, @function
81 +__copy_user:
82 +
83 +       /* First we check whether from or to are unaligned */
84 +       mov     r9, r11
85 +       andl    r9, 3, COH
86 +       mov     r8, r12
87 +       andl    r8, 3, COH
88 +
89 +       /* Is it impossible to align both? Branch to single-byte copies
90 +        * if we can't align both.
91 +        */
92 +       cp.w    r8, r9
93 +       brne    4f
94 +
95 +       /* Do they need alignment? */
96 +       cp.w    r9, 0
97 +       brne    6f
98 +
99 +       /* At this point, both from and to are word-aligned */
100 +1:     sub     r10, 4
101 +       brlt    3f
102 +
103 +2:
104 +10:    ld.w    r8, r11++
105 +11:    st.w    r12++, r8
```

```
+        sub     r10, 4
+        brge    2b
+
+3:      sub     r10, -4
+        reteq   0
+
+        /*
+         * Do byte copies. This takes care of unaligned count and those cases
+         * where we are unable to align both from and to on word-boundaries.
+         * Need to be careful with r10 here so that we return the correct
+         * value even if we get a fault
+         */
+4:      sub     r10, 1
+        retlt   0
+20:     ld.ub   r8, r11++
+21:     st.b    r12++, r8
+        rjmp    4b
+
+        /* Handle unaligned from/to-pointer */
+6:
+        cp.w    r10, 4
+        brlt    4b
+        rsub    r9, r9, 4
+
+30:     ld.ub   r8, r11++
+31:     st.b    r12++, r8
+        sub     r10, 1
+        sub     r9, 1
+        breq    1b
+32:     ld.ub   r8, r11++
+33:     st.b    r12++, r8
+        sub     r10, 1
+        sub     r9, 1
+        breq    1b
+34:     ld.ub   r8, r11++
+35:     st.b    r12++, r8
+        sub     r10, 1
+        rjmp    1b
+        .size   __copy_user, . - __copy_user
+
+        .section .fixup,"ax"
+        .align  1
+19:     sub     r10, -4
+29:     retal   r10
+
+        .section __ex_table,"a"
+        .align  2
+        .long   10b, 19b
+        .long   11b, 19b
+        .long   20b, 29b
+        .long   21b, 29b
+        .long   30b, 29b
+        .long   31b, 29b
+        .long   32b, 29b
+        .long   33b, 29b
+        .long   34b, 29b
+        .long   35b, 29b
--
1.6.2.2
```

D.14 csum_partial: support for chips that cannot do unaligned accesses

```
From aa418bb9eeb14bf5c225b94bd0e3c3a2e5aeb18b Mon Sep 17 00:00:00 2001
Message-Id: <aa418bb9eeb14bf5c225b94bd0e3c3a2e5aeb18b.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Thu, 23 Apr 2009 15:50:11 +0200
Subject: [PATCH 14/29] AVR32: csum_partial: Support chips that cannot do unaligned memory accesses.

---
 arch/avr32/include/asm/checksum.h |   28 ++++++++++++++++++++++++++++
 arch/avr32/lib/csum_partial.S     |   31 +++++++++++++++++++++++++++++++
 2 files changed, 59 insertions(+), 0 deletions(-)

diff --git a/arch/avr32/include/asm/checksum.h b/arch/avr32/include/asm/checksum.h
index 4ddbfd2..865147f 100644
```

```
--- a/arch/avr32/include/asm/checksum.h
+++ b/arch/avr32/include/asm/checksum.h
@@ -44,15 +44,43 @@ static inline
 __wsum csum_partial_copy_nocheck(const void *src, void *dst,
                                  int len, __wsum sum)
 {
+#ifdef CONFIG_AVR32_UNALIGNED
        return csum_partial_copy_generic(src, dst, len, sum, NULL, NULL);
+#else
+       if (((unsigned long)src & 3) == 0 && ((unsigned long)dst & 3) == 0) {
+               /* Both src & dst are aligned. Do it the fast way. */
+               return csum_partial_copy_generic(src, dst, len, sum, NULL, NULL);
+       }
+
+       /* Unaligned. Do it the slow way. */
+       memcpy(dst, src, len);
+       return csum_partial(dst, len, sum);
+#endif
 }

 static inline
 __wsum csum_partial_copy_from_user(const void __user *src, void *dst,
                                    int len, __wsum sum, int *err_ptr)
 {
+#ifdef CONFIG_AVR32_UNALIGNED
        return csum_partial_copy_generic((const void __force *)src, dst, len,
                                         sum, err_ptr, NULL);
+#else
+       int missing;
+
+       if (((unsigned long)src & 3) == 0 && ((unsigned long)dst & 3) == 0) {
+               /* Both src & dst are aligned. Do it the fast way. */
+               return csum_partial_copy_generic(src, dst, len, sum, NULL, NULL);
+       }
+
+       missing = copy_from_user(dst, src, len);
+       if (missing) {
+               memset(dst + len - missing, 0, missing);
+               *err_ptr = -EFAULT;
+       }
+
+       return csum_partial(dst, len, sum);
+#endif
 }

 /*
diff --git a/arch/avr32/lib/csum_partial.S b/arch/avr32/lib/csum_partial.S
index 6a262b5..d1906bb 100644
--- a/arch/avr32/lib/csum_partial.S
+++ b/arch/avr32/lib/csum_partial.S
@@ -18,6 +18,14 @@ csum_partial:
        /* checksum complete words, aligned or not */
 3:     sub     r11, 4
        brlt    5f
+
+#ifndef CONFIG_AVR32_UNALIGNED
+       /* check whether the buffer is aligned */
+       mov     r8, r12
+       andl    r8, 3, COH
+       brne    8f
+#endif
+
 4:     ld.w    r9, r12++
        add     r10, r9
        acr     r10
@@ -33,7 +41,13 @@ csum_partial:
        mov     r8, 0
        cp      r11, 2
        brlt    6f
+#ifndef CONFIG_AVR32_UNALIGNED
+       ld.ub   r9, r12[1]
+       ldins.b r9:1, r12[0]
+       sub     r12, -2
+#else
        ld.uh   r9, r12++
+#endif
        sub     r11, 2
        breq    7f
        lsl     r9, 16
@@ -44,4 +58,21 @@ csum_partial:
        acr     r10

        retal   r10
+
```

```
+#ifndef CONFIG_AVR32_UNALIGNED
+       /* do unaligned loads */
+8:     ld.ub   r9, r12[3]
+       ldins.b r9:l, r12[2]
+       ldins.b r9:u, r12[1]
+       ldins.b r9:t, r12[0]
+       sub     r12, -4
+
+       add     r10, r9
+       acr     r10
+       sub     r11, 4
+       brge    8b
+
+       rjmp    5b
+#endif /* CONFIG_AVR32_UNALIGNED */
+
        .size   csum_partial, . - csum_partial
--
1.6.2.2
```

D.15 Avoid unaligned access in uaccess.h

```
From 42b25ac9dfc8fd06f193f5d7858518d2f1f5f4b9 Mon Sep 17 00:00:00 2001
Message-Id: <42b25ac9dfc8fd06f193f5d7858518d2f1f5f4b9.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 15:17:24 +0200
Subject: [PATCH 15/29] AVR32: avoid unaligned access in uaccess.h

The patch fixes __get_user_check by calling copy_from_user if the
pointer is unaligned. Note that there are three more macros that needs
to be changed: get_user_nocheck, put_user_check and put_user_nocheck.

This patch really needs a better solution that doesn't involve calling
copy_from_user or copy_to_user.
---
 arch/avr32/include/asm/uaccess.h |   17 +++++++++++++++-
 1 files changed, 16 insertions(+), 1 deletions(-)

diff --git a/arch/avr32/include/asm/uaccess.h b/arch/avr32/include/asm/uaccess.h
index ed09239..99652f2 100644
--- a/arch/avr32/include/asm/uaccess.h
+++ b/arch/avr32/include/asm/uaccess.h
@@ -179,6 +179,13 @@ static inline __kernel_size_t __copy_from_user(void *to,
 extern int __get_user_bad(void);
 extern int __put_user_bad(void);

+/* We need a simple way to test this flag in the following macros. */
+#ifdef CONFIG_AVR32_UNALIGNED
+#define AVR32_UNALIGNED 1
+#else
+#define AVR32_UNALIGNED 0
+#endif
+
 #define __get_user_nocheck(x, ptr, size)                                \
 ({                                                                      \
        unsigned long __gu_val = 0;                                     \
@@ -201,7 +208,15 @@ extern int __put_user_bad(void);
        const typeof(*(ptr)) __user * __gu_addr = (ptr);                \
        int __gu_err = 0;                                               \
                                                                        \
-       if (access_ok(VERIFY_READ, __gu_addr, size)) {                  \
+       if (!AVR32_UNALIGNED && (unsigned long)__gu_addr % (size)) {    \
+               unsigned long count;                                    \
+               count = copy_from_user(&__gu_val, __gu_addr, size);     \
+               if (count == size) {                                    \
+                       __gu_val >>= 8 * (4 - (size));                  \
+               } else {                                                \
+                       __gu_err = -EFAULT;                             \
+               }                                                       \
+       } else if (access_ok(VERIFY_READ, __gu_addr, size)) {           \
                switch (size) {                                         \
                case 1:                                                 \
                        __get_user_asm("ub", __gu_val, __gu_addr,       \
--
1.6.2.2
```

D.16 memcpy for !CONFIG_NOUNALIGNED

```
From 280937f4f84e0f11e0e50a759211503824730b05 Mon Sep 17 00:00:00 2001
Message-Id: <280937f4f84e0f11e0e50a759211503824730b05.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 15:28:14 +0200
Subject: [PATCH 16/29] AVR32: memcpy implementation for chips that cannot do unaligned memory accesses.

---
 arch/avr32/lib/Makefile             |    3 +-
 arch/avr32/lib/memcpy-nounaligned.S |   86 +++++++++++++++++++++++++++++++++++
 2 files changed, 88 insertions(+), 1 deletions(-)
 create mode 100644 arch/avr32/lib/memcpy-nounaligned.S

diff --git a/arch/avr32/lib/Makefile b/arch/avr32/lib/Makefile
index be35b6a..faa63ea 100644
--- a/arch/avr32/lib/Makefile
+++ b/arch/avr32/lib/Makefile
@@ -4,10 +4,11 @@

 lib-y  := clear_user.o
 lib-y  += strncpy_from_user.o strnlen_user.o
-lib-y  += delay.o memset.o memcpy.o findbit.o
+lib-y  += delay.o memset.o findbit.o
 lib-y  += csum_partial.o csum_partial_copy_generic.o
 lib-y  += io-readsw.o io-readsl.o io-writesw.o io-writesl.o
 lib-y  += io-readsb.o io-writesb.o
 lib-y  += __avr32_lsl64.o __avr32_lsr64.o __avr32_asr64.o
 lib-y  += copy_user$(ALIGNEXT).o
+lib-y  += memcpy$(ALIGNEXT).o

diff --git a/arch/avr32/lib/memcpy-nounaligned.S b/arch/avr32/lib/memcpy-nounaligned.S
new file mode 100644
index 0000000..c10fcde
--- /dev/null
+++ b/arch/avr32/lib/memcpy-nounaligned.S
@@ -0,0 +1,86 @@
+/*
+ * Copyright (C) 2004-2006 Atmel Corporation
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the Free Software Foundation.
+ */
+
+       /*
+        * void *memcpy(void *to, const void *from, unsigned long n)
+        *
+        * This implementation does word-aligned loads and stores if possible,
+        * and falls back to byte-copy if not.
+        *
+        * Hopefully, in most cases, both "to" and "from" will be
+        * word-aligned to begin with.
+        */
+       .text
+       .global memcpy
+       .type   memcpy, @function
+memcpy:
+       /*
+        * Check alignedness of "from" and "to". Three possibilities:
+        * - Both are aligned on a word boundary.
+        * - Both can be aligned on a word boundary.
+        * - Not possible to align both on a word boundary.
+        */
+       mov     r8, r12
+       andl    r8, 3, COH
+       mov     r9, r11
+       andl    r9, 3, COH
+
+       /* Is it impossible to align both? */
+       cp.w    r8, r9
+       brne    6f
+
+       /* Do they need alignment? */
+       cp.w    r8, 0
+       brne    1f
+
+       /* At this point, "from" and "to" are word-aligned */
+2:     sub     r10, 4
+       mov     r9, r12
+       brlt    4f
```

```
+
+3:     ld.w    r8, r11++
+       sub     r10, 4
+       st.w    r12++, r8
+       brge    3b
+
+4:     neg     r10
+       reteq   r9
+
+       /* Handle unaligned count */
+       lsl     r10, 2
+       add     pc, pc, r10
+       ld.ub   r8, r11++
+       st.b    r12++, r8
+       ld.ub   r8, r11++
+       st.b    r12++, r8
+       ld.ub   r8, r11++
+       st.b    r12++, r8
+       retal   r9
+
+       /* Handle unaligned "from" and "to" pointer */
+1:     sub     r10, 4
+       brlt    4b
+       add     r10, r9
+       lsl     r9, 2
+       add     pc, pc, r9
+       ld.ub   r8, r11++
+       st.b    r12++, r8
+       ld.ub   r8, r11++
+       st.b    r12++, r8
+       ld.ub   r8, r11++
+       st.b    r12++, r8
+       rjmp    2b
+
+6:     /* Impossible to align both "from" and "to" on a word boundary */
+       mov     r9, r12
+       cp.w    r10, 0
+7:     reteq   r9
+       ld.ub   r8, r11++
+       st.b    r12++, r8
+       sub     r10, 1
+       rjmp    7b
--
1.6.2.2
```

D.17 Mark AVR32B code with subarch flag

```
From 7985d3c97d2a55d4457b5ebcee832d68d05bada7 Mon Sep 17 00:00:00 2001
Message-Id: <7985d3c97d2a55d4457b5ebcee832d68d05bada7.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 15:42:09 +0200
Subject: [PATCH 17/29] AVR32: Mark AVR32B specific assumptions with CONFIG_SUBARCH_AVR32B in strnlen.

---
 arch/avr32/lib/strnlen_user.S |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)

diff --git a/arch/avr32/lib/strnlen_user.S b/arch/avr32/lib/strnlen_user.S
index 65ce11a..482f967 100644
--- a/arch/avr32/lib/strnlen_user.S
+++ b/arch/avr32/lib/strnlen_user.S
@@ -18,10 +18,12 @@
        .type   strnlen_user, "function"
 strnlen_user:
        branch_if_kernel r8, __strnlen_user
+#ifdef CONFIG_SUBARCH_AVR32B
        sub     r8, r11, 1
        add     r8, r12
        retcs   0
        brmi    adjust_length   /* do a closer inspection */
+#endif  /* CONFIG_SUBARCH_AVR32B */

        .global __strnlen_user
        .type   __strnlen_user, "function"
@@ -39,6 +41,7 @@ __strnlen_user:
        retal   r12

```

```
33
34 +#ifdef CONFIG_SUBARCH_AVR32B
35          .type   adjust_length, "function"
36  adjust_length:
37          cp.w    r12, 0          /* addr must always be < TASK_SIZE */
38 @@ -57,6 +60,7 @@ adjust_length:
39          .align  2
40  _task_size:
41          .long   TASK_SIZE
42 +#endif  /* CONFIG_SUBARCH_AVR32B */
43
44          .section .fixup, "ax"
45          .align  1
46 --
47 1.6.2.2
```

D.18 mm-dma-coherent.c: ifdef AVR32B code

```
1  From 8e753b2595e893fa0eec74756ef4458adeaa3caa Mon Sep 17 00:00:00 2001
2  Message-Id: <8e753b2595e893fa0eec74756ef4458adeaa3caa.1242388774.git.rangoy@mnops.(none)>
3  In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
4  References: <cover.1242388773.git.rangoy@mnops.(none)>
5  From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
6  Date: Mon, 27 Apr 2009 13:00:33 +0200
7  Subject: [PATCH 18/29] AVR32: mm/dma-coherent.c - ifdef AVR32B specific code.
8
9  ---
10  arch/avr32/mm/dma-coherent.c |    2 ++
11  1 files changed, 2 insertions(+), 0 deletions(-)
12
13 diff --git a/arch/avr32/mm/dma-coherent.c b/arch/avr32/mm/dma-coherent.c
14 index 6d8c794..99e0b95 100644
15 --- a/arch/avr32/mm/dma-coherent.c
16 +++ b/arch/avr32/mm/dma-coherent.c
17 @@ -13,11 +13,13 @@
18
19  void dma_cache_sync(struct device *dev, void *vaddr, size_t size, int direction)
20  {
21 +#ifdef CONFIG_SUBARCH_AVR32B
22          /*
23           * No need to sync an uncached area
24           */
25          if (PXSEG(vaddr) == P2SEG)
26                  return;
27 +#endif /* CONFIG_SUBARCH_AVR32B */
28
29          switch (direction) {
30          case DMA_FROM_DEVICE:           /* invalidate only */
31 --
32 1.6.2.2
```

D.19 Disable ret_if_privileged macro

```
1  From 6b69cb847f4fa8a7fd8a381d13cbada309823701 Mon Sep 17 00:00:00 2001
2  Message-Id: <6b69cb847f4fa8a7fd8a381d13cbada309823701.1242388774.git.rangoy@mnops.(none)>
3  In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
4  References: <cover.1242388773.git.rangoy@mnops.(none)>
5  From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
6  Date: Thu, 23 Apr 2009 15:25:02 +0200
7  Subject: [PATCH 19/29] AVR32: Disable ret_if_privileged macro for !CONFIG_SUBARCH_AVR32B.
8
9  ---
10  arch/avr32/include/asm/asm.h |    2 ++
11  1 files changed, 2 insertions(+), 0 deletions(-)
12
13 diff --git a/arch/avr32/include/asm/asm.h b/arch/avr32/include/asm/asm.h
14 index a2c64f4..1bad0c5 100644
15 --- a/arch/avr32/include/asm/asm.h
16 +++ b/arch/avr32/include/asm/asm.h
17 @@ -93,10 +93,12 @@
18          .endm
19
20          .macro ret_if_privileged scratch, addr, size, ret
21 +#ifdef CONFIG_SUBARCH_AVR32B
```

```
22          sub     \scratch, \size, 1
23          add     \scratch, \addr
24          retcs   \ret
25          retmi   \ret
26 +#endif /* CONFIG_SUBARCH_AVR32B */
27          .endm
28
29  #endif /* __ASM_AVR32_ASM_H__ */
30 --
31 1.6.2.2
```

D.20 AVR32A-support in Kconfig

```
1  From 585c7e18290ffa6f4ca7d436b41e12f5ba100cd6 Mon Sep 17 00:00:00 2001
2  Message-Id: <585c7e18290ffa6f4ca7d436b41e12f5ba100cd6.1242388774.git.rangoy@mnops.(none)>
3  In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
4  References: <cover.1242388773.git.rangoy@mnops.(none)>
5  From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
6  Date: Fri, 24 Apr 2009 14:37:19 +0200
7  Subject: [PATCH 20/29] AVR32: AVR32A support in Kconfig
8
9  ---
10  arch/avr32/Kconfig |    3 +++
11  1 files changed, 3 insertions(+), 0 deletions(-)
12
13 diff --git a/arch/avr32/Kconfig b/arch/avr32/Kconfig
14 index 9e984b0..e3f6653 100644
15 --- a/arch/avr32/Kconfig
16 +++ b/arch/avr32/Kconfig
17 @@ -81,6 +81,9 @@ source "kernel/time/Kconfig"
18  config MMU
19          bool
20
21 +config SUBARCH_AVR32A
22 +        bool
23 +
24  config SUBARCH_AVR32B
25          bool
26          select MMU
27 --
28 1.6.2.2
```

D.21 AVR32A address space support

```
1  From f355d930aad8d21463b119fcfe6e1d6a3717d8de Mon Sep 17 00:00:00 2001
2  Message-Id: <f355d930aad8d21463b119fcfe6e1d6a3717d8de.1242388774.git.rangoy@mnops.(none)>
3  In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
4  References: <cover.1242388773.git.rangoy@mnops.(none)>
5  From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
6  Date: Thu, 23 Apr 2009 15:14:40 +0200
7  Subject: [PATCH 21/29] AVR32: AVR32A address space support.
8
9  ---
10  arch/avr32/include/asm/addrspace.h |   12 ++++++++++--
11  arch/avr32/include/asm/io.h        |   29 +++++++++++++++++++++++++++++
12  arch/avr32/include/asm/page.h      |    7 +++++++
13  3 files changed, 46 insertions(+), 2 deletions(-)
14
15 diff --git a/arch/avr32/include/asm/addrspace.h b/arch/avr32/include/asm/addrspace.h
16 index 3667948..45e1083 100644
17 --- a/arch/avr32/include/asm/addrspace.h
18 +++ b/arch/avr32/include/asm/addrspace.h
19 @@ -11,7 +11,11 @@
20  #ifndef __ASM_AVR32_ADDRSPACE_H
21  #define __ASM_AVR32_ADDRSPACE_H
22
23 -#ifdef CONFIG_MMU
24 +#ifdef CONFIG_SUBARCH_AVR32A
25 +
26 +#define PHYSADDR(a)     ((unsigned long)(a))
27 +
28 +#elif CONFIG_SUBARCH_AVR32B
29
30  /* Memory segments when segmentation is enabled */
```

```
 #define POSEG           0x00000000
@@ -38,6 +42,10 @@
 #define P4SEGADDR(a) ((__typeof__(a))(((unsigned long)(a) & 0x1fffffff) \
                                       | P4SEG))

-#endif /* CONFIG_MMU */
+#else
+
+#error Unknown AVR32 subarch.
+
+#endif /* CONFIG_SUBARCH_* */

 #endif /* __ASM_AVR32_ADDRSPACE_H */
diff --git a/arch/avr32/include/asm/io.h b/arch/avr32/include/asm/io.h
index 22c97ef..96a81b1 100644
--- a/arch/avr32/include/asm/io.h
+++ b/arch/avr32/include/asm/io.h
@@ -10,6 +10,27 @@

 #include <mach/io.h>

+
+#ifdef CONFIG_SUBARCH_AVR32A
+
+static __inline__ unsigned long virt_to_phys(volatile void *address)
+{
+       return (unsigned long)address;
+}
+
+static __inline__ void * phys_to_virt(unsigned long address)
+{
+       return (void *)address;
+}
+
+#define cached_to_phys(addr)   ((unsigned long)(addr))
+#define uncached_to_phys(addr) ((unsigned long)(addr))
+#define phys_to_cached(addr)   ((void *)(addr))
+#define phys_to_uncached(addr) ((void *)(addr))
+
+
+#elif CONFIG_SUBARCH_AVR32B
+
 /* virt_to_phys will only work when address is in P1 or P2 */
 static __inline__ unsigned long virt_to_phys(volatile void *address)
 {
@@ -26,6 +47,14 @@ static __inline__ void * phys_to_virt(unsigned long address)
 #define phys_to_cached(addr)   ((void *)P1SEGADDR(addr))
 #define phys_to_uncached(addr) ((void *)P2SEGADDR(addr))

+
+#else /* CONFIG_SUBARCH_* */
+
+#error Unknown AVR32 subarch.
+
+#endif /* CONFIG_SUBARCH_* */
+
+
 /*
  * Generic IO read/write.  These perform native-endian accesses.  Note
  * that some architectures will want to re-define __raw_{read,write}w.
diff --git a/arch/avr32/include/asm/page.h b/arch/avr32/include/asm/page.h
index f805d1c..ca36368 100644
--- a/arch/avr32/include/asm/page.h
+++ b/arch/avr32/include/asm/page.h
@@ -74,7 +74,14 @@ static inline int get_order(unsigned long size)
  * What's the difference between __pa() and virt_to_phys() anyway?
  */
 #define __pa(x)                 PHYSADDR(x)
+
+#ifdef CONFIG_SUBARCH_AVR32A
+#define __va(x)                 ((void *)(x))
+#elif CONFIG_SUBARCH_AVR32B
 #define __va(x)                 ((void *)(P1SEGADDR(x)))
+#else /* CONFIG_SUBARCH_* */
+#error Unknown AVR32 subarch.
+#endif /* CONFIG_SUBARCH_* */

 #define MAP_NR(addr)    (((unsigned long)(addr) - PAGE_OFFSET) >> PAGE_SHIFT)

--
1.6.2.2
```

D.22 Change maximum task size for AVR32A

```
From cfe6bfd67af7f4caf10f73fc176a160b87da8bb8 Mon Sep 17 00:00:00 2001
Message-Id: <cfe6bfd67af7f4caf10f73fc176a160b87da8bb8.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 14:14:44 +0200
Subject: [PATCH 22/29] AVR32: Change maximum task size for AVR32A

---
 arch/avr32/include/asm/processor.h |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)

diff --git a/arch/avr32/include/asm/processor.h b/arch/avr32/include/asm/processor.h
index 3fb964d..843d7a3 100644
--- a/arch/avr32/include/asm/processor.h
+++ b/arch/avr32/include/asm/processor.h
@@ -11,7 +11,11 @@
 #include <asm/page.h>
 #include <asm/cache.h>

+#ifdef CONFIG_SUBARCH_AVR32A
+#define TASK_SIZE      0xffffffff
+#else
 #define TASK_SIZE      0x80000000
+#endif

 #ifdef __KERNEL__
 #define STACK_TOP      TASK_SIZE
--
1.6.2.2
```

D.23 Fix ___range_ok for AVR32A in uaccess.h

```
From 1894ee64853872a75c0f5f52029ad31f7208db3d Mon Sep 17 00:00:00 2001
Message-Id: <1894ee64853872a75c0f5f52029ad31f7208db3d.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 14:52:15 +0200
Subject: [PATCH 23/29] AVR32: Fix uaccess __range_ok macro for AVR32A.

---
 arch/avr32/include/asm/uaccess.h |   14 ++++++++++++-
 1 files changed, 13 insertions(+), 1 deletions(-)

diff --git a/arch/avr32/include/asm/uaccess.h b/arch/avr32/include/asm/uaccess.h
index 99652f2..6156289 100644
--- a/arch/avr32/include/asm/uaccess.h
+++ b/arch/avr32/include/asm/uaccess.h
@@ -51,7 +51,16 @@ static inline void set_fs(mm_segment_t s)
 /*
  * Test whether a block of memory is a valid user space address.
  * Returns 0 if the range is valid, nonzero otherwise.
- *
+ */
+#ifdef CONFIG_SUBARCH_AVR32A
+/*
+ * No easy check for user space address possible, but we don't have
+ * very much protection in any case since we don't have an MMU.
+ */
+#define __range_ok(addr, size) 0
+
+#elif CONFIG_SUBARCH_AVR32B
+/*
  * We do the following checks:
  *   1. Is the access from kernel space?
  *   2. Does (addr + size) set the carry bit?
@@ -65,6 +74,9 @@ static inline void set_fs(mm_segment_t s)
         && (((unsigned long)(addr) >= 0x80000000)                       \
             || ((unsigned long)(size) > 0x80000000)                     \
             || (((unsigned long)(addr) + (unsigned long)(size)) > 0x80000000)))
+#else
+#error Unknown AVR32 subarch.
+#endif /* CONFIG_SUBARCH_* */

 #define access_ok(type, addr, size) (likely(__range_ok(addr, size) == 0))
```

```
44
45 --
46 1.6.2.2
```

D.24 Support for AVR32A entry-avr32a.S

```
1  From 1f99f4536db8830ab1817ad460627d14d4de5d2d Mon Sep 17 00:00:00 2001
2  Message-Id: <1f99f4536db8830ab1817ad460627d14d4de5d2d.1242388774.git.rangoy@mnops.(none)>
3  In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
4  References: <cover.1242388773.git.rangoy@mnops.(none)>
5  From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
6  Date: Fri, 24 Apr 2009 15:03:39 +0200
7  Subject: [PATCH 24/29] AVR32: Support for AVR32A (entry-avr32a.c)
8
9  ---
10  arch/avr32/kernel/Makefile       |    1 +
11  arch/avr32/kernel/entry-avr32a.S |  705 ++++++++++++++++++++++++++++++++++++++
12  2 files changed, 706 insertions(+), 0 deletions(-)
13  create mode 100644 arch/avr32/kernel/entry-avr32a.S
14
15 diff --git a/arch/avr32/kernel/Makefile b/arch/avr32/kernel/Makefile
16 index 18229d0..76adcd2 100644
17 --- a/arch/avr32/kernel/Makefile
18 +++ b/arch/avr32/kernel/Makefile
19 @@ -4,6 +4,7 @@
20
21  extra-y                                 := head.o vmlinux.lds
22
23 +obj-$(CONFIG_SUBARCH_AVR32A)    += entry-avr32a.o
24  obj-$(CONFIG_SUBARCH_AVR32B)    += entry-avr32b.o
25  obj-y                           += syscall_table.o syscall-stubs.o irq.o
26  obj-y                           += setup.o traps.o ocd.o ptrace.o
27 diff --git a/arch/avr32/kernel/entry-avr32a.S b/arch/avr32/kernel/entry-avr32a.S
28 new file mode 100644
29 index 0000000..2b97739
30 --- /dev/null
31 +++ b/arch/avr32/kernel/entry-avr32a.S
32 @@ -0,0 +1,705 @@
33 +/*
34 + * Copyright (C) 2004-2006 Atmel Corporation
35 + *
36 + * This program is free software; you can redistribute it and/or modify
37 + * it under the terms of the GNU General Public License version 2 as
38 + * published by the Free Software Foundation.
39 + */
40 +
41 +/*
42 + * This file contains the low-level entry-points into the kernel, that is,
43 + * exception handlers, debug trap handlers, interrupt handlers and the
44 + * system call handler.
45 + */
46 +#include <linux/errno.h>
47 +
48 +#include <asm/asm.h>
49 +#include <asm/hardirq.h>
50 +#include <asm/irq.h>
51 +#include <asm/ocd.h>
52 +#include <asm/page.h>
53 +#include <asm/pgtable.h>
54 +#include <asm/ptrace.h>
55 +#include <asm/sysreg.h>
56 +#include <asm/thread_info.h>
57 +#include <asm/unistd.h>
58 +
59 +       .section .ex.text,"ax",@progbits
60 +       .align  2
61 +exception_vectors:                      /* (EVBA) Name, Event source */
62 +       bral    handle_critical         /* (0x00) Unrecoverable exception, Internal */
63 +       .align  2
64 +       bral    handle_critical         /* (0x04) TLB Multiple hit, Internal Signal */
65 +       .align  2
66 +       bral    do_bus_error_write      /* (0x08) Bus error data fetch, Data bus */
67 +       .align  2
68 +       bral    do_bus_error_read       /* (0x0c) Bus error instruction fetch, Data bus */
69 +       .align  2
70 +       bral    do_nmi_ll               /* (0x10) NMI (Non Maskable Interrupt), External input */
71 +       .align  2
72 +       bral    handle_address_fault    /* (0x14) Instruction address, ITLB */
73 +       .align  2
```

```
+       bral    handle_protection_fault /* (0x18) ITLB Protection, ITLB */
+       .align  2
+       bral    handle_debug            /* (0x1c) Breakpoint, OCD system */
+       .align  2
+       bral    do_illegal_opcode_ll    /* (0x20) Illegal opcode, Instruction */
+       .align  2
+       bral    do_illegal_opcode_ll    /* (0x24) Unimplmented instruction, Instruction */
+       .align  2
+       bral    do_illegal_opcode_ll    /* (0x28) Privilege violation, Instruction */
+       .align  2
+       bral    do_fpe_ll               /* (0x2c) Floating-point, FP Hardware */
+       .align  2
+       bral    do_illegal_opcode_ll    /* (0x30) Coprocessor absent, Insctruction */
+       .align  2
+       bral    handle_address_fault    /* (0x34) Data address (Read), DTLB */
+       .align  2
+       bral    handle_address_fault    /* (0x38) Data address (Write), DTLB */
+       .align  2
+       bral    handle_protection_fault /* (0x3c) DTLB Protection (Read), DTLB */
+       .align  2
+       bral    handle_protection_fault /* (0x40) DTLB Protection (Write), DTLB */
+       .align  2
+       bral    do_dtlb_modified        /* (0x44) DTLB Modified, DTLB */
+
+
+       .org    0x50                    /* (0x50) ITLB Miss, ITLB */
+       .global itlb_miss
+itlb_miss:
+       rjmp    tlb_miss_common
+
+       .org    0x60                    /* (0x60) DTLB Miss (Read), DTLB */
+dtlb_miss_read:
+       rjmp    tlb_miss_common
+
+       .org    0x70                    /* (0x70) DTLB Miss (write), DTLB */
+dtlb_miss_write:
+
+       .global tlb_miss_common
+       .align  2
+tlb_miss_common:
+       /* this should never be called... */
+       sub     r12, pc, (. - 1f)
+       bral    panic
+       .align  2
+1:     .asciz  "tlb_miss_common..."
+
+       /* ---                    System Call                    --- */
+
+       .org    0x100                   /* (0x100) Superviser call, Instruction */
+system_call:
+       stmts   --sp, r0-lr
+       pushm   r12             /* r12_orig */
+
+       zero_fp /* [remove comment (RC) ] sets frame pointer[R7] to zero to ensure that the frame pointer,
+               so that the backtrace does not follow a context switch */
+
+       /* check for syscall tracing */
+       get_thread_info r0
+       ld.w    r1, r0[TI_flags] /* RC: load TI_flags to r1 */
+       bld     r1, TIF_SYSCALL_TRACE /* RC: Set carry flag if TIF_SYSCALL_TRACE is set in thread_info */
+       brcs    syscall_trace_enter /* RC:branch if ^ */
+
+syscall_trace_cont:
+       cp.w    r8, NR_syscalls
+       brhs    syscall_badsys /* RC: branch if system call is out of range */
+
+       lddpc   lr, syscall_table_addr /* set lr to syscall base address */
+       ld.w    lr, lr[r8 << 2] /* RC: fetch the syscall address from syscall address based on the syscall number (R8) to lr */
+       mov     r8, r5          /* 5th argument (6th is pushed by stub) */
+       icall   lr              /* call syscall handeling*/
+
+       .global syscall_return
+syscall_return:
+       get_thread_info r0
+       mask_interrupts         /* make sure we don't miss an interrupt
+                                  setting need_resched or sigpending
+                                  between sampling and the rets */
+
+       /* Store the return value so that the correct value is loaded below */
+       stdsp   sp[REG_R12], r12
+
```

```
155 +       ld.w    r1, r0[TI_flags]
156 +       andl    r1, _TIF_ALLWORK_MASK, COH
157 +       brne    syscall_exit_work       /* RC: branch if work has to be done */
158 +
159 +syscall_exit_cont:
160 +       sub     sp, -4          /* r12_orig */
161 +       ldmts   sp++, r0-lr     /* restoring registers */
162 +       rets
163 +
164 +       .align  2
165 +syscall_table_addr:
166 +       .long   sys_call_table
167 +
168 +syscall_badsys: /* RC: comefrom: syscall_trace_cont */
169 +       mov     r12, -ENOSYS
170 +       rjmp    syscall_return /* RC: return -ENOSYS */
171 +
172 +syscall_trace_enter:
173 +       pushm   r8-r12
174 +       rcall   syscall_trace
175 +       popm    r8-r12
176 +       rjmp    syscall_trace_cont
177 +
178 +       .global ret_from_fork
179 +ret_from_fork: /* RC: a newborn child starts it's exciting new thread here */
180 +       rcall   schedule_tail
181 +
182 +       /* check for syscall tracing */
183 +       get_thread_info r0
184 +       ld.w    r1, r0[TI_flags]
185 +       andl    r1, _TIF_ALLWORK_MASK, COH
186 +       breq    syscall_exit_cont
187 +       /*
188 +        * Fall through to syscall_exit_work since one or more of the
189 +        * bits in TIF_ALLWORK_MASK was set.
190 +        */
191 +
192 +syscall_exit_work:
193 +       bld     r1, TIF_SYSCALL_TRACE
194 +       brcc    syscall_exit_work_loop
195 +       unmask_interrupts
196 +       rcall   syscall_trace
197 +       mask_interrupts
198 +       ld.w    r1, r0[TI_flags]
199 +
200 +       /*
201 +        * This loop will run until no work-flags are set in the
202 +        * thread info.
203 +        */
204 +syscall_exit_work_loop:
205 +       bld     r1, TIF_NEED_RESCHED
206 +       brcc    syscall_exit_work_nosched
207 +       unmask_interrupts
208 +       rcall   schedule
209 +       mask_interrupts
210 +       ld.w    r1, r0[TI_flags]
211 +       rjmp    syscall_exit_work_loop
212 +
213 +syscall_exit_work_nosched:
214 +       mov     r2, _TIF_SIGPENDING | _TIF_RESTORE_SIGMASK
215 +       tst     r1, r2
216 +       breq    syscall_exit_work_nosigs
217 +       unmask_interrupts
218 +       mov     r12, sp
219 +       mov     r11, r0
220 +       rcall   do_notify_resume
221 +       mask_interrupts
222 +       ld.w    r1, r0[TI_flags]
223 +       rjmp    syscall_exit_work_loop
224 +
225 +syscall_exit_work_nosigs:
226 +       bld     r1, TIF_BREAKPOINT
227 +       brcc    syscall_exit_cont
228 +       rjmp    enter_monitor_mode
229 +
230 +
231 +       .type   save_full_context_ex, @function
232 +       .align  2
233 +save_full_context_ex:
234 +       /*
235 +        * Check whether the return address of the exception is the
236 +        * debug_trampoline, since that would need special handling.
237 +        */
238 +       lddsp   r11, sp[REG_PC]
```

```
+       sub     r9, pc, . - debug_trampoline
+       cp.w    r9, r11
+       breq    save_full_context_dbg_tramp
+
+       /* Check for kernel-mode. */
+       lddsp   r8, sp[REG_SR]
+       mov     r12, r8
+       andh    r8, (MODE_MASK >> 16), COH
+       brne    save_full_context_kernel_mode
+
+save_full_context_done:
+       unmask_exceptions
+       ret     r12
+
+save_full_context_kernel_mode:
+       sub     r10, sp, -FRAME_SIZE_FULL
+       stdsp   sp[REG_SP], r10 /* replace saved SP with kernel-mode SP */
+       rjmp    save_full_context_done
+
+       /*
+        * The debug handler set up a trampoline to make us
+        * automatically enter monitor mode upon return, but since
+        * we're saving the full context, we must assume that the
+        * exception handler might want to alter the return address
+        * and/or status register. So we need to restore the original
+        * context and enter monitor mode manually after the exception
+        * has been handled.
+        */
+save_full_context_dbg_tramp:
+       get_thread_info r8
+       ld.w    r11, r8[TI_rar_saved]
+       ld.w    r12, r8[TI_rsr_saved]
+       stdsp   sp[REG_PC], r11
+       stdsp   sp[REG_SR], r12
+
+       rjmp    save_full_context_done
+       .size   save_full_context_ex, . - save_full_context_ex
+
+       /* Low-level exception handlers */
+handle_critical:
+       pushm   r0-r12
+       sub     sp, 12  /* lr, sp, r12_orig */
+
+       mfsr    r12, SYSREG_ECR
+       mov     r11, sp
+       rcall   do_critical_exception
+
+       /* We should never get here... */
+       sub     r12, pc, (. - 1f)
+       bral    panic
+       .align  2
+1:     .asciz  "Return from critical exception!"
+
+       .align  1
+do_bus_error_write:
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+       rcall   save_full_context_ex
+       mov     r11, 1
+       rjmp    do_bus_error_common
+
+do_bus_error_read:
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+       rcall   save_full_context_ex
+       mov     r11, 0
+
+do_bus_error_common:
+       mfsr    r12, SYSREG_BEAR
+       mov     r10, sp
+       rcall   do_bus_error
+       rjmp    ret_from_exception
+
+       .align  1
+do_nmi_ll:
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+
+       /* Check for kernel-mode. */
+       lddsp   r9, sp[REG_SR]
+       bfextu  r0, r9, MODE_SHIFT, 3
+       brne    do_nmi_ll_kernel_fixup
+
+do_nmi_ll_cont:
```

```
+       mfsr    r12, SYSREG_ECR
+       mov     r11, sp
+       rcall   do_nmi
+       tst     r0, r0
+       brne    do_nmi_ll_kernel_exit
+
+       sub     sp, -4          /* skip r12_orig */
+       ldmts   sp++, r0-lr
+       rete
+
+       /* Kernel mode save */
+do_nmi_ll_kernel_fixup:
+       sub     r10, sp, -FRAME_SIZE_FULL
+       stdsp   sp[REG_SP], r10 /* replace saved SP */
+       rjmp    do_nmi_ll_cont
+
+       /* Kernel mode restore */
+do_nmi_ll_kernel_exit:
+       sub     sp, -4          /* skip r12_orig */
+       popm    lr
+       sub     sp, -4          /* skip sp */
+       popm    r0-r12
+       rete
+
+
+handle_address_fault:
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+       rcall   save_full_context_ex
+       mfsr    r12, SYSREG_ECR
+       mov     r11, sp
+       rcall   do_address_exception
+       rjmp    ret_from_exception
+
+
+handle_protection_fault:
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+       rcall   save_full_context_ex
+       mfsr    r12, SYSREG_ECR
+       mov     r11, sp
+       rcall   do_page_fault
+       rjmp    ret_from_exception
+
+
+       .align  1
+do_illegal_opcode_ll:
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+       rcall   save_full_context_ex
+       mfsr    r12, SYSREG_ECR
+       mov     r11, sp
+       rcall   do_illegal_opcode
+       rjmp    ret_from_exception
+
+
+do_dtlb_modified:
+       sub     r12, pc, (. - 1f)
+       bral    panic
+       .align 2
+1:     .asciz  "do_dtlb_modified"
+
+
+       .align 1
+do_fpe_ll:
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+       rcall   save_full_context_ex
+       unmask_interrupts
+       mov     r12, 26 /* TODO: this should probably be 11 (0x2C/4) */
+       mov     r11, sp
+       rcall   do_fpe
+       rjmp    ret_from_exception
+
+
+       /* Common code for returning from an exception handler. */
+ret_from_exception:
+       mask_interrupts
+       lddsp   r4, sp[REG_SR]
+       andh    r4, (MODE_MASK >> 16), COH
+       brne    fault_resume_kernel
+
+       get_thread_info r0
+       ld.w    r1, r0[TI_flags]
```

```
+       andl    r1, _TIF_WORK_MASK, COH
+       brne    fault_exit_work
+
+fault_resume_user:
+       mask_exceptions
+       sub     sp, -4          /* skip r12_orig */
+       ldmts   sp++, r0-lr
+       rete
+
+fault_resume_kernel:
+#ifdef CONFIG_PREEMPT
+       /* Check whether we should preempt this kernel thread. */
+       get_thread_info r0
+       ld.w    r2, r0[TI_preempt_count]
+       cp.w    r2, 0
+       brne    fault_resume_kernel_no_schedule
+       ld.w    r1, r0[TI_flags]
+       bld     r1, TIF_NEED_RESCHED
+       brcc    fault_resume_kernel_no_schedule
+       lddsp   r4, sp[REG_SR]
+       bld     r4, SYSREG_GM_OFFSET
+       brcs    fault_resume_kernel_no_schedule
+       rcall   preempt_schedule_irq
+fault_resume_kernel_no_schedule:
+#endif
+
+       mask_exceptions
+       sub     sp, -4          /* ignore r12_orig */
+       popm    lr
+       sub     sp, -4          /* ignore SP */
+       popm    r0-r12
+       rete
+
+       /*
+        * Common code for IRQ and exception handlers.
+        * Expects r0 to contain a reference to the thread_info struct,
+        * and r1 to contain TI_flags from the thread_info struct.
+        */
+fault_exit_work:
+       bld     r1, TIF_NEED_RESCHED
+       brcc    fault_exit_work_no_resched
+       unmask_interrupts
+       rcall   schedule
+       mask_interrupts
+       ld.w    r1, r0[TI_flags]
+       rjmp    fault_exit_work
+
+fault_exit_work_no_resched:
+       mov     r2, _TIF_SIGPENDING | _TIF_RESTORE_SIGMASK
+       tst     r1, r2
+       breq    fault_exit_work_no_sigwork
+       unmask_interrupts
+       mov     r12, sp
+       mov     r11, r0
+       rcall   do_notify_resume
+       mask_interrupts
+       ld.w    r1, r0[TI_flags]
+       rjmp    fault_exit_work
+
+fault_exit_work_no_sigwork:
+       bld     r1, TIF_BREAKPOINT
+       brcc    fault_resume_user
+       rjmp    enter_monitor_mode
+
+
+       .section .kprobes.text, "ax", @progbits
+       .type   handle_debug, @function
+handle_debug:
+       sub     sp, 8           /* Make room for REG_PC and REG_SR */
+       stmts   --sp, r0-lr
+       sub     sp, 4           /* skip r12_orig */
+       mfsr    r8, SYSREG_RAR_DBG
+       stdsp   sp[REG_PC], r8
+       mfsr    r9, SYSREG_RSR_DBG
+       stdsp   sp[REG_SR], r9
+       unmask_exceptions
+       bfextu  r9, r9, SYSREG_MODE_OFFSET, SYSREG_MODE_SIZE
+       brne    debug_fixup_regs
+
+.Ldebug_fixup_cont:
+#ifdef CONFIG_TRACE_IRQFLAGS
+       rcall   trace_hardirqs_off
+#endif
+       mov     r12, sp
```

```
491 +       rcall   do_debug
492 +       mov     sp, r12
493 +
494 +       lddsp   r2, sp[REG_SR]
495 +       bfextu  r3, r2, SYSREG_MODE_OFFSET, SYSREG_MODE_SIZE
496 +       brne    debug_resume_kernel
497 +
498 +       get_thread_info r0
499 +       ld.w    r1, r0[TI_flags]
500 +       mov     r2, _TIF_DBGWORK_MASK
501 +       tst     r1, r2
502 +       brne    debug_exit_work
503 +
504 +       bld     r1, TIF_SINGLE_STEP
505 +       brcc    1f
506 +       mfdr    r4, OCD_DC
507 +       sbr     r4, OCD_DC_SS_BIT
508 +       mtdr    OCD_DC, r4
509 +
510 +1:     mask_exceptions
511 +
512 +#ifdef CONFIG_TRACE_IRQFLAGS
513 +       rcall   trace_hardirqs_on
514 +1:
515 +#endif
516 +       sub     sp, -4
517 +       ldmts   sp++, r0-lr
518 +       retd
519 +       .size   handle_debug, . - handle_debug
520 +
521 +       /* Mode of the trapped context is in r9 */
522 +       .type   debug_fixup_regs, @function
523 +debug_fixup_regs:
524 +       sub     r8, sp, -FRAME_SIZE_FULL
525 +       stdsp   sp[REG_SP], r8
526 +       rjmp    .Ldebug_fixup_cont
527 +       .size   debug_fixup_regs, . - debug_fixup_regs
528 +
529 +       .type   debug_resume_kernel, @function
530 +debug_resume_kernel:
531 +       mask_exceptions
532 +#ifdef CONFIG_TRACE_IRQFLAGS
533 +       bld     r11, SYSREG_GM_OFFSET
534 +       brcc    1f
535 +       rcall   trace_hardirqs_on
536 +1:
537 +#endif
538 +       mfsr    r2, SYSREG_SR
539 +       mov     r1, r2
540 +       bfins   r2, r3, SYSREG_MODE_OFFSET, SYSREG_MODE_SIZE
541 +       mtsr    SYSREG_SR, r2
542 +       sub     pc, -2
543 +       mtsr    SYSREG_SR, r1
544 +       sub     pc, -2          /* flush pipeline */
545 +       sub     sp, -4          /* Skip r12_orig */
546 +       popm    lr
547 +       sub     sp, -4          /* skip SP */
548 +       popm    r0-r12
549 +       retd
550 +       .size   debug_resume_kernel, . - debug_resume_kernel
551 +
552 +       .type   debug_exit_work, @function
553 +
554 +/*end of fixups after reg change */
555 +debug_exit_work:
556 +       /*
557 +        * We must return from Monitor Mode using a retd, and we must
558 +        * not schedule since that involves the D bit in SR getting
559 +        * cleared by something other than the debug hardware. This
560 +        * may cause undefined behaviour according to the Architecture
561 +        * manual.
562 +        *
563 +        * So we fix up the return address and status and return to a
564 +        * stub below in Exception mode. From there, we can follow the
565 +        * normal exception return path.
566 +        *
567 +        * The real return address and status registers are stored on
568 +        * the stack in the way the exception return path understands,
569 +        * so no need to fix anything up there.
570 +        */
571 +       sub     r8, pc, . - fault_exit_work
572 +       st.w    sp[REG_PC], r8
573 +       mov     r9, 0
574 +       orh     r9, hi(SR_EM | SR_GM | MODE_EXCEPTION)
```

```
575 +       st.w    sp[REG_SR], r9
576 +       sub     pc, -2
577 +       retd
578 +       .size   debug_exit_work, . - debug_exit_work
579 +
580 +
581 +       .macro  IRQ_LEVEL level
582 +       .type   irq_level\level, @function
583 +irq_level\level:
584 +       /* Stack:
585 +        * sp+0   SR
586 +        * sp+4   PC
587 +        * sp+8   LR
588 +        * sp+12  R12
589 +        * sp+16  R11
590 +        * sp+20  R10
591 +        * sp+24  R9
592 +        * sp+28  R8
593 +        */
594 +       stmts   --sp,r0-lr
595 +       sub     sp, 4           /* skip r12_orig */
596 +       lddsp   r8, sp[REG_PC]
597 +       lddsp   r9, sp[REG_SR]
598 +
599 +       mov     r11, sp
600 +       mov     r12, \level
601 +
602 +       rcall   do_IRQ
603 +
604 +       lddsp   r4, sp[REG_SR]
605 +       bfextu  r4, r4, SYSREG_M0_OFFSET, 3
606 +       cp.w    r4, MODE_SUPERVISOR >> SYSREG_M0_OFFSET
607 +       breq    2f
608 +       cp.w    r4, MODE_USER >> SYSREG_M0_OFFSET
609 +#ifdef CONFIG_PREEMPT
610 +       brne    3f
611 +#else
612 +       brne    1f
613 +#endif
614 +
615 +       /* Interrupt was entered from user-mode. */
616 +       get_thread_info r0
617 +       ld.w    r1, r0[TI_flags]
618 +       mov     r2, r1
619 +       andl    r2, _TIF_WORK_MASK, COH
620 +       brne    fault_exit_work
621 +
622 +       /* Exit interrupt handling. */
623 +1:
624 +#ifdef CONFIG_TRACE_IRQFLAGS
625 +       rcall   trace_hardirqs_on
626 +#endif
627 +       sub     sp, -4          /* ignore r12_orig */
628 +       ldmts   sp++,r0-lr
629 +       rete
630 +
631 +
632 +       /*
633 +        * Interrupt was entered from supervisor mode. We need to check
634 +        * that this didn't happen while the processor was going to
635 +        * sleep. The power-manager will set the CPU_GOING_TO_SLEEP flag
636 +        * when entering sleep mode. We test that flag, and if it is
637 +        * set, we change the return address of the interrupt to the
638 +        * instruction following the sleep-instruction.
639 +        */
640 +2:     get_thread_info r0
641 +       ld.w    r1, r0[TI_flags]
642 +       bld     r1, TIF_CPU_GOING_TO_SLEEP
643 +#ifdef CONFIG_PREEMPT
644 +       brcc    3f
645 +#else
646 +       brcc    1b
647 +#endif
648 +
649 +       /*
650 +        * Update the return address so that the sleep-instruction
651 +        * isn't executed.
652 +        */
653 +       sub     r1, pc, . - cpu_idle_skip_sleep
654 +       stdsp   sp[REG_PC], r1
655 +
656 +#ifdef CONFIG_PREEMPT
657 +       /*
658 +        * When interrupts are entered from kernel mode, and preemption
```

```
+         * is enabled, we need to check whether we should schedule after
+         * executing the interrupt. This is done in this block of code.
+         */
+3:      get_thread_info r0
+        ld.w    r2, r0[TI_preempt_count]
+        cp.w    r2, 0
+        brne    1b
+        ld.w    r1, r0[TI_flags]
+        bld     r1, TIF_NEED_RESCHED
+        brcc    1b
+        lddsp   r4, sp[REG_SR]
+        bld     r4, SYSREG_GM_OFFSET
+        brcs    1b
+        rcall   preempt_schedule_irq
+#endif
+        rjmp    1b
+        .endm
+
+        .section .irq.text,"ax",@progbits
+
+        .global irq_level0
+        .global irq_level1
+        .global irq_level2
+        .global irq_level3
+        IRQ_LEVEL 0
+        IRQ_LEVEL 1
+        IRQ_LEVEL 2
+        IRQ_LEVEL 3
+
+        .section .kprobes.text, "ax", @progbits
+        .type   enter_monitor_mode, @function
+enter_monitor_mode:
+        /*
+         * We need to enter monitor mode to do a single step. The
+         * monitor code will alter the return address so that we
+         * return directly to the user instead of returning here.
+         */
+        breakpoint
+        rjmp    breakpoint_failed
+
+        .size   enter_monitor_mode, . - enter_monitor_mode
+
+        .type   debug_trampoline, @function
+        .global debug_trampoline
+debug_trampoline:
+        /*
+         * Save the registers on the stack so that the monitor code
+         * can find them easily.
+         */
+        stmts   --sp, r0-lr
+        sub     sp, 4           /* skip r12_orig */
+        get_thread_info r0
+        ld.w    r8, r0[TI_rar_saved]
+        ld.w    r9, r0[TI_rsr_saved]
+        stdsp   sp[REG_PC], r8
+        stdsp   sp[REG_SR], r9
+
+        /*
+         * The monitor code will alter the return address so we don't
+         * return here.
+         */
+        breakpoint
+        rjmp    breakpoint_failed
+        .size   debug_trampoline, . - debug_trampoline
+
+        .type breakpoint_failed, @function
+breakpoint_failed:
+        /*
+         * Something went wrong. Perhaps the debug hardware isn't
+         * enabled?
+         */
+        lda.w   r12, msg_breakpoint_failed
+        mov     r11, sp
+        mov     r10, 9          /* SIGKILL */
+        call    die
+1:      rjmp    1b
+
+msg_breakpoint_failed:
+        .asciz  "Failed to enter Debug Mode"
--
1.6.2.2
```

D.25 Change HIMEM_START for AVR32A

```
From 881604261316b978975207e22d919a07924e0927 Mon Sep 17 00:00:00 2001
Message-Id: <881604261316b978975207e22d919a07924e0927.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Fri, 24 Apr 2009 15:21:38 +0200
Subject: [PATCH 25/29] AVR32: Change HIGHMEM_START for AVR32A.

---
 arch/avr32/include/asm/page.h |    6 ++++++
 1 files changed, 6 insertions(+), 0 deletions(-)

diff --git a/arch/avr32/include/asm/page.h b/arch/avr32/include/asm/page.h
index ca36368..b69e6c1 100644
--- a/arch/avr32/include/asm/page.h
+++ b/arch/avr32/include/asm/page.h
@@ -106,6 +106,12 @@ static inline int get_order(unsigned long size)
 /*
  * Memory above this physical address will be considered highmem.
  */
+#ifdef CONFIG_SUBARCH_AVR32A
+#define HIGHMEM_START          0xffffffffUL
+#elif CONFIG_SUBARCH_AVR32B
 #define HIGHMEM_START          0x20000000UL
+#else /* CONFIG_SUBARCH_* */
+#error Unknown AVR32 subarch.
+#endif /* CONFIG_SUBARCH_* */

 #endif /* __ASM_AVR32_PAGE_H */
--
1.6.2.2
```

D.26 New pt_regs layout for AVR32A

```
From 814e2ae4fa15281d76b1ee21e1f565e1880a9698 Mon Sep 17 00:00:00 2001
Message-Id: <814e2ae4fa15281d76b1ee21e1f565e1880a9698.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Mon, 27 Apr 2009 14:49:13 +0200
Subject: [PATCH 26/29] AVR32: New pt_regs layout for AVR32A.

---
 arch/avr32/include/asm/ptrace.h |   79 +++++++++++++++++++++++++++++++++++++++
 1 files changed, 79 insertions(+), 0 deletions(-)

diff --git a/arch/avr32/include/asm/ptrace.h b/arch/avr32/include/asm/ptrace.h
index 9e2d44f..043b873 100644
--- a/arch/avr32/include/asm/ptrace.h
+++ b/arch/avr32/include/asm/ptrace.h
@@ -61,6 +61,41 @@
 #define SR_Z_BIT       1
 #define SR_C_BIT       0

+
+#ifdef __AVR32_AVR32A__
+
+/*
+ * The SR and PC registers are always saved on interrupts and exceptions
+ * on AVR32A, so we give those the hightest addresses. The order of the
+ * others is defined by the stmts instruction. r0 is stored first, so it
+ * gets the highest address.
+ */
+#define REG_R12_ORIG    0
+
+#define REG_LR          4
+#define REG_SP          8
+#define REG_R12                 12
+#define REG_R11                 16
+#define REG_R10                 20
+#define REG_R9          24
+#define REG_R8          28
+#define REG_R7          32
+#define REG_R6          36
+#define REG_R5          40
+#define REG_R4          44
```

```
+#define REG_R3          48
+#define REG_R2          52
+#define REG_R1          56
+#define REG_R0          60
+
+#define REG_SR          64
+#define REG_PC          68
+
+#define FRAME_SIZE_MIN  8
+#define FRAME_SIZE_FULL 72
+
+#elif __AVR32_AVR32B__
+
 /*
  * The order is defined by the stmts instruction. r0 is stored first,
  * so it gets the highest address.
@@ -93,7 +128,45 @@
 #define REG_PC           4
 #define REG_SR           0

+#else /* __AVR32_AVR32*__ */
+
+#error Unknown AVR32 subarch.
+
+#endif /* __AVR32_AVR32*__ */
+
 #ifndef __ASSEMBLY__
+
+#ifdef __AVR32_AVR32A__
+
+struct pt_regs {
+
+       /* Only saved on system call, and is used to restart system calls. */
+       unsigned long r12_orig;
+
+       /* Always saved, but some might be optimized away? */
+       unsigned long lr;
+       unsigned long sp;
+       unsigned long r12;
+       unsigned long r11;
+       unsigned long r10;
+       unsigned long r9;
+       unsigned long r8;
+       unsigned long r7;
+       unsigned long r6;
+       unsigned long r5;
+       unsigned long r4;
+       unsigned long r3;
+       unsigned long r2;
+       unsigned long r1;
+       unsigned long r0;
+
+       /* These are automatically saved when an interrupt or exception occurs */
+       unsigned long sr;
+       unsigned long pc;
+};
+
+#elif __AVR32_AVR32B__
+
 struct pt_regs {
        /* These are always saved */
        unsigned long sr;
@@ -120,6 +193,12 @@ struct pt_regs {
        unsigned long r12_orig;
 };

+#else /* __AVR32_AVR32*__ */
+
+#error Unknown AVR32 subarch.
+
+#endif /* __AVR32_AVR32*__ */
+
 #ifdef __KERNEL__

 #include <asm/ocd.h>
--
1.6.2.2
```

D.27 UC3A0512ES interrupt bug workaround

```
From e836ea71931e9bb5a4caf6066d59785823bae32b Mon Sep 17 00:00:00 2001
Message-Id: <e836ea71931e9bb5a4caf6066d59785823bae32b.1242388774.git.rangoy@mnops.(none)>
In-Reply-To: <cover.1242388773.git.rangoy@mnops.(none)>
References: <cover.1242388773.git.rangoy@mnops.(none)>
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <gunnar@rangoy.com>
Date: Thu, 23 Apr 2009 15:04:08 +0200
Subject: [PATCH 27/29] AVR32: UC3A0512ES Interrupt bug workaround

---
 arch/avr32/include/asm/asm.h      |   26 ++++++++++++++++++++++--
 arch/avr32/include/asm/irqflags.h |    8 ++++++++
 2 files changed, 32 insertions(+), 2 deletions(-)

diff --git a/arch/avr32/include/asm/asm.h b/arch/avr32/include/asm/asm.h
index 1bad0c5..20f737b 100644
--- a/arch/avr32/include/asm/asm.h
+++ b/arch/avr32/include/asm/asm.h
@@ -12,8 +12,30 @@
 #include <asm/asm-offsets.h>
 #include <asm/thread_info.h>

-#define mask_interrupts                 ssrf    SYSREG_GM_OFFSET
-#define mask_exceptions                 ssrf    SYSREG_EM_OFFSET
+       .macro  mask_interrupts
+       ssrf    SYSREG_GM_OFFSET
+#ifdef CONFIG_CPU_AT32UC3A0XXX
+       /*
+        * Workaround for errata 41.4.5.5:
+        * "Need two NOPs instruction after instructions masking interrupts"
+        */
+       nop
+       nop
+#endif
+       .endm
+
+       .macro  mask_exceptions
+       ssrf    SYSREG_EM_OFFSET
+#ifdef CONFIG_CPU_AT32UC3A0XXX
+       /*
+        * Workaround for errata 41.4.5.5:
+        * "Need two NOPs instruction after instructions masking interrupts"
+        */
+       nop
+       nop
+#endif
+       .endm
+
 #define unmask_interrupts       csrf    SYSREG_GM_OFFSET
 #define unmask_exceptions       csrf    SYSREG_EM_OFFSET

diff --git a/arch/avr32/include/asm/irqflags.h b/arch/avr32/include/asm/irqflags.h
index 93570da..e25fc64 100644
--- a/arch/avr32/include/asm/irqflags.h
+++ b/arch/avr32/include/asm/irqflags.h
@@ -33,7 +33,15 @@ static inline void raw_local_irq_restore(unsigned long flags)

 static inline void raw_local_irq_disable(void)
 {
+#ifdef CONFIG_CPU_AT32UC3A0XXX
+       /*
+        * Workaround for errata 41.4.5.5:
+        * "Need two NOPs instruction after instructions masking interrupts"
+        */
+       asm volatile("ssrf %0; nop; nop" : : "n"(SYSREG_GM_OFFSET) : "memory");
+#else
        asm volatile("ssrf %0" : : "n"(SYSREG_GM_OFFSET) : "memory");
+#endif
 }

 static inline void raw_local_irq_enable(void)
--
1.6.2.2
```

D.28 UC3A0xxx support

```
commit 45ef6ebbbc75acd8e5aa69ed61023482bfa1b61b
Author: Gunnar Rangoy <gunnar@rangoy.com>
Date:   Thu May 7 13:24:24 2009 +0200

```

```
    AVR32: UC3A0xxx-support

diff --git a/arch/avr32/Kconfig b/arch/avr32/Kconfig
index e3f6653..631d388 100644
--- a/arch/avr32/Kconfig
+++ b/arch/avr32/Kconfig
@@ -107,6 +107,13 @@ config PLATFORM_AT32AP
        select AVR32_CACHE
        select AVR32_UNALIGNED

+config PLATFORM_AT32UC3A
+       bool
+       select SUBARCH_AVR32A
+       select PERFORMANCE_COUNTERS
+       select ARCH_REQUIRE_GPIOLIB
+       select GENERIC_ALLOCATOR
+
 #
 # CPU types
 #
@@ -125,6 +132,11 @@ config CPU_AT32AP7002
        bool
        select CPU_AT32AP700X

+# UC3A0
+config CPU_AT32UC3A0XXX
+       bool
+       select PLATFORM_AT32UC3A
+
 choice
        prompt "AVR32 board type"
        default BOARD_ATSTK1000
@@ -158,18 +170,22 @@ config    LOADER_U_BOOT
 endchoice

 source "arch/avr32/mach-at32ap/Kconfig"
+source "arch/avr32/mach-at32uc3a/Kconfig"

 config LOAD_ADDRESS
        hex
        default 0x10000000 if LOADER_U_BOOT=y && CPU_AT32AP700X=y
+       default 0xc8000000 if LOADER_U_BOOT=y && CPU_AT32UC3A0XXX=y

 config ENTRY_ADDRESS
        hex
        default 0x90000000 if LOADER_U_BOOT=y && CPU_AT32AP700X=y
+       default 0xc8000000 if LOADER_U_BOOT=y && CPU_AT32UC3A0XXX=y

 config PHYS_OFFSET
        hex
        default 0x10000000 if CPU_AT32AP700X=y
+       default 0xc8000000 if CPU_AT32UC3A0XXX=y

 source "kernel/Kconfig.preempt"

diff --git a/arch/avr32/Makefile b/arch/avr32/Makefile
index 4864cb1..ad1dd87 100644
--- a/arch/avr32/Makefile
+++ b/arch/avr32/Makefile
@@ -31,6 +31,7 @@ CFLAGS_MODULE += -mno-relax
 LDFLAGS_vmlinux        += --relax

 cpuflags-$(CONFIG_PLATFORM_AT32AP)     += -march=ap
+cpuflags-$(CONFIG_PLATFORM_AT32UC3A)   += -march=ucr1

 KBUILD_CFLAGS  += $(cpuflags-y)
 KBUILD_AFLAGS  += $(cpuflags-y)
@@ -38,6 +39,7 @@ KBUILD_AFLAGS += $(cpuflags-y)
 CHECKFLAGS     += -D__avr32__ -D__BIG_ENDIAN

 machine-$(CONFIG_PLATFORM_AT32AP) := at32ap
+machine-$(CONFIG_PLATFORM_AT32UC3A) := at32uc3a
 machdirs       := $(patsubst %,arch/avr32/mach-%/, $(machine-y))

 KBUILD_CPPFLAGS        += $(patsubst %,-I$(srctree)/%include,$(machdirs))
diff --git a/arch/avr32/kernel/cpu.c b/arch/avr32/kernel/cpu.c
index e84faff..905a920 100644
--- a/arch/avr32/kernel/cpu.c
+++ b/arch/avr32/kernel/cpu.c
@@ -208,6 +208,7 @@ struct chip_id_map {

 static const struct chip_id_map chip_names[] = {
        { .mid = 0x1f, .pn = 0x1e82, .name = "AT32AP700x" },
+       { .mid = 0x1f, .pn = 0x1edc, .name = "AT32UC3A0xxx" },
```

```
 };
 #define NR_CHIP_NAMES ARRAY_SIZE(chip_names)

diff --git a/arch/avr32/mach-at32ap/Kconfig b/arch/avr32/mach-at32uc3a/Kconfig
similarity index 52%
copy from arch/avr32/mach-at32ap/Kconfig
copy to arch/avr32/mach-at32uc3a/Kconfig
index a7bbcc8..dea8d93 100644
--- a/arch/avr32/mach-at32ap/Kconfig
+++ b/arch/avr32/mach-at32uc3a/Kconfig
@@ -1,13 +1,13 @@
-if PLATFORM_AT32AP
+if PLATFORM_AT32UC3A

-menu "Atmel AVR32 AP options"
+menu "Atmel AVR32 UC3A options"

 choice
-       prompt "AT32AP700x static memory bus width"
-       depends on CPU_AT32AP700X
-       default AP700X_16_BIT_SMC
+       prompt "AT32UC3A0XXX static memory bus width"
+       depends on CPU_AT32UC3A0XXX
+       default UC3A0XXX_16_BIT_SMC
        help
-         Define the width of the AP7000 external static memory interface.
+         Define the width of the UC3A external static memory interface.
          This is used to determine how to mangle the address and/or data
          when doing little-endian port access.

@@ -15,17 +15,14 @@ choice
          width for all chip selects, excluding the flash (which is using
          raw access and is thus not affected by any of this.)

-config AP700X_32_BIT_SMC
-       bool "32 bit"
-
-config AP700X_16_BIT_SMC
+config UC3A0XXX_16_BIT_SMC
        bool "16 bit"

-config AP700X_8_BIT_SMC
+config UC3A0XXX_8_BIT_SMC
        bool "8 bit"

 endchoice

 endmenu

-endif # PLATFORM_AT32AP
+endif # PLATFORM_AT32UC3A
diff --git a/arch/avr32/mach-at32uc3a/Makefile b/arch/avr32/mach-at32uc3a/Makefile
new file mode 100644
index 0000000..0bf3edc
--- /dev/null
+++ b/arch/avr32/mach-at32uc3a/Makefile
@@ -0,0 +1,9 @@
+obj-y                          += pdca.o clock.o intc.o extint.o gpio.o hsmc.o
+obj-y                          += hmatrix.o
+obj-$(CONFIG_CPU_AT32UC3A0XXX) += at32uc3a0xxx.o pm-at32uc3a0xxx.o
+obj-$(CONFIG_CPU_FREQ_AT32UC3A0)       += cpufreq.o
+obj-$(CONFIG_PM)               += pm.o
+
+ifeq ($(CONFIG_PM_DEBUG),y)
+CFLAGS_pm.o    += -DDEBUG
+endif
diff --git a/arch/avr32/mach-at32uc3a/at32uc3a0xxx.c b/arch/avr32/mach-at32uc3a/at32uc3a0xxx.c
new file mode 100644
index 0000000..f7610f1
--- /dev/null
+++ b/arch/avr32/mach-at32uc3a/at32uc3a0xxx.c
@@ -0,0 +1,1453 @@
+/*
+ * Copyright (C) 2005-2006 Atmel Corporation
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the Free Software Foundation.
+ */
+#include <linux/clk.h>
+#include <linux/delay.h>
+#include <linux/dw_dmac.h>
+#include <linux/fb.h>
+#include <linux/init.h>
```

```
+#include <linux/platform_device.h>
+#include <linux/dma-mapping.h>
+#include <linux/gpio.h>
+#include <linux/spi/spi.h>
+#include <linux/usb/atmel_usba_udc.h>
+
+#include <asm/atmel-mci.h>
+#include <asm/io.h>
+#include <asm/irq.h>
+
+#include <mach/at32uc3a0xxx.h>
+#include <mach/board.h>
+#include <mach/hmatrix.h>
+#include <mach/portmux.h>
+#include <mach/sram.h>
+
+#include "clock.h"
+#include "gpio.h"
+#include "pm.h"
+
+#define MEMRANGE(base,size)                             \
+       {                                                \
+               .start          = base,                  \
+               .end            = base + size - 1,       \
+               .flags          = IORESOURCE_MEM,        \
+       }
+#define PBMEM(base)                                     \
+       {                                                \
+               .start          = base,                  \
+               .end            = base + 0x3ff,          \
+               .flags          = IORESOURCE_MEM,        \
+       }
+#define IRQ(num)                                        \
+       {                                                \
+               .start          = num,                   \
+               .end            = num,                   \
+               .flags          = IORESOURCE_IRQ,        \
+       }
+#define NAMED_IRQ(num, _name)                           \
+       {                                                \
+               .start          = num,                   \
+               .end            = num,                   \
+               .name           = _name,                 \
+               .flags          = IORESOURCE_IRQ,        \
+       }
+
+/* REVISIT these assume *every* device supports DMA, but several
+ * don't ... tc, smc, pio, rtc, watchdog, pwm, ps2, and more.
+ */
+#define DEFINE_DEV(_name, _id)                                  \
+static u64 _name##_id##_dma_mask = DMA_32BIT_MASK;              \
+static struct platform_device _name##_id##_device = {           \
+       .name           = #_name,                                \
+       .id             = _id,                                   \
+       .dev            = {                                      \
+               .dma_mask = &_name##_id##_dma_mask,              \
+               .coherent_dma_mask = DMA_32BIT_MASK,             \
+       },                                                       \
+       .resource       = _name##_id##_resource,                 \
+       .num_resources  = ARRAY_SIZE(_name##_id##_resource),     \
+}
+#define DEFINE_DEV_DATA(_name, _id)                             \
+static u64 _name##_id##_dma_mask = DMA_32BIT_MASK;              \
+static struct platform_device _name##_id##_device = {           \
+       .name           = #_name,                                \
+       .id             = _id,                                   \
+       .dev            = {                                      \
+               .dma_mask = &_name##_id##_dma_mask,              \
+               .platform_data  = &_name##_id##_data,            \
+               .coherent_dma_mask = DMA_32BIT_MASK,             \
+       },                                                       \
+       .resource       = _name##_id##_resource,                 \
+       .num_resources  = ARRAY_SIZE(_name##_id##_resource),     \
+}
+
+#define select_peripheral(pin, periph, flags)                   \
+       at32_select_periph(GPIO_PIN_##pin, GPIO_##periph, flags)
+
+#define DEV_CLK(_name, devname, bus, _index)                    \
+static struct clk devname##_##_name = {                                 \
+       .name           = #_name,                                \
+       .dev            = &devname##_device.dev,                 \
+       .parent         = &bus##_clk,                            \
+       .mode           = bus##_clk_mode,                        \
```

```
257 +       .get_rate       = bus##_clk_get_rate,                   \
258 +       .index          = _index,                               \
259 +}
260 +
261 +static DEFINE_SPINLOCK(pm_lock);
262 +
263 +static struct clk osc0;
264 +static struct clk osc1;
265 +
266 +static unsigned long osc_get_rate(struct clk *clk)
267 +{
268 +       return at32_board_osc_rates[clk->index];
269 +}
270 +
271 +static unsigned long pll_get_rate(struct clk *clk, unsigned long control)
272 +{
273 +       unsigned long div, mul, rate;
274 +
275 +       div = PM_BFEXT(PLLDIV, control) + 1;
276 +       mul = PM_BFEXT(PLLMUL, control) + 1;
277 +
278 +       rate = clk->parent->get_rate(clk->parent);
279 +       rate = (rate + div / 2) / div;
280 +       rate *= mul;
281 +
282 +       return rate;
283 +}
284 +
285 +static long pll_set_rate(struct clk *clk, unsigned long rate,
286 +                        u32 *pll_ctrl)
287 +{
288 +       unsigned long mul;
289 +       unsigned long mul_best_fit = 0;
290 +       unsigned long div;
291 +       unsigned long div_min;
292 +       unsigned long div_max;
293 +       unsigned long div_best_fit = 0;
294 +       unsigned long base;
295 +       unsigned long pll_in;
296 +       unsigned long actual = 0;
297 +       unsigned long rate_error;
298 +       unsigned long rate_error_prev = ~0UL;
299 +       u32 ctrl;
300 +
301 +       /* Rate must be between 80 MHz and 200 Mhz. */
302 +       if (rate < 80000000UL || rate > 200000000UL)
303 +               return -EINVAL;
304 +
305 +       ctrl = PM_BF(PLLOPT, 4);
306 +       base = clk->parent->get_rate(clk->parent);
307 +
308 +       /* PLL input frequency must be between 6 MHz and 32 MHz. */
309 +       div_min = DIV_ROUND_UP(base, 32000000UL);
310 +       div_max = base / 6000000UL;
311 +
312 +       if (div_max < div_min)
313 +               return -EINVAL;
314 +
315 +       for (div = div_min; div <= div_max; div++) {
316 +               pll_in = (base + div / 2) / div;
317 +               mul = (rate + pll_in / 2) / pll_in;
318 +
319 +               if (mul == 0)
320 +                       continue;
321 +
322 +               actual = pll_in * mul;
323 +               rate_error = abs(actual - rate);
324 +
325 +               if (rate_error < rate_error_prev) {
326 +                       mul_best_fit = mul;
327 +                       div_best_fit = div;
328 +                       rate_error_prev = rate_error;
329 +               }
330 +
331 +               if (rate_error == 0)
332 +                       break;
333 +       }
334 +
335 +       if (div_best_fit == 0)
336 +               return -EINVAL;
337 +
338 +       ctrl |= PM_BF(PLLMUL, mul_best_fit - 1);
339 +       ctrl |= PM_BF(PLLDIV, div_best_fit - 1);
340 +       ctrl |= PM_BF(PLLCOUNT, 16);
```

```
+
+       if (clk->parent == &osc1)
+               ctrl |= PM_BIT(PLLOSC);
+
+       *pll_ctrl = ctrl;
+
+       return actual;
+}
+
+static unsigned long pll0_get_rate(struct clk *clk)
+{
+       u32 control;
+
+       control = pm_readl(PLL0);
+
+       return pll_get_rate(clk, control);
+}
+
+static void pll1_mode(struct clk *clk, int enabled)
+{
+       unsigned long timeout;
+       u32 status;
+       u32 ctrl;
+
+       ctrl = pm_readl(PLL1);
+
+       if (enabled) {
+               if (!PM_BFEXT(PLLMUL, ctrl) && !PM_BFEXT(PLLDIV, ctrl)) {
+                       pr_debug("clk %s: failed to enable, rate not set\n",
+                                clk->name);
+                       return;
+               }
+
+               ctrl |= PM_BIT(PLLEN);
+               pm_writel(PLL1, ctrl);
+
+               /* Wait for PLL lock. */
+               for (timeout = 10000; timeout; timeout--) {
+                       status = pm_readl(ISR);
+                       if (status & PM_BIT(LOCK1))
+                               break;
+                       udelay(10);
+               }
+
+               if (!(status & PM_BIT(LOCK1)))
+                       printk(KERN_ERR "clk %s: timeout waiting for lock\n",
+                              clk->name);
+       } else {
+               ctrl &= ~PM_BIT(PLLEN);
+               pm_writel(PLL1, ctrl);
+       }
+}
+
+static unsigned long pll1_get_rate(struct clk *clk)
+{
+       u32 control;
+
+       control = pm_readl(PLL1);
+
+       return pll_get_rate(clk, control);
+}
+
+static long pll1_set_rate(struct clk *clk, unsigned long rate, int apply)
+{
+       u32 ctrl = 0;
+       unsigned long actual_rate;
+
+       actual_rate = pll_set_rate(clk, rate, &ctrl);
+
+       if (apply) {
+               if (actual_rate != rate)
+                       return -EINVAL;
+               if (clk->users > 0)
+                       return -EBUSY;
+               pr_debug(KERN_INFO "clk %s: new rate %lu (actual rate %lu)\n",
+                        clk->name, rate, actual_rate);
+               pm_writel(PLL1, ctrl);
+       }
+
+       return actual_rate;
+}
+
+static int pll1_set_parent(struct clk *clk, struct clk *parent)
+{
```

```
425 +       u32 ctrl;
426 +
427 +       if (clk->users > 0)
428 +               return -EBUSY;
429 +
430 +       ctrl = pm_readl(PLL1);
431 +       WARN_ON(ctrl & PM_BIT(PLLEN));
432 +
433 +       if (parent == &osc0)
434 +               ctrl &= ~PM_BIT(PLLOSC);
435 +       else if (parent == &osc1)
436 +               ctrl |= PM_BIT(PLLOSC);
437 +       else
438 +               return -EINVAL;
439 +
440 +       pm_writel(PLL1, ctrl);
441 +       clk->parent = parent;
442 +
443 +       return 0;
444 +}
445 +
446 +/*
447 + * The AT32UC3A0512 has six primary clock sources: One 32kHz oscillator,
448 + * one, external slow-clock, two crystal oscillators and two PLLs.
449 + */
450 +static struct clk osc32k = {
451 +       .name           = "osc32k",
452 +       .get_rate       = osc_get_rate,
453 +       .users          = 1,
454 +       .index          = 0,
455 +};
456 +static struct clk osc0 = {
457 +       .name           = "osc0",
458 +       .get_rate       = osc_get_rate,
459 +       .users          = 1,
460 +       .index          = 1,
461 +};
462 +static struct clk osc1 = {
463 +       .name           = "osc1",
464 +       .get_rate       = osc_get_rate,
465 +       .index          = 2,
466 +};
467 +static struct clk pll0 = {
468 +       .name           = "pll0",
469 +       .get_rate       = pll0_get_rate,
470 +       .parent         = &osc0,
471 +};
472 +static struct clk pll1 = {
473 +       .name           = "pll1",
474 +       .mode           = pll1_mode,
475 +       .get_rate       = pll1_get_rate,
476 +       .set_rate       = pll1_set_rate,
477 +       .set_parent     = pll1_set_parent,
478 +       .parent         = &osc0,
479 +};
480 +
481 +/*
482 + * The main clock can be either osc0 or pll0.  The boot loader may
483 + * have chosen one for us, so we don't really know which one until we
484 + * have a look at the SM.
485 + */
486 +static struct clk *main_clock;
487 +
488 +/*
489 + * Synchronous clocks are generated from the main clock. The clocks
490 + * must satisfy the constraint
491 + *   fCPU >= fHSB >= fPB
492 + * i.e. each clock must not be faster than its parent.
493 + */
494 +static unsigned long bus_clk_get_rate(struct clk *clk, unsigned int shift)
495 +{
496 +       return main_clock->get_rate(main_clock) >> shift;
497 +};
498 +
499 +static void cpu_clk_mode(struct clk *clk, int enabled)
500 +{
501 +       unsigned long flags;
502 +       u32 mask;
503 +
504 +       spin_lock_irqsave(&pm_lock, flags);
505 +       mask = pm_readl(CPU_MASK);
506 +       if (enabled)
507 +               mask |= 1 << clk->index;
508 +       else
```

```
509 +                mask &= ~(1 << clk->index);
510 +        pm_writel(CPU_MASK, mask);
511 +        spin_unlock_irqrestore(&pm_lock, flags);
512 +}
513 +
514 +static unsigned long cpu_clk_get_rate(struct clk *clk)
515 +{
516 +        unsigned long cksel, shift = 0;
517 +
518 +        cksel = pm_readl(CKSEL);
519 +        if (cksel & PM_BIT(CPUDIV))
520 +                shift = PM_BFEXT(CPUSEL, cksel) + 1;
521 +
522 +        return bus_clk_get_rate(clk, shift);
523 +}
524 +
525 +static long cpu_clk_set_rate(struct clk *clk, unsigned long rate, int apply)
526 +{
527 +        u32 control;
528 +        unsigned long parent_rate, child_div, actual_rate, div;
529 +
530 +        parent_rate = clk->parent->get_rate(clk->parent);
531 +        control = pm_readl(CKSEL);
532 +
533 +        if (control & PM_BIT(HSBDIV))
534 +                child_div = 1 << (PM_BFEXT(HSBSEL, control) + 1);
535 +        else
536 +                child_div = 1;
537 +
538 +        if (rate > 3 * (parent_rate / 4) || child_div == 1) {
539 +                actual_rate = parent_rate;
540 +                control &= ~PM_BIT(CPUDIV);
541 +        } else {
542 +                unsigned int cpusel;
543 +                div = (parent_rate + rate / 2) / rate;
544 +                if (div > child_div)
545 +                        div = child_div;
546 +                cpusel = (div > 1) ? (fls(div) - 2) : 0;
547 +                control = PM_BIT(CPUDIV) | PM_BFINS(CPUSEL, cpusel, control);
548 +                actual_rate = parent_rate / (1 << (cpusel + 1));
549 +        }
550 +
551 +        pr_debug("clk %s: new rate %lu (actual rate %lu)\n",
552 +                        clk->name, rate, actual_rate);
553 +
554 +        if (apply)
555 +                pm_writel(CKSEL, control);
556 +
557 +        return actual_rate;
558 +}
559 +
560 +static void hsb_clk_mode(struct clk *clk, int enabled)
561 +{
562 +        unsigned long flags;
563 +        u32 mask;
564 +
565 +        spin_lock_irqsave(&pm_lock, flags);
566 +        mask = pm_readl(HSB_MASK);
567 +        if (enabled)
568 +                mask |= 1 << clk->index;
569 +        else
570 +                mask &= ~(1 << clk->index);
571 +        pm_writel(HSB_MASK, mask);
572 +        spin_unlock_irqrestore(&pm_lock, flags);
573 +}
574 +
575 +static unsigned long hsb_clk_get_rate(struct clk *clk)
576 +{
577 +        unsigned long cksel, shift = 0;
578 +
579 +        cksel = pm_readl(CKSEL);
580 +        if (cksel & PM_BIT(HSBDIV))
581 +                shift = PM_BFEXT(HSBSEL, cksel) + 1;
582 +
583 +        return bus_clk_get_rate(clk, shift);
584 +}
585 +
586 +static void pba_clk_mode(struct clk *clk, int enabled)
587 +{
588 +        unsigned long flags;
589 +        u32 mask;
590 +
591 +        spin_lock_irqsave(&pm_lock, flags);
592 +        mask = pm_readl(PBA_MASK);
```

```
+	if (enabled)
+		mask |= 1 << clk->index;
+	else
+		mask &= ~(1 << clk->index);
+	pm_writel(PBA_MASK, mask);
+	spin_unlock_irqrestore(&pm_lock, flags);
+}
+
+static unsigned long pba_clk_get_rate(struct clk *clk)
+{
+	unsigned long cksel, shift = 0;
+
+	cksel = pm_readl(CKSEL);
+	if (cksel & PM_BIT(PBADIV))
+		shift = PM_BFEXT(PBASEL, cksel) + 1;
+
+	return bus_clk_get_rate(clk, shift);
+}
+
+static void pbb_clk_mode(struct clk *clk, int enabled)
+{
+	unsigned long flags;
+	u32 mask;
+
+	spin_lock_irqsave(&pm_lock, flags);
+	mask = pm_readl(PBB_MASK);
+	if (enabled)
+		mask |= 1 << clk->index;
+	else
+		mask &= ~(1 << clk->index);
+	pm_writel(PBB_MASK, mask);
+	spin_unlock_irqrestore(&pm_lock, flags);
+}
+
+static unsigned long pbb_clk_get_rate(struct clk *clk)
+{
+	unsigned long cksel, shift = 0;
+
+	cksel = pm_readl(CKSEL);
+	if (cksel & PM_BIT(PBBDIV))
+		shift = PM_BFEXT(PBBSEL, cksel) + 1;
+
+	return bus_clk_get_rate(clk, shift);
+}
+
+static struct clk cpu_clk = {
+	.name		= "cpu",
+	.get_rate	= cpu_clk_get_rate,
+	.set_rate	= cpu_clk_set_rate,
+	.users		= 1,
+};
+static struct clk hsb_clk = {
+	.name		= "hsb",
+	.parent		= &cpu_clk,
+	.get_rate	= hsb_clk_get_rate,
+};
+static struct clk pba_clk = {
+	.name		= "pba",
+	.parent		= &hsb_clk,
+	.mode		= hsb_clk_mode,
+	.get_rate	= pba_clk_get_rate,
+	.users		= 1,
+	.index		= 1,
+};
+static struct clk pbb_clk = {
+	.name		= "pbb",
+	.parent		= &hsb_clk,
+	.mode		= hsb_clk_mode,
+	.get_rate	= pbb_clk_get_rate,
+	.users		= 1,
+	.index		= 2,
+};
+
+/* --------------------------------------------------------------------
+ *  Generic Clock operations
+ * -------------------------------------------------------------------- */
+
+static void genclk_mode(struct clk *clk, int enabled)
+{
+	u32 control;
+
+	control = pm_readl(GCCTRL(clk->index));
+	if (enabled)
+		control |= PM_BIT(CEN);
```

```
+       else
+               control &= ~PM_BIT(CEN);
+       pm_writel(GCCTRL(clk->index), control);
+}
+
+static unsigned long genclk_get_rate(struct clk *clk)
+{
+       u32 control;
+       unsigned long div = 1;
+
+       control = pm_readl(GCCTRL(clk->index));
+       if (control & PM_BIT(DIVEN))
+               div = 2 * (PM_BFEXT(DIV, control) + 1);
+
+       return clk->parent->get_rate(clk->parent) / div;
+}
+
+static long genclk_set_rate(struct clk *clk, unsigned long rate, int apply)
+{
+       u32 control;
+       unsigned long parent_rate, actual_rate, div;
+
+       parent_rate = clk->parent->get_rate(clk->parent);
+       control = pm_readl(GCCTRL(clk->index));
+
+       if (rate > 3 * parent_rate / 4) {
+               actual_rate = parent_rate;
+               control &= ~PM_BIT(DIVEN);
+       } else {
+               div = (parent_rate + rate) / (2 * rate) - 1;
+               control = PM_BFINS(DIV, div, control) | PM_BIT(DIVEN);
+               actual_rate = parent_rate / (2 * (div + 1));
+       }
+
+       dev_dbg(clk->dev, "clk %s: new rate %lu (actual rate %lu)\n",
+               clk->name, rate, actual_rate);
+
+       if (apply)
+               pm_writel(GCCTRL(clk->index), control);
+
+       return actual_rate;
+}
+
+int genclk_set_parent(struct clk *clk, struct clk *parent)
+{
+       u32 control;
+
+       dev_dbg(clk->dev, "clk %s: new parent %s (was %s)\n",
+               clk->name, parent->name, clk->parent->name);
+
+       control = pm_readl(GCCTRL(clk->index));
+
+       if (parent == &osc1 || parent == &pll1)
+               control |= PM_BIT(OSCSEL);
+       else if (parent == &osc0 || parent == &pll0)
+               control &= ~PM_BIT(OSCSEL);
+       else
+               return -EINVAL;
+
+       if (parent == &pll0 || parent == &pll1)
+               control |= PM_BIT(PLLSEL);
+       else
+               control &= ~PM_BIT(PLLSEL);
+
+       pm_writel(GCCTRL(clk->index), control);
+       clk->parent = parent;
+
+       return 0;
+}
+
+static void __init genclk_init_parent(struct clk *clk)
+{
+       u32 control;
+       struct clk *parent;
+
+       BUG_ON(clk->index > 7);
+
+       control = pm_readl(GCCTRL(clk->index));
+       if (control & PM_BIT(OSCSEL))
+               parent = (control & PM_BIT(PLLSEL)) ? &pll1 : &osc1;
+       else
+               parent = (control & PM_BIT(PLLSEL)) ? &pll0 : &osc0;
+
+       clk->parent = parent;
```

```
761 +}
762 +
763 +/* ---------------------------------------------------------------------
764 + *  System peripherals
765 + * --------------------------------------------------------------------- */
766 +static struct resource at32_pm0_resource[] = {
767 +       /* Note that the PM has a size of at least 0x208. However, the
768 +        * RTC, WDT and EIC are embedded in this structure, so we set
769 +        * the size to 0x100 to avoid overlap.
770 +        */
771 +       MEMRANGE(0xffff0c00, 0x100),
772 +       IRQ(1),
773 +};
774 +
775 +static struct resource at32uc3a0xxx_rtc0_resource[] = {
776 +       MEMRANGE(0xffff0d00, 0x24),
777 +       IRQ(1),
778 +};
779 +
780 +static struct resource at32_wdt0_resource[] = {
781 +       MEMRANGE(0xffff0d30, 0x8),
782 +};
783 +
784 +static struct resource at32_eic0_resource[] = {
785 +       MEMRANGE(0xffff0d80, 0x3c),
786 +       IRQ(1),
787 +};
788 +
789 +DEFINE_DEV(at32_pm, 0);
790 +DEFINE_DEV(at32uc3a0xxx_rtc, 0);
791 +DEFINE_DEV(at32_wdt, 0);
792 +DEFINE_DEV(at32_eic, 0);
793 +
794 +/*
795 + * Peripheral clock for PM, RTC and EIC. PM will ensure that this
796 + * is always running.
797 + */
798 +static struct clk at32_pm_pclk = {
799 +       .name           = "pclk",
800 +       .dev            = &at32_pm0_device.dev,
801 +       .parent         = &pba_clk,
802 +       .mode           = pba_clk_mode,
803 +       .get_rate       = pba_clk_get_rate,
804 +       .users          = 1,
805 +       .index          = 3,
806 +};
807 +
808 +static struct resource intc0_resource[] = {
809 +       PBMEM(0xffff0800),
810 +};
811 +struct platform_device at32_intc0_device = {
812 +       .name           = "intc",
813 +       .id             = 0,
814 +       .resource       = intc0_resource,
815 +       .num_resources  = ARRAY_SIZE(intc0_resource),
816 +};
817 +DEV_CLK(pclk, at32_intc0, pba, 0);
818 +
819 +static struct clk ebi_clk = {
820 +       .name           = "ebi",
821 +       .parent         = &hsb_clk,
822 +       .mode           = hsb_clk_mode,
823 +       .get_rate       = hsb_clk_get_rate,
824 +       .users          = 6,
825 +};
826 +static struct clk sdramc_clk = {
827 +       .name           = "sdramc_clk",
828 +       .parent         = &pbb_clk,
829 +       .mode           = pbb_clk_mode,
830 +       .get_rate       = pbb_clk_get_rate,
831 +       .users          = 1,
832 +       .index          = 5,
833 +};
834 +
835 +static struct resource smc0_resource[] = {
836 +       PBMEM(0xfffe1c00),
837 +};
838 +DEFINE_DEV(smc, 0);
839 +static struct clk smc0_pclk = {
840 +       .name           = "pclk",
841 +       .dev            = &smc0_device.dev,
842 +       .parent         = &pbb_clk,
843 +       .mode           = pbb_clk_mode,
844 +       .get_rate       = pbb_clk_get_rate,
```

```
845 +       .users          = 1,
846 +       .index          = 4,
847 +};
848 +static struct clk smc0_mck = {
849 +       .name           = "mck",
850 +       .dev            = &smc0_device.dev,
851 +       .parent         = &hsb_clk,
852 +       .mode           = hsb_clk_mode,
853 +       .get_rate       = hsb_clk_get_rate,
854 +       .users          = 1,
855 +       .index          = 6,
856 +};
857 +
858 +static struct platform_device pdca_device = {
859 +       .name           = "pdca",
860 +       .id             = 0,
861 +};
862 +DEV_CLK(pclk, pdca, pba, 2);
863 +
864 +/* --------------------------------------------------------------------
865 + * HMATRIX
866 + * -------------------------------------------------------------------- */
867 +
868 +struct clk at32_hmatrix_clk = {
869 +       .name           = "hmatrix_clk",
870 +       .parent         = &pbb_clk,
871 +       .mode           = pbb_clk_mode,
872 +       .get_rate       = pbb_clk_get_rate,
873 +       .index          = 0,
874 +       .users          = 1,
875 +};
876 +
877 +/*
878 + * Set bits in the HMATRIX Special Function Register (SFR) used by the
879 + * External Bus Interface (EBI). This can be used to enable special
880 + * features like CompactFlash support, NAND Flash support, etc. on
881 + * certain chipselects.
882 + */
883 +static inline void set_ebi_sfr_bits(u32 mask)
884 +{
885 +       hmatrix_sfr_set_bits(HMATRIX_SLAVE_EBI, mask);
886 +}
887 +
888 +/* --------------------------------------------------------------------
889 + *  Timer/Counter (TC)
890 + * -------------------------------------------------------------------- */
891 +
892 +static struct resource at32_tc0_resource[] = {
893 +       PBMEM(0xffff3800),
894 +       IRQ(14),
895 +};
896 +static struct platform_device at32_tc0_device = {
897 +       .name           = "atmel_tc",
898 +       .id             = 0,
899 +       .resource       = at32_tc0_resource,
900 +       .num_resources  = ARRAY_SIZE(at32_tc0_resource),
901 +};
902 +DEV_CLK(t0_clk, at32_tc0, pbb, 3);
903 +
904 +/* --------------------------------------------------------------------
905 + *  On-Chip Debug
906 + * -------------------------------------------------------------------- */
907 +
908 +static struct resource at32_ocd0_resource[] = {
909 +};
910 +static struct platform_device at32_ocd0_device = {
911 +       .name           = "atmel_ocd",
912 +       .id             = 0,
913 +       .resource       = at32_ocd0_resource,
914 +       .num_resources  = ARRAY_SIZE(at32_ocd0_resource),
915 +};
916 +struct clk at32_ocd0_clk = {
917 +       .name           = "at32_ocd0_clk",
918 +       .parent         = &cpu_clk,
919 +       .mode           = cpu_clk_mode,
920 +       .get_rate       = cpu_clk_get_rate,
921 +       .index          = 1,
922 +       .users          = 1,
923 +};
924 +
925 +
926 +/* --------------------------------------------------------------------
927 + *  GPIO
928 + * -------------------------------------------------------------------- */
```

```
929 +static struct resource gpio0_resource[] = {
930 +       MEMRANGE(0xffff1000, 0x100),
931 +       IRQ(2),
932 +};
933 +DEFINE_DEV(gpio, 0);
934 +DEV_CLK(mck, gpio0, pba, 1);
935 +
936 +static struct resource gpio1_resource[] = {
937 +       MEMRANGE(0xffff1100, 0x100),
938 +       IRQ(2),
939 +};
940 +DEFINE_DEV(gpio, 1);
941 +DEV_CLK(mck, gpio1, pba, 1);
942 +
943 +static struct resource gpio2_resource[] = {
944 +       MEMRANGE(0xffff1200, 0x100),
945 +       IRQ(2),
946 +};
947 +DEFINE_DEV(gpio, 2);
948 +DEV_CLK(mck, gpio2, pba, 1);
949 +
950 +static struct resource gpio3_resource[] = {
951 +       MEMRANGE(0xffff1300, 0x100),
952 +       IRQ(2),
953 +};
954 +DEFINE_DEV(gpio, 3);
955 +DEV_CLK(mck, gpio3, pba, 1);
956 +
957 +void __init at32_add_system_devices(void)
958 +{
959 +       platform_device_register(&at32_pm0_device);
960 +       platform_device_register(&at32_intc0_device);
961 +       platform_device_register(&at32uc3a0xxx_rtc0_device);
962 +       platform_device_register(&at32_wdt0_device);
963 +       platform_device_register(&at32_eic0_device);
964 +       platform_device_register(&smc0_device);
965 +       platform_device_register(&pdca_device);
966 +       platform_device_register(&at32_ocd0_device);
967 +
968 +       platform_device_register(&at32_tc0_device);
969 +
970 +       platform_device_register(&gpio0_device);
971 +       platform_device_register(&gpio1_device);
972 +       platform_device_register(&gpio2_device);
973 +       platform_device_register(&gpio3_device);
974 +}
975 +
976 +
977 +/* --------------------------------------------------------------------
978 + *  USART
979 + * -------------------------------------------------------------------- */
980 +
981 +static struct atmel_uart_data atmel_usart0_data = {
982 +       .use_dma_tx     = 1,
983 +       .use_dma_rx     = 1,
984 +};
985 +static struct resource atmel_usart0_resource[] = {
986 +       PBMEM(0xffff1400),
987 +       IRQ(5),
988 +};
989 +DEFINE_DEV_DATA(atmel_usart, 0);
990 +DEV_CLK(usart, atmel_usart0, pba, 8);
991 +
992 +static struct atmel_uart_data atmel_usart1_data = {
993 +       .use_dma_tx     = 1,
994 +       .use_dma_rx     = 1,
995 +};
996 +static struct resource atmel_usart1_resource[] = {
997 +       PBMEM(0xffff1800),
998 +       IRQ(6),
999 +};
1000 +DEFINE_DEV_DATA(atmel_usart, 1);
1001 +DEV_CLK(usart, atmel_usart1, pba, 9);
1002 +
1003 +static struct atmel_uart_data atmel_usart2_data = {
1004 +       .use_dma_tx     = 1,
1005 +       .use_dma_rx     = 1,
1006 +};
1007 +static struct resource atmel_usart2_resource[] = {
1008 +       PBMEM(0xffff1c00),
1009 +       IRQ(7),
1010 +};
1011 +DEFINE_DEV_DATA(atmel_usart, 2);
1012 +DEV_CLK(usart, atmel_usart2, pba, 10);
```

```
+
+static struct atmel_uart_data atmel_usart3_data = {
+       .use_dma_tx     = 1,
+       .use_dma_rx     = 1,
+};
+static struct resource atmel_usart3_resource[] = {
+       PBMEM(0xffff2000),
+       IRQ(8),
+};
+DEFINE_DEV_DATA(atmel_usart, 3);
+DEV_CLK(usart, atmel_usart3, pba, 11);
+
+static inline void configure_usart0_pins(void)
+{
+       select_peripheral(PA(0),  PERIPH_A, 0); /* RXD  */
+       select_peripheral(PA(1),  PERIPH_A, 0); /* TXD  */
+}
+
+static inline void configure_usart1_pins(void)
+{
+       select_peripheral(PA(5), PERIPH_A, 0);  /* RXD  */
+       select_peripheral(PA(6), PERIPH_A, 0);  /* TXD  */
+}
+
+static inline void configure_usart2_pins(void)
+{
+       select_peripheral(PB(29), PERIPH_A, 0); /* RXD  */
+       select_peripheral(PB(30), PERIPH_A, 0); /* TXD  */
+}
+
+static inline void configure_usart3_pins(void)
+{
+       select_peripheral(PB(10), PERIPH_B, 0); /* RXD  */
+       select_peripheral(PB(11), PERIPH_B, 0); /* TXD  */
+}
+
+static struct platform_device *__initdata at32_usarts[4];
+
+void __init at32_map_usart(unsigned int hw_id, unsigned int line)
+{
+       struct platform_device *pdev;
+       struct atmel_uart_data *data;
+
+       switch (hw_id) {
+       case 0:
+               pdev = &atmel_usart0_device;
+               configure_usart0_pins();
+               break;
+       case 1:
+               pdev = &atmel_usart1_device;
+               configure_usart1_pins();
+               break;
+       case 2:
+               pdev = &atmel_usart2_device;
+               configure_usart2_pins();
+               break;
+       case 3:
+               pdev = &atmel_usart3_device;
+               configure_usart3_pins();
+               break;
+       default:
+               return;
+       }
+
+       data = pdev->dev.platform_data;
+       data->regs = (void __iomem *)pdev->resource[0].start;
+
+       pdev->id = line;
+       at32_usarts[line] = pdev;
+}
+
+struct platform_device *__init at32_add_device_usart(unsigned int id)
+{
+       platform_device_register(at32_usarts[id]);
+       return at32_usarts[id];
+}
+
+struct platform_device *atmel_default_console_device;
+
+void __init at32_setup_serial_console(unsigned int usart_id)
+{
+       atmel_default_console_device = at32_usarts[usart_id];
+}
+
```

```
+/* --------------------------------------------------------------------
+ *  Ethernet
+ * -------------------------------------------------------------------- */
+
+static struct eth_platform_data macb0_data;
+static struct resource macb0_resource[] = {
+       PBMEM(0xfffe1800),
+       IRQ(16),
+};
+DEFINE_DEV_DATA(macb, 0);
+DEV_CLK(hclk, macb0, hsb, 4);
+DEV_CLK(pclk, macb0, pbb, 3);
+
+
+struct platform_device *__init
+at32_add_device_eth(unsigned int id, struct eth_platform_data *data)
+{
+       struct platform_device *pdev;
+
+       switch (id) {
+       case 0:
+               pdev = &macb0_device;
+
+               at32_select_periph(34, GPIO_PERIPH_A, 0);       /* TXD0 */
+               at32_select_periph(35, GPIO_PERIPH_A, 0);       /* TXD1 */
+               at32_select_periph(33, GPIO_PERIPH_A, 0);       /* TXEN */
+               at32_select_periph(32, GPIO_PERIPH_A, 0);       /* TXCK */
+               at32_select_periph(37, GPIO_PERIPH_A, 0);       /* RXD0 */
+               at32_select_periph(38, GPIO_PERIPH_A, 0);       /* RXD1 */
+               at32_select_periph(39, GPIO_PERIPH_A, 0);       /* RXER */
+               at32_select_periph(47, GPIO_PERIPH_A, 0);       /* RXDV */
+               at32_select_periph(40, GPIO_PERIPH_A, 0);       /* MDC  */
+               at32_select_periph(41, GPIO_PERIPH_A, 0);       /* MDIO */
+
+               if (!data->is_rmii) {
+                       select_peripheral(PC(0),  PERIPH_A, 0); /* COL  */
+                       select_peripheral(PC(1),  PERIPH_A, 0); /* CRS  */
+                       select_peripheral(PC(2),  PERIPH_A, 0); /* TXER */
+                       select_peripheral(PC(5),  PERIPH_A, 0); /* TXD2 */
+                       select_peripheral(PC(6),  PERIPH_A, 0); /* TXD3 */
+                       select_peripheral(PC(11), PERIPH_A, 0); /* RXD2 */
+                       select_peripheral(PC(12), PERIPH_A, 0); /* RXD3 */
+                       select_peripheral(PC(14), PERIPH_A, 0); /* RXCK */
+                       select_peripheral(PC(18), PERIPH_A, 0); /* SPD  */
+               }
+               break;
+
+       default:
+               return NULL;
+       }
+
+       memcpy(pdev->dev.platform_data, data, sizeof(struct eth_platform_data));
+       platform_device_register(pdev);
+
+       return pdev;
+}
+
+/* --------------------------------------------------------------------
+ *  SPI
+ * -------------------------------------------------------------------- */
+static struct resource atmel_spi0_resource[] = {
+       PBMEM(0xffff2400),
+       IRQ(9),
+};
+DEFINE_DEV(atmel_spi, 0);
+DEV_CLK(spi_clk, atmel_spi0, pba, 5);
+
+static struct resource atmel_spi1_resource[] = {
+       PBMEM(0xffff2800),
+       IRQ(10),
+};
+DEFINE_DEV(atmel_spi, 1);
+DEV_CLK(spi_clk, atmel_spi1, pba, 6);
+
+static void __init
+at32_spi_setup_slaves(unsigned int bus_num, struct spi_board_info *b,
+                     unsigned int n, const u8 *pins)
+{
+       unsigned int pin, mode;
+
+       for (; n; n--, b++) {
+               b->bus_num = bus_num;
+               if (b->chip_select >= 4)
+                       continue;
```

```
+               pin = (unsigned)b->controller_data;
+               if (!pin) {
+                       pin = pins[b->chip_select];
+                       b->controller_data = (void *)pin;
+               }
+               mode = AT32_GPIOF_OUTPUT;
+               if (!(b->mode & SPI_CS_HIGH))
+                       mode |= AT32_GPIOF_HIGH;
+               at32_select_gpio(pin, mode);
+       }
+}
+
+struct platform_device *__init
+at32_add_device_spi(unsigned int id, struct spi_board_info *b, unsigned int n)
+{
+       /*
+        * Manage the chipselects as GPIOs, normally using the same pins
+        * the SPI controller expects; but boards can use other pins.
+        */
+       static u8 __initdata spi0_pins[] =
+               { GPIO_PIN_PA(10), GPIO_PIN_PA(8),
+                 GPIO_PIN_PA(9), GPIO_PIN_PA(7), };
+       static u8 __initdata spi1_pins[] =
+               { GPIO_PIN_PA(14), GPIO_PIN_PA(18),
+                 GPIO_PIN_PA(19), GPIO_PIN_PA(20), };
+       struct platform_device *pdev;
+
+       switch (id) {
+       case 0:
+               pdev = &atmel_spi0_device;
+               /* pullup MISO so a level is always defined */
+               select_peripheral(PA(11),  PERIPH_A, AT32_GPIOF_PULLUP);
+               select_peripheral(PA(12),  PERIPH_A, 0);        /* MOSI  */
+               select_peripheral(PA(13),  PERIPH_A, 0);        /* SCK   */
+               at32_spi_setup_slaves(0, b, n, spi0_pins);
+               break;
+
+       case 1:
+               pdev = &atmel_spi1_device;
+               /* pullup MISO so a level is always defined */
+               select_peripheral(PA(17),  PERIPH_B, AT32_GPIOF_PULLUP);
+               select_peripheral(PA(16),  PERIPH_B, 0);        /* MOSI  */
+               select_peripheral(PA(15),  PERIPH_B, 0);        /* SCK   */
+               at32_spi_setup_slaves(1, b, n, spi1_pins);
+               break;
+
+       default:
+               return NULL;
+       }
+
+       spi_register_board_info(b, n);
+       platform_device_register(pdev);
+       return pdev;
+}
+
+/* --------------------------------------------------------------------
+ *  TWI
+ * -------------------------------------------------------------------- */
+static struct resource atmel_twi0_resource[] __initdata = {
+       PBMEM(0xffff2c00),
+       IRQ(11),
+};
+static struct clk atmel_twi0_pclk = {
+       .name           = "twi_pclk",
+       .parent         = &pba_clk,
+       .mode           = pba_clk_mode,
+       .get_rate       = pba_clk_get_rate,
+       .index          = 7,
+};
+
+struct platform_device *__init at32_add_device_twi(unsigned int id,
+                                                  struct i2c_board_info *b,
+                                                  unsigned int n)
+{
+       struct platform_device *pdev;
+
+       if (id != 0)
+               return NULL;
+
+       pdev = platform_device_alloc("atmel_twi", id);
+       if (!pdev)
+               return NULL;
+
+       if (platform_device_add_resources(pdev, atmel_twi0_resource,
```

```
1265 +                               ARRAY_SIZE(atmel_twi0_resource)))
1266 +               goto err_add_resources;
1267 +
1268 +       select_peripheral(PA(6),  PERIPH_A, 0); /* SDA  */
1269 +       select_peripheral(PA(7),  PERIPH_A, 0); /* SDL  */
1270 +
1271 +       atmel_twi0_pclk.dev = &pdev->dev;
1272 +
1273 +       if (b)
1274 +               i2c_register_board_info(id, b, n);
1275 +
1276 +       platform_device_add(pdev);
1277 +       return pdev;
1278 +
1279 +err_add_resources:
1280 +       platform_device_put(pdev);
1281 +       return NULL;
1282 +}
1283 +
1284 +
1285 +/* --------------------------------------------------------------------
1286 + *  PWM
1287 + * -------------------------------------------------------------------- */
1288 +static struct resource atmel_pwm0_resource[] __initdata = {
1289 +       PBMEM(0xffff3000),
1290 +       IRQ(12),
1291 +};
1292 +static struct clk atmel_pwm0_mck = {
1293 +       .name           = "pwm_clk",
1294 +       .parent         = &pba_clk,
1295 +       .mode           = pba_clk_mode,
1296 +       .get_rate       = pba_clk_get_rate,
1297 +       .index          = 12,
1298 +};
1299 +
1300 +struct platform_device *__init at32_add_device_pwm(u32 mask)
1301 +{
1302 +       struct platform_device *pdev;
1303 +
1304 +       if (!mask)
1305 +               return NULL;
1306 +
1307 +       pdev = platform_device_alloc("atmel_pwm", 0);
1308 +       if (!pdev)
1309 +               return NULL;
1310 +
1311 +       if (platform_device_add_resources(pdev, atmel_pwm0_resource,
1312 +                                ARRAY_SIZE(atmel_pwm0_resource)))
1313 +               goto out_free_pdev;
1314 +
1315 +       if (platform_device_add_data(pdev, &mask, sizeof(mask)))
1316 +               goto out_free_pdev;
1317 +
1318 +       if (mask & (1 << 0))
1319 +               select_peripheral(PA(28), PERIPH_A, 0);
1320 +       if (mask & (1 << 1))
1321 +               select_peripheral(PA(29), PERIPH_A, 0);
1322 +       if (mask & (1 << 2))
1323 +               select_peripheral(PA(21), PERIPH_B, 0);
1324 +       if (mask & (1 << 3))
1325 +               select_peripheral(PA(22), PERIPH_B, 0);
1326 +
1327 +       atmel_pwm0_mck.dev = &pdev->dev;
1328 +
1329 +       platform_device_add(pdev);
1330 +
1331 +       return pdev;
1332 +
1333 +out_free_pdev:
1334 +       platform_device_put(pdev);
1335 +       return NULL;
1336 +}
1337 +
1338 +/* --------------------------------------------------------------------
1339 + *  SSC
1340 + * -------------------------------------------------------------------- */
1341 +static struct resource ssc0_resource[] = {
1342 +       PBMEM(0xffff3400),
1343 +       IRQ(13),
1344 +};
1345 +DEFINE_DEV(ssc, 0);
1346 +DEV_CLK(pclk, ssc0, pba, 13);
1347 +
1348 +struct platform_device *__init
```

```
+at32_add_device_ssc(unsigned int id, unsigned int flags)
+{
+       struct platform_device *pdev;
+
+       switch (id) {
+       case 0:
+               pdev = &ssc0_device;
+               if (flags & ATMEL_SSC_RF)
+                       select_peripheral(PA(21), PERIPH_A, 0); /* RF */
+               if (flags & ATMEL_SSC_RK)
+                       select_peripheral(PA(22), PERIPH_A, 0); /* RK */
+               if (flags & ATMEL_SSC_TK)
+                       select_peripheral(PA(23), PERIPH_A, 0); /* TK */
+               if (flags & ATMEL_SSC_TF)
+                       select_peripheral(PA(24), PERIPH_A, 0); /* TF */
+               if (flags & ATMEL_SSC_TD)
+                       select_peripheral(PA(25), PERIPH_A, 0); /* TD */
+               if (flags & ATMEL_SSC_RD)
+                       select_peripheral(PA(26), PERIPH_A, 0); /* RD */
+               break;
+       default:
+               return NULL;
+       }
+
+       platform_device_register(pdev);
+       return pdev;
+}
+
+/* --------------------------------------------------------------------
+ *  USB Device Controller
+ * -------------------------------------------------------------------- */
+static struct resource usba0_resource[] __initdata = {
+       {
+               .start          = 0xe0000000,
+               .end            = 0xefffffff,
+               .flags          = IORESOURCE_MEM,
+       }, {
+               .start          = 0xfffe0000,
+               .end            = 0xfffe0fff,
+               .flags          = IORESOURCE_MEM,
+       },
+       IRQ(17),
+};
+static struct clk usba0_pclk = {
+       .name           = "pclk",
+       .parent         = &pbb_clk,
+       .mode           = pbb_clk_mode,
+       .get_rate       = pbb_clk_get_rate,
+       .index          = 2,
+};
+static struct clk usba0_hclk = {
+       .name           = "hclk",
+       .parent         = &hsb_clk,
+       .mode           = hsb_clk_mode,
+       .get_rate       = hsb_clk_get_rate,
+       .index          = 3,
+};
+
+#define EP(nam, idx, maxpkt, maxbk, dma, isoc)                 \
+       [idx] = {                                               \
+               .name           = nam,                          \
+               .index          = idx,                          \
+               .fifo_size      = maxpkt,                       \
+               .nr_banks       = maxbk,                        \
+               .can_dma        = dma,                          \
+               .can_isoc       = isoc,                         \
+       }
+
+static struct usba_ep_data at32_usba_ep[] __initdata = {
+       EP("ep0",     0,   64, 1, 0, 0),
+       EP("ep1",     1,  512, 2, 1, 1),
+       EP("ep2",     2,  512, 2, 1, 1),
+       EP("ep3-int", 3,   64, 3, 1, 0),
+       EP("ep4-int", 4,   64, 3, 1, 0),
+       EP("ep5",     5, 1024, 3, 1, 1),
+       EP("ep6",     6, 1024, 3, 1, 1),
+};
+
+#undef EP
+
+struct platform_device *__init
+at32_add_device_usba(unsigned int id, struct usba_platform_data *data)
+{
+       /*
```

```
+         * pdata doesn't have room for any endpoints, so we need to
+         * append room for the ones we need right after it.
+         */
+        struct {
+                struct usba_platform_data pdata;
+                struct usba_ep_data ep[7];
+        } usba_data;
+        struct platform_device *pdev;
+
+        if (id != 0)
+                return NULL;
+
+        pdev = platform_device_alloc("atmel_usba_udc", 0);
+        if (!pdev)
+                return NULL;
+
+        if (platform_device_add_resources(pdev, usba0_resource,
+                                          ARRAY_SIZE(usba0_resource)))
+                goto out_free_pdev;
+
+        if (data)
+                usba_data.pdata.vbus_pin = data->vbus_pin;
+        else
+                usba_data.pdata.vbus_pin = -EINVAL;
+
+        data = &usba_data.pdata;
+        data->num_ep = ARRAY_SIZE(at32_usba_ep);
+        memcpy(data->ep, at32_usba_ep, sizeof(at32_usba_ep));
+
+        if (platform_device_add_data(pdev, data, sizeof(usba_data)))
+                goto out_free_pdev;
+
+        if (data->vbus_pin >= 0)
+                at32_select_gpio(data->vbus_pin, 0);
+
+        usba0_pclk.dev = &pdev->dev;
+        usba0_hclk.dev = &pdev->dev;
+
+        platform_device_add(pdev);
+
+        return pdev;
+
+out_free_pdev:
+        platform_device_put(pdev);
+        return NULL;
+}
+
+
+/* --------------------------------------------------------------------
+ *  GCLK
+ * -------------------------------------------------------------------- */
+static struct clk gclk0 = {
+        .name           = "gclk0",
+        .mode           = genclk_mode,
+        .get_rate       = genclk_get_rate,
+        .set_rate       = genclk_set_rate,
+        .set_parent     = genclk_set_parent,
+        .index          = 0,
+};
+
+struct clk *at32_clock_list[] = {
+        &osc32k,
+        &osc0,
+        &osc1,
+        &pll0,
+        &pll1,
+        &cpu_clk,
+        &hsb_clk,
+        &pba_clk,
+        &pbb_clk,
+        &at32_pm_pclk,
+        &at32_intc0_pclk,
+        &at32_hmatrix_clk,
+        &ebi_clk,
+        &sdramc_clk,
+        &smc0_pclk,
+        &smc0_mck,
+        &pdca_pclk,
+        &at32_ocd0_clk,
+        &gpio0_mck,
+        &gpio1_mck,
+        &gpio2_mck,
+        &gpio3_mck,
+        &at32_tc0_t0_clk,
```

```
+       &atmel_usart0_usart,
+       &atmel_usart1_usart,
+       &atmel_usart2_usart,
+       &atmel_usart3_usart,
+       &atmel_pwm0_mck,
+       &macb0_hclk,
+       &macb0_pclk,
+       &atmel_spi0_spi_clk,
+       &atmel_spi1_spi_clk,
+       &atmel_twi0_pclk,
+       &ssc0_pclk,
+       &usba0_hclk,
+       &usba0_pclk,
+       &gclk0,
+};
+unsigned int at32_nr_clocks = ARRAY_SIZE(at32_clock_list);
+
+void __init setup_platform(void)
+{
+       u32 cpu_mask = 0, hsb_mask = 0, pba_mask = 0, pbb_mask = 0;
+       int i;
+
+       if (pm_readl(MCCTRL) & PM_BIT(PLLSEL)) {
+               main_clock = &pll0;
+               cpu_clk.parent = &pll0;
+       } else {
+               main_clock = &osc0;
+               cpu_clk.parent = &osc0;
+       }
+
+       if (pm_readl(PLL0) & PM_BIT(PLLOSC))
+               pll0.parent = &osc1;
+       if (pm_readl(PLL1) & PM_BIT(PLLOSC))
+               pll1.parent = &osc1;
+
+       genclk_init_parent(&gclk0);
+
+       /*
+        * Turn on all clocks that have at least one user already, and
+        * turn off everything else. We only do this for module
+        * clocks, and even though it isn't particularly pretty to
+        * check the address of the mode function, it should do the
+        * trick...
+        */
+       for (i = 0; i < ARRAY_SIZE(at32_clock_list); i++) {
+               struct clk *clk = at32_clock_list[i];
+
+               if (clk->users == 0)
+                       continue;
+
+               if (clk->mode == &cpu_clk_mode)
+                       cpu_mask |= 1 << clk->index;
+               else if (clk->mode == &hsb_clk_mode)
+                       hsb_mask |= 1 << clk->index;
+               else if (clk->mode == &pba_clk_mode)
+                       pba_mask |= 1 << clk->index;
+               else if (clk->mode == &pbb_clk_mode)
+                       pbb_mask |= 1 << clk->index;
+       }
+
+       pm_writel(CPU_MASK, cpu_mask);
+       pm_writel(HSB_MASK, hsb_mask);
+       pm_writel(PBA_MASK, pba_mask);
+       pm_writel(PBB_MASK, pbb_mask);
+
+       /* Initialize the port muxes */
+       at32_init_gpio(&gpio0_device);
+       at32_init_gpio(&gpio1_device);
+       at32_init_gpio(&gpio2_device);
+       at32_init_gpio(&gpio3_device);
+}
+
+struct gen_pool *sram_pool;
+
+static int __init sram_init(void)
+{
+       struct gen_pool *pool;
+
+       /* 1KiB granularity */
+       pool = gen_pool_create(10, -1);
+       if (!pool)
+               goto fail;
+
+       /* All UC3A chips currently have at least 32 KiB of internal SRAM. */
```

```
+       if (gen_pool_add(pool, 0x00000000, 32*1024, -1))
+               goto err_pool_add;
+
+       sram_pool = pool;
+       return 0;
+
+err_pool_add:
+       gen_pool_destroy(pool);
+fail:
+       pr_err("Failed to create SRAM pool\n");
+       return -ENOMEM;
+}
+core_initcall(sram_init);
diff --git a/arch/avr32/mach-at32ap/clock.c b/arch/avr32/mach-at32uc3a/clock.c
similarity index 84%
copy from arch/avr32/mach-at32ap/clock.c
copy to arch/avr32/mach-at32uc3a/clock.c
index 138a00a..6c27dda 100644
--- a/arch/avr32/mach-at32ap/clock.c
+++ b/arch/avr32/mach-at32uc3a/clock.c
@@ -15,40 +15,24 @@
 #include <linux/err.h>
 #include <linux/device.h>
 #include <linux/string.h>
-#include <linux/list.h>

 #include <mach/chip.h>

 #include "clock.h"

-/* at32 clock list */
-static LIST_HEAD(at32_clock_list);
-
 static DEFINE_SPINLOCK(clk_lock);
-static DEFINE_SPINLOCK(clk_list_lock);
-
-void at32_clk_register(struct clk *clk)
-{
-       spin_lock(&clk_list_lock);
-       /* add the new item to the end of the list */
-       list_add_tail(&clk->list, &at32_clock_list);
-       spin_unlock(&clk_list_lock);
-}

 struct clk *clk_get(struct device *dev, const char *id)
 {
-       struct clk *clk;
+       int i;

-       spin_lock(&clk_list_lock);
+       for (i = 0; i < at32_nr_clocks; i++) {
+               struct clk *clk = at32_clock_list[i];

-       list_for_each_entry(clk, &at32_clock_list, list) {
-               if (clk->dev == dev && strcmp(id, clk->name) == 0) {
-                       spin_unlock(&clk_list_lock);
+               if (clk->dev == dev && strcmp(id, clk->name) == 0)
                        return clk;
-               }
        }

-       spin_unlock(&clk_list_lock);
        return ERR_PTR(-ENOENT);
 }
 EXPORT_SYMBOL(clk_get);
@@ -219,8 +203,8 @@ dump_clock(struct clk *parent, struct clkinf *r)

        /* cost of this scan is small, but not linear... */
        r->nest = nest + NEST_DELTA;
-
-       list_for_each_entry(clk, &at32_clock_list, list) {
+       for (i = 3; i < at32_nr_clocks; i++) {
+               clk = at32_clock_list[i];
                if (clk->parent == parent)
                        dump_clock(clk, r);
        }
@@ -231,7 +215,6 @@ static int clk_show(struct seq_file *s, void *unused)
 {
        struct clkinf   r;
        int             i;
-       struct clk      *clk;

        /* show all the power manager registers */
        seq_printf(s, "MCCTRL  = %8x\n", pm_readl(MCCTRL));
```

```
@@ -251,25 +234,14 @@ static int clk_show(struct seq_file *s, void *unused)

        seq_printf(s, "\n");

+       /* show clock tree as derived from the three oscillators
+        * we "know" are at the head of the list
+        */
        r.s = s;
        r.nest = 0;
-       /* protected from changes on the list while dumping */
-       spin_lock(&clk_list_lock);
-
-       /* show clock tree as derived from the three oscillators */
-       clk = clk_get(NULL, "osc32k");
-       dump_clock(clk, &r);
-       clk_put(clk);
-
-       clk = clk_get(NULL, "osc0");
-       dump_clock(clk, &r);
-       clk_put(clk);
-
-       clk = clk_get(NULL, "osc1");
-       dump_clock(clk, &r);
-       clk_put(clk);
-
-       spin_unlock(&clk_list_lock);
+       dump_clock(at32_clock_list[0], &r);
+       dump_clock(at32_clock_list[1], &r);
+       dump_clock(at32_clock_list[2], &r);

        return 0;
 }
diff --git a/arch/avr32/mach-at32ap/clock.h b/arch/avr32/mach-at32uc3a/clock.h
similarity index 88%
copy from arch/avr32/mach-at32ap/clock.h
copy to arch/avr32/mach-at32uc3a/clock.h
index 623bf0e..bb8e1f2 100644
--- a/arch/avr32/mach-at32ap/clock.h
+++ b/arch/avr32/mach-at32uc3a/clock.h
@@ -12,13 +12,8 @@
  * published by the Free Software Foundation.
  */
 #include <linux/clk.h>
-#include <linux/list.h>
-
-
-void at32_clk_register(struct clk *clk);

 struct clk {
-       struct list_head list;          /* linking element */
        const char      *name;          /* Clock name/function */
        struct device   *dev;           /* Device the clock is used by */
        struct clk      *parent;        /* Parent clock, if any */
@@ -30,3 +25,6 @@ struct clk {
        u16             users;          /* Enabled if non-zero */
        u16             index;          /* Sibling index */
 };
+
+extern struct clk *at32_clock_list[];
+extern unsigned int at32_nr_clocks;
diff --git a/arch/avr32/mach-at32ap/cpufreq.c b/arch/avr32/mach-at32uc3a/cpufreq.c
similarity index 86%
copy from arch/avr32/mach-at32ap/cpufreq.c
copy to arch/avr32/mach-at32uc3a/cpufreq.c
index 024c586..5dd8d25 100644
--- a/arch/avr32/mach-at32ap/cpufreq.c
+++ b/arch/avr32/mach-at32uc3a/cpufreq.c
@@ -40,9 +40,6 @@ static unsigned int at32_get_speed(unsigned int cpu)
        return (unsigned int)((clk_get_rate(cpuclk) + 500) / 1000);
 }

-static unsigned int    ref_freq;
-static unsigned long   loops_per_jiffy_ref;
-
 static int at32_set_target(struct cpufreq_policy *policy,
                           unsigned int target_freq,
                           unsigned int relation)
@@ -64,19 +61,8 @@ static int at32_set_target(struct cpufreq_policy *policy,
        freqs.cpu = 0;
        freqs.flags = 0;

-       if (!ref_freq) {
-               ref_freq = freqs.old;
-               loops_per_jiffy_ref = boot_cpu_data.loops_per_jiffy;
```

```
-       }
-
        cpufreq_notify_transition(&freqs, CPUFREQ_PRECHANGE);
-       if (freqs.old < freqs.new)
-               boot_cpu_data.loops_per_jiffy = cpufreq_scale(
-                               loops_per_jiffy_ref, ref_freq, freqs.new);
        clk_set_rate(cpuclk, freq);
-       if (freqs.new < freqs.old)
-               boot_cpu_data.loops_per_jiffy = cpufreq_scale(
-                               loops_per_jiffy_ref, ref_freq, freqs.new);
        cpufreq_notify_transition(&freqs, CPUFREQ_POSTCHANGE);

        pr_debug("cpufreq: set frequency %lu Hz\n", freq);
@@ -101,6 +87,7 @@ static int __init at32_cpufreq_driver_init(struct cpufreq_policy *policy)
        policy->cur = at32_get_speed(0);
        policy->min = policy->cpuinfo.min_freq;
        policy->max = policy->cpuinfo.max_freq;
+       policy->governor = CPUFREQ_DEFAULT_GOVERNOR;

        printk("cpufreq: AT32AP CPU frequency driver\n");

diff --git a/arch/avr32/mach-at32ap/extint.c b/arch/avr32/mach-at32uc3a/extint.c
similarity index 98%
copy from arch/avr32/mach-at32ap/extint.c
copy to arch/avr32/mach-at32uc3a/extint.c
index 310477b..c36a6d5 100644
--- a/arch/avr32/mach-at32ap/extint.c
+++ b/arch/avr32/mach-at32uc3a/extint.c
@@ -191,7 +191,7 @@ static int __init eic_probe(struct platform_device *pdev)
        struct eic *eic;
        struct resource *regs;
        unsigned int i;
-       unsigned int nr_of_irqs;
+       unsigned int nr_irqs;
        unsigned int int_irq;
        int ret;
        u32 pattern;
@@ -224,7 +224,7 @@ static int __init eic_probe(struct platform_device *pdev)
        eic_writel(eic, IDR, ~0UL);
        eic_writel(eic, MODE, ~0UL);
        pattern = eic_readl(eic, MODE);
-       nr_of_irqs = fls(pattern);
+       nr_irqs = fls(pattern);

        /* Trigger on low level unless overridden by driver */
        eic_writel(eic, EDGE, 0UL);
@@ -232,7 +232,7 @@ static int __init eic_probe(struct platform_device *pdev)

        eic->chip = &eic_chip;

-       for (i = 0; i < nr_of_irqs; i++) {
+       for (i = 0; i < nr_irqs; i++) {
                set_irq_chip_and_handler(eic->first_irq + i, &eic_chip,
                                         handle_level_irq);
                set_irq_chip_data(eic->first_irq + i, eic);
@@ -256,7 +256,7 @@ static int __init eic_probe(struct platform_device *pdev)
                 eic->regs, int_irq);
        dev_info(&pdev->dev,
                 "Handling %u external IRQs, starting with IRQ %u\n",
-                nr_of_irqs, eic->first_irq);
+                nr_irqs, eic->first_irq);

        return 0;

diff --git a/arch/avr32/mach-at32uc3a/gpio.c b/arch/avr32/mach-at32uc3a/gpio.c
new file mode 100644
index 0000000..0d3d4e6
--- /dev/null
+++ b/arch/avr32/mach-at32uc3a/gpio.c
@@ -0,0 +1,453 @@
+/*
+ * Atmel GPIO Port Multiplexer support
+ *
+ * Copyright (C) 2004-2006 Atmel Corporation
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the Free Software Foundation.
+ */
+
+#include <linux/clk.h>
+#include <linux/debugfs.h>
+#include <linux/fs.h>
+#include <linux/platform_device.h>
```

```
+#include <linux/irq.h>
+
+#include <asm/gpio.h>
+#include <asm/io.h>
+
+#include <mach/portmux.h>
+
+#include "gpio.h"
+
+#define MAX_NR_GPIO_DEVICES            5
+
+struct gpio_device {
+       struct gpio_chip chip;
+       void __iomem *regs;
+       const struct platform_device *pdev;
+       struct clk *clk;
+       u32 pinmux_mask;
+       char name[8];
+};
+
+static struct gpio_device gpio_dev[MAX_NR_GPIO_DEVICES];
+
+static struct gpio_device *gpio_pin_to_dev(unsigned int gpio_pin)
+{
+       struct gpio_device *gpio;
+       unsigned int index;
+
+       index = gpio_pin >> 5;
+       if (index >= MAX_NR_GPIO_DEVICES)
+               return NULL;
+       gpio = &gpio_dev[index];
+       if (!gpio->regs)
+               return NULL;
+
+       return gpio;
+}
+
+/* Pin multiplexing API */
+
+void __init at32_select_periph(unsigned int pin, unsigned int periph,
+                               unsigned long flags)
+{
+       struct gpio_device *gpio;
+       unsigned int pin_index = pin & 0x1f;
+       u32 mask = 1 << pin_index;
+
+       gpio = gpio_pin_to_dev(pin);
+       if (unlikely(!gpio)) {
+               printk("gpio: invalid pin %u\n", pin);
+               goto fail;
+       }
+
+       if (unlikely(test_and_set_bit(pin_index, &gpio->pinmux_mask)
+                    || gpiochip_is_requested(&gpio->chip, pin_index))) {
+               printk("%s: pin %u is busy\n", gpio->name, pin_index);
+               goto fail;
+       }
+
+       gpio_writel(gpio, PUERS, mask);
+       switch (periph) {
+       case 0:
+               gpio_writel(gpio, PMR0C, mask);
+               gpio_writel(gpio, PMR1C, mask);
+               break;
+       case 1:
+               gpio_writel(gpio, PMR0S, mask);
+               gpio_writel(gpio, PMR1C, mask);
+               break;
+       case 2:
+               gpio_writel(gpio, PMR0C, mask);
+               gpio_writel(gpio, PMR1S, mask);
+               break;
+       case 3:
+               gpio_writel(gpio, PMR0S, mask);
+               gpio_writel(gpio, PMR1S, mask);
+               break;
+       default:
+               printk("%s: invalid periphial %u\n", gpio->name, periph);
+               goto fail;
+       }
+
+       gpio_writel(gpio, GPERC, mask);
+       if (!(flags & AT32_GPIOF_PULLUP))
+               gpio_writel(gpio, PUERC, mask);
```

```
1937 +
1938 +       return;
1939 +
1940 +fail:
1941 +       dump_stack();
1942 +}
1943 +
1944 +void __init at32_select_gpio(unsigned int pin, unsigned long flags)
1945 +{
1946 +       struct gpio_device *gpio;
1947 +       unsigned int pin_index = pin & 0x1f;
1948 +       u32 mask = 1 << pin_index;
1949 +
1950 +       gpio = gpio_pin_to_dev(pin);
1951 +       if (unlikely(!gpio)) {
1952 +               printk("gpio: invalid pin %u\n", pin);
1953 +               goto fail;
1954 +       }
1955 +
1956 +       if (unlikely(test_and_set_bit(pin_index, &gpio->pinmux_mask))) {
1957 +               printk("%s: pin %u is busy\n", gpio->name, pin_index);
1958 +               goto fail;
1959 +       }
1960 +
1961 +       if (flags & AT32_GPIOF_OUTPUT) {
1962 +               if (flags & AT32_GPIOF_HIGH)
1963 +                       gpio_writel(gpio, OVRS, mask);
1964 +               else
1965 +                       gpio_writel(gpio, OVRC, mask);
1966 +
1967 +               if (flags & AT32_GPIOF_OPENDRAIN)
1968 +                       gpio_writel(gpio, ODMERS, mask);
1969 +               else
1970 +                       gpio_writel(gpio, ODMERC, mask);
1971 +
1972 +               gpio_writel(gpio, PUERC, mask);
1973 +               gpio_writel(gpio, ODERS, mask);
1974 +       } else {
1975 +               if (flags & AT32_GPIOF_PULLUP)
1976 +                       gpio_writel(gpio, PUERS, mask);
1977 +               else
1978 +                       gpio_writel(gpio, PUERC, mask);
1979 +
1980 +               if (flags & AT32_GPIOF_DEGLITCH)
1981 +                       gpio_writel(gpio, GFERS, mask);
1982 +               else
1983 +                       gpio_writel(gpio, GFERC, mask);
1984 +               gpio_writel(gpio, ODERC, mask);
1985 +       }
1986 +
1987 +       gpio_writel(gpio, GPERS, mask);
1988 +
1989 +       return;
1990 +
1991 +fail:
1992 +       dump_stack();
1993 +}
1994 +
1995 +/* Reserve a pin, preventing anyone else from changing its configuration. */
1996 +void __init at32_reserve_pin(unsigned int pin)
1997 +{
1998 +       struct gpio_device *gpio;
1999 +       unsigned int pin_index = pin & 0x1f;
2000 +
2001 +       gpio = gpio_pin_to_dev(pin);
2002 +       if (unlikely(!gpio)) {
2003 +               printk("gpio: invalid pin %u\n", pin);
2004 +               goto fail;
2005 +       }
2006 +
2007 +       if (unlikely(test_and_set_bit(pin_index, &gpio->pinmux_mask))) {
2008 +               printk("%s: pin %u is busy\n", gpio->name, pin_index);
2009 +               goto fail;
2010 +       }
2011 +
2012 +       return;
2013 +
2014 +fail:
2015 +       dump_stack();
2016 +}
2017 +
2018 +/*-------------------------------------------------------------------------*/
2019 +
2020 +/* GPIO API */
```

```
+
+static int direction_input(struct gpio_chip *chip, unsigned offset)
+{
+       struct gpio_device *gpio = container_of(chip, struct gpio_device, chip);
+       u32 mask = 1 << offset;
+
+       if (!(gpio_readl(gpio, GPER) & mask))
+               return -EINVAL;
+
+       gpio_writel(gpio, ODERC, mask);
+       return 0;
+}
+
+static int gpio_get(struct gpio_chip *chip, unsigned offset)
+{
+       struct gpio_device *gpio = container_of(chip, struct gpio_device, chip);
+
+       return (gpio_readl(gpio, PVR) >> offset) & 1;
+}
+
+static void gpio_set(struct gpio_chip *chip, unsigned offset, int value)
+{
+       struct gpio_device *gpio = container_of(chip, struct gpio_device, chip);
+       u32 mask = 1 << offset;
+
+       if (value)
+               gpio_writel(gpio, OVRS, mask);
+       else
+               gpio_writel(gpio, OVRC, mask);
+}
+
+static int direction_output(struct gpio_chip *chip, unsigned offset, int value)
+{
+       struct gpio_device *gpio = container_of(chip, struct gpio_device, chip);
+       u32 mask = 1 << offset;
+
+       if (!(gpio_readl(gpio, GPER) & mask))
+               return -EINVAL;
+
+       gpio_set(chip, offset, value);
+       gpio_writel(gpio, ODERS, mask);
+       return 0;
+}
+
+
+/*--------------------------------------------------------------------------*/
+
+/* GPIO IRQ support */
+
+static void gpio_irq_mask(unsigned irq)
+{
+       unsigned                gpio_pin = irq_to_gpio(irq);
+       struct gpio_device      *gpio = &gpio_dev[gpio_pin >> 5];
+
+       gpio_writel(gpio, IERC, 1 << (gpio_pin & 0x1f));
+}
+
+static void gpio_irq_unmask(unsigned irq)
+{
+       unsigned                gpio_pin = irq_to_gpio(irq);
+       struct gpio_device      *gpio = &gpio_dev[gpio_pin >> 5];
+
+       gpio_writel(gpio, IERS, 1 << (gpio_pin & 0x1f));
+}
+
+static int gpio_irq_type(unsigned irq, unsigned type)
+{
+       if (type != IRQ_TYPE_EDGE_BOTH && type != IRQ_TYPE_NONE)
+               return -EINVAL;
+
+       return 0;
+}
+
+static struct irq_chip gpio_irqchip = {
+       .name           = "gpio",
+       .mask           = gpio_irq_mask,
+       .unmask         = gpio_irq_unmask,
+       .set_type       = gpio_irq_type,
+};
+
+static void gpio_irq_handler(unsigned irq, struct irq_desc *desc)
+{
+       struct gpio_device      *gpio = get_irq_chip_data(irq);
+       unsigned                gpio_irq;
```

```
2105 +
2106 +       gpio_irq = (unsigned) get_irq_data(irq);
2107 +       for (;;) {
2108 +               u32             isr;
2109 +               struct irq_desc *d;
2110 +
2111 +               /* ack pending GPIO interrupts */
2112 +               isr = gpio_readl(gpio, IFR) & gpio_readl(gpio, IER);
2113 +               if (!isr)
2114 +                       break;
2115 +               gpio_writel(gpio, IFRC, isr);
2116 +               do {
2117 +                       int i;
2118 +
2119 +                       i = ffs(isr) - 1;
2120 +                       isr &= ~(1 << i);
2121 +
2122 +                       i += gpio_irq;
2123 +                       d = &irq_desc[i];
2124 +
2125 +                       d->handle_irq(i, d);
2126 +               } while (isr);
2127 +       }
2128 +}
2129 +
2130 +static void __init
2131 +gpio_irq_setup(struct gpio_device *gpio, int irq, int gpio_irq)
2132 +{
2133 +       unsigned        i;
2134 +
2135 +       set_irq_chip_data(irq, gpio);
2136 +       set_irq_data(irq, (void *) gpio_irq);
2137 +
2138 +       for (i = 0; i < 32; i++, gpio_irq++) {
2139 +               set_irq_chip_data(gpio_irq, gpio);
2140 +               set_irq_chip_and_handler(gpio_irq, &gpio_irqchip,
2141 +                               handle_simple_irq);
2142 +       }
2143 +
2144 +       set_irq_chained_handler(irq, gpio_irq_handler);
2145 +}
2146 +
2147 +/*-------------------------------------------------------------------------*/
2148 +
2149 +#ifdef CONFIG_DEBUG_FS
2150 +
2151 +#include <linux/seq_file.h>
2152 +
2153 +/*
2154 + * This shows more info than the generic gpio dump code:
2155 + * pullups, deglitching, open drain drive.
2156 + */
2157 +static void gpio_bank_show(struct seq_file *s, struct gpio_chip *chip)
2158 +{
2159 +       struct gpio_device *gpio = container_of(chip, struct gpio_device, chip);
2160 +       u32                     oder, ier, pvr, puer, gfer, odmer;
2161 +       unsigned                i;
2162 +       u32                     mask;
2163 +       char                    bank;
2164 +
2165 +       oder = gpio_readl(gpio, ODER);
2166 +       ier = gpio_readl(gpio, IER);
2167 +       pvr = gpio_readl(gpio, PVR);
2168 +       puer = gpio_readl(gpio, PUER);
2169 +       gfer = gpio_readl(gpio, GFER);
2170 +       odmer = gpio_readl(gpio, ODMER);
2171 +
2172 +       bank = 'A' + gpio->pdev->id;
2173 +
2174 +       for (i = 0, mask = 1; i < 32; i++, mask <<= 1) {
2175 +               const char *label;
2176 +
2177 +               label = gpiochip_is_requested(chip, i);
2178 +               if (!label && (imr & mask))
2179 +                       label = "[irq]";
2180 +               if (!label)
2181 +                       continue;
2182 +
2183 +               seq_printf(s, " gpio-%-3d P%c%-2d (%-12s) %s %s %s",
2184 +                       chip->base + i, bank, i,
2185 +                       label,
2186 +                       (oder & mask) ? "out" : "in ",
2187 +                       (pvr  & mask) ? "hi" : "lo",
2188 +                       (puer & mask) ? "   " : "up");
```

```
2189 +                if (gfer & mask)
2190 +                        seq_printf(s, " deglitch");
2191 +                if ((oder & odmer) & mask)
2192 +                        seq_printf(s, " open-drain");
2193 +                if (ier & mask)
2194 +                        seq_printf(s, " irq-%d edge-both",
2195 +                                gpio_to_irq(chip->base + i));
2196 +                seq_printf(s, "\n");
2197 +        }
2198 +}
2199 +
2200 +#else
2201 +#define gpio_bank_show NULL
2202 +#endif
2203 +
2204 +
2205 +/*--------------------------------------------------------------------------*/
2206 +
2207 +static int __init gpio_probe(struct platform_device *pdev)
2208 +{
2209 +        struct gpio_device *gpio = NULL;
2210 +        int irq = platform_get_irq(pdev, 0);
2211 +        int gpio_irq_base = GPIO_IRQ_BASE + pdev->id * 32;
2212 +
2213 +        BUG_ON(pdev->id >= MAX_NR_GPIO_DEVICES);
2214 +        gpio = &gpio_dev[pdev->id];
2215 +        BUG_ON(!gpio->regs);
2216 +
2217 +        gpio->chip.label = gpio->name;
2218 +        gpio->chip.base = pdev->id * 32;
2219 +        gpio->chip.ngpio = 32;
2220 +        gpio->chip.dev = &pdev->dev;
2221 +        gpio->chip.owner = THIS_MODULE;
2222 +
2223 +        gpio->chip.direction_input = direction_input;
2224 +        gpio->chip.get = gpio_get;
2225 +        gpio->chip.direction_output = direction_output;
2226 +        gpio->chip.set = gpio_set;
2227 +        gpio->chip.dbg_show = gpio_bank_show;
2228 +
2229 +        gpiochip_add(&gpio->chip);
2230 +
2231 +        gpio_irq_setup(gpio, irq, gpio_irq_base);
2232 +
2233 +        platform_set_drvdata(pdev, gpio);
2234 +
2235 +        printk(KERN_DEBUG "%s: base 0x%p, irq %d chains %d..%d\n",
2236 +               gpio->name, gpio->regs, irq, gpio_irq_base, gpio_irq_base + 31);
2237 +
2238 +        return 0;
2239 +}
2240 +
2241 +static struct platform_driver gpio_driver = {
2242 +        .probe          = gpio_probe,
2243 +        .driver         = {
2244 +                .name           = "gpio",
2245 +        },
2246 +};
2247 +
2248 +static int __init gpio_init(void)
2249 +{
2250 +        return platform_driver_register(&gpio_driver);
2251 +}
2252 +postcore_initcall(gpio_init);
2253 +
2254 +void __init at32_init_gpio(struct platform_device *pdev)
2255 +{
2256 +        struct resource *regs;
2257 +        struct gpio_device *gpio;
2258 +
2259 +        if (pdev->id > MAX_NR_GPIO_DEVICES) {
2260 +                dev_err(&pdev->dev, "only %d GPIO devices supported\n",
2261 +                        MAX_NR_GPIO_DEVICES);
2262 +                return;
2263 +        }
2264 +
2265 +        gpio = &gpio_dev[pdev->id];
2266 +        snprintf(gpio->name, sizeof(gpio->name), "gpio%d", pdev->id);
2267 +
2268 +        regs = platform_get_resource(pdev, IORESOURCE_MEM, 0);
2269 +        if (!regs) {
2270 +                dev_err(&pdev->dev, "no mmio resource defined\n");
2271 +                return;
2272 +        }
```

```
2273 +
2274 +       gpio->clk = clk_get(&pdev->dev, "mck");
2275 +       if (IS_ERR(gpio->clk))
2276 +               /*
2277 +                * This is a fatal error, but if we continue we might
2278 +                * be so lucky that we manage to initialize the
2279 +                * console and display this message...
2280 +                */
2281 +               dev_err(&pdev->dev, "no mck clock defined\n");
2282 +       else
2283 +               clk_enable(gpio->clk);
2284 +
2285 +       gpio->pdev = pdev;
2286 +       gpio->regs = ioremap(regs->start, regs->end - regs->start + 1);
2287 +
2288 +       /* start with irqs disabled and acked */
2289 +       gpio_writel(gpio, IERC, ~0UL);
2290 +       (void) gpio_readl(gpio, IER);
2291 +}
2292 diff --git a/arch/avr32/mach-at32uc3a/gpio.h b/arch/avr32/mach-at32uc3a/gpio.h
2293 new file mode 100644
2294 index 0000000..5016db9
2295 --- /dev/null
2296 +++ b/arch/avr32/mach-at32uc3a/gpio.h
2297 @@ -0,0 +1,77 @@
2298 +/*
2299 + * Atmel GPIO Port Multiplexer support
2300 + *
2301 + * Copyright (C) 2004-2006 Atmel Corporation
2302 + *
2303 + * This program is free software; you can redistribute it and/or modify
2304 + * it under the terms of the GNU General Public License version 2 as
2305 + * published by the Free Software Foundation.
2306 + */
2307 +#ifndef __ARCH_AVR32_AT32UC_GPIO_H__
2308 +#define __ARCH_AVR32_AT32UC_GPIO_H__
2309 +
2310 +/* PIO register offsets */
2311 +#define GPIO_GPER                              0x00
2312 +#define GPIO_GPERS                             0x04
2313 +#define GPIO_GPERC                             0x08
2314 +#define GPIO_GPERT                             0x0c
2315 +#define GPIO_PMR0                              0x10
2316 +#define GPIO_PMR0S                             0x14
2317 +#define GPIO_PMR0C                             0x18
2318 +#define GPIO_PMR0T                             0x1c
2319 +#define GPIO_PMR1                              0x20
2320 +#define GPIO_PMR1S                             0x24
2321 +#define GPIO_PMR1C                             0x28
2322 +#define GPIO_PMR1T                             0x2c
2323 +#define GPIO_ODER                              0x40
2324 +#define GPIO_ODERS                             0x44
2325 +#define GPIO_ODERC                             0x48
2326 +#define GPIO_ODERT                             0x4c
2327 +#define GPIO_OVR                               0x50
2328 +#define GPIO_OVRS                              0x54
2329 +#define GPIO_OVRC                              0x58
2330 +#define GPIO_OVRT                              0x5c
2331 +#define GPIO_PVR                               0x60
2332 +#define GPIO_PUER                              0x70
2333 +#define GPIO_PUERS                             0x74
2334 +#define GPIO_PUERC                             0x78
2335 +#define GPIO_PUERT                             0x7c
2336 +#define GPIO_ODMER                             0x80
2337 +#define GPIO_ODMERS                            0x84
2338 +#define GPIO_ODMERC                            0x88
2339 +#define GPIO_ODMERT                            0x8c
2340 +#define GPIO_IER                               0x90
2341 +#define GPIO_IERS                              0x94
2342 +#define GPIO_IERC                              0x98
2343 +#define GPIO_IERT                              0x9c
2344 +#define GPIO_IMR0                              0xa0
2345 +#define GPIO_IMR0S                             0xa4
2346 +#define GPIO_IMR0C                             0xa8
2347 +#define GPIO_IMR0T                             0xac
2348 +#define GPIO_IMR1                              0xb0
2349 +#define GPIO_IMR1S                             0xb4
2350 +#define GPIO_IMR1C                             0xb8
2351 +#define GPIO_IMR1T                             0xbc
2352 +#define GPIO_GFER                              0xc0
2353 +#define GPIO_GFERS                             0xc4
2354 +#define GPIO_GFERC                             0xc8
2355 +#define GPIO_GFERT                             0xcc
2356 +#define GPIO_IFR                               0xd0
```

```
+#define GPIO_IFRC                               0xd8
+
+
+/* Bit manipulation macros */
+#define GPIO_BIT(name)                          (1 << GPIO_##name##_OFFSET)
+#define GPIO_BF(name,value)                     (((value) & ((1 << GPIO_##name##_SIZE) - 1)) << GPIO_##name##_OFFSET)
+#define GPIO_BFEXT(name,value)                  (((value) >> GPIO_##name##_OFFSET) & ((1 << GPIO_##name##_SIZE) - 1))
+#define GPIO_BFINS(name,value,old)              (((old) & ~(((1 << GPIO_##name##_SIZE) - 1) << GPIO_##name##_OFFSET)) | GPIO_BF(name,value))
+
+/* Register access macros */
+#define gpio_readl(port,reg)                                    \
+       __raw_readl((port)->regs + GPIO_##reg)
+#define gpio_writel(port,reg,value)                             \
+       __raw_writel((value), (port)->regs + GPIO_##reg)
+
+void at32_init_gpio(struct platform_device *pdev);
+
+#endif /* __ARCH_AVR32_AT32UC_GPIO_H__ */
diff --git a/arch/avr32/mach-at32ap/hmatrix.c b/arch/avr32/mach-at32uc3a/hmatrix.c
similarity index 100%
copy from arch/avr32/mach-at32ap/hmatrix.c
copy to arch/avr32/mach-at32uc3a/hmatrix.c
diff --git a/arch/avr32/mach-at32ap/hsmc.c b/arch/avr32/mach-at32uc3a/hsmc.c
similarity index 100%
copy from arch/avr32/mach-at32ap/hsmc.c
copy to arch/avr32/mach-at32uc3a/hsmc.c
diff --git a/arch/avr32/mach-at32ap/hsmc.h b/arch/avr32/mach-at32uc3a/hsmc.h
similarity index 100%
copy from arch/avr32/mach-at32ap/hsmc.h
copy to arch/avr32/mach-at32uc3a/hsmc.h
diff --git a/arch/avr32/mach-at32uc3a/include/mach/at32uc3a0xxx.h b/arch/avr32/mach-at32uc3a/include/mach/at32uc3a0xxx.h
new file mode 100644
index 0000000..76783d0
--- /dev/null
+++ b/arch/avr32/mach-at32uc3a/include/mach/at32uc3a0xxx.h
@@ -0,0 +1,78 @@
+/*
+ * Pin definitions for AT32AP7000.
+ *
+ * Copyright (C) 2006 Atmel Corporation
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the Free Software Foundation.
+ */
+#ifndef __ASM_ARCH_AT32UC3A0XXX_H__
+#define __ASM_ARCH_AT32UC3A0XXX_H__
+
+#define GPIO_PERIPH_A  0
+#define GPIO_PERIPH_B  1
+#define GPIO_PERIPH_C  2
+
+/*
+ * Pin numbers identifying specific GPIO pins on the chip. They can
+ * also be converted to IRQ numbers by passing them through
+ * gpio_to_irq().
+ */
+#define GPIO_PIOA_BASE (0)
+#define GPIO_PIOB_BASE (GPIO_PIOA_BASE + 32)
+#define GPIO_PIOC_BASE (GPIO_PIOB_BASE + 32)
+#define GPIO_PIOD_BASE (GPIO_PIOC_BASE + 32)
+#define GPIO_PIOE_BASE (GPIO_PIOD_BASE + 32)
+
+#define GPIO_PIN_PA(N) (GPIO_PIOA_BASE + (N))
+#define GPIO_PIN_PB(N) (GPIO_PIOB_BASE + (N))
+#define GPIO_PIN_PC(N) (GPIO_PIOC_BASE + (N))
+#define GPIO_PIN_PD(N) (GPIO_PIOD_BASE + (N))
+#define GPIO_PIN_PE(N) (GPIO_PIOE_BASE + (N))
+
+
+/*
+ * DMAC peripheral hardware handshaking interfaces, used with dw_dmac
+ */
+#define DMAC_MCI_RX             0
+#define DMAC_MCI_TX             1
+#define DMAC_DAC_TX             2
+#define DMAC_AC97_A_RX          3
+#define DMAC_AC97_A_TX          4
+#define DMAC_AC97_B_RX          5
+#define DMAC_AC97_B_TX          6
```

```
+#define DMAC_DMAREQ_0          7
+#define DMAC_DMAREQ_1          8
+#define DMAC_DMAREQ_2          9
+#define DMAC_DMAREQ_3          10
+
+/* HSB masters */
+#define HMATRIX_MASTER_CPU_DATA                        0
+#define HMATRIX_MASTER_CPU_INSTRUCTIONS                1
+#define HMATRIX_MASTER_CPU_SAB                 2
+#define HMATRIX_MASTER_PDCA                    3
+#define HMATRIX_MASTER_MACB_DMA                        4
+#define HMATRIX_MASTER_USBB_DMA                        5
+
+/* HSB slaves */
+#define HMATRIX_SLAVE_INT_FLASH                        0
+#define HMATRIX_SLAVE_HSB_PB_BR0               1
+#define HMATRIX_SLAVE_HSB_PB_BR1               2
+#define HMATRIX_SLAVE_INT_SRAM                 3
+#define HMATRIX_SLAVE_USBB_DPRAM               4
+#define HMATRIX_SLAVE_EBI                      5
+
+
+/* Bits in HMATRIX SFR5 (EBI) */
+#define HMATRIX_EBI_SDRAM_ENABLE               (1 << 1)
+
+/*
+ * Base addresses of controllers that may be accessed early by
+ * platform code.
+ */
+#define PM_BASE                0xffff0c00
+#define HMATRIX_BASE   0xfffe1000
+#define SDRAMC_BASE    0xfffe2000
+
+#endif /* __ASM_ARCH_AT32UC3A0XXX_H__ */
diff --git a/arch/avr32/mach-at32ap/include/mach/board.h b/arch/avr32/mach-at32uc3a/include/mach/board.h
similarity index 93%
copy from arch/avr32/mach-at32ap/include/mach/board.h
copy to arch/avr32/mach-at32uc3a/include/mach/board.h
index aafaf7a..e60e907 100644
--- a/arch/avr32/mach-at32ap/include/mach/board.h
+++ b/arch/avr32/mach-at32uc3a/include/mach/board.h
@@ -14,14 +14,8 @@
   */
 extern unsigned long at32_board_osc_rates[];

-/*
- * This used to add essential system devices, but this is now done
- * automatically. Please don't use it in new board code.
- */
-static inline void __deprecated at32_add_system_devices(void)
-{
-
-}
+/* Add basic devices: system manager, interrupt controller, portmuxes, etc. */
+void at32_add_system_devices(void);

 #define ATMEL_MAX_UART 4
 extern struct platform_device *atmel_default_console_device;
@@ -49,7 +43,7 @@ struct atmel_lcdfb_info;
 struct platform_device *
 at32_add_device_lcdc(unsigned int id, struct atmel_lcdfb_info *data,
                      unsigned long fbmem_start, unsigned long fbmem_len,
-                     u64 pin_mask);
+                     unsigned int pin_config);

 struct usba_platform_data;
 struct platform_device *
diff --git a/arch/avr32/mach-at32ap/include/mach/chip.h b/arch/avr32/mach-at32uc3a/include/mach/chip.h
similarity index 87%
copy from arch/avr32/mach-at32ap/include/mach/chip.h
copy to arch/avr32/mach-at32uc3a/include/mach/chip.h
index 5efca6d..b29e191 100644
--- a/arch/avr32/mach-at32ap/include/mach/chip.h
+++ b/arch/avr32/mach-at32uc3a/include/mach/chip.h
@@ -12,6 +12,8 @@

 #if defined(CONFIG_CPU_AT32AP700X)
 # include <mach/at32ap700x.h>
+#elif defined(CONFIG_CPU_AT32UC3A0XXX)
+# include <mach/at32uc3a0xxx.h>
 #else
 # error Unknown chip type selected
 #endif
diff --git a/arch/avr32/mach-at32ap/include/mach/cpu.h b/arch/avr32/mach-at32uc3a/include/mach/cpu.h
```

```
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/cpu.h
copy to arch/avr32/mach-at32uc3a/include/mach/cpu.h
diff --git a/arch/avr32/mach-at32ap/include/mach/gpio.h b/arch/avr32/mach-at32uc3a/include/mach/gpio.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/gpio.h
copy to arch/avr32/mach-at32uc3a/include/mach/gpio.h
diff --git a/arch/avr32/mach-at32ap/include/mach/hmatrix.h b/arch/avr32/mach-at32uc3a/include/mach/
    hmatrix.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/hmatrix.h
copy to arch/avr32/mach-at32uc3a/include/mach/hmatrix.h
diff --git a/arch/avr32/mach-at32ap/include/mach/init.h b/arch/avr32/mach-at32uc3a/include/mach/init.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/init.h
copy to arch/avr32/mach-at32uc3a/include/mach/init.h
diff --git a/arch/avr32/mach-at32ap/include/mach/io.h b/arch/avr32/mach-at32uc3a/include/mach/io.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/io.h
copy to arch/avr32/mach-at32uc3a/include/mach/io.h
diff --git a/arch/avr32/mach-at32ap/include/mach/irq.h b/arch/avr32/mach-at32uc3a/include/mach/irq.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/irq.h
copy to arch/avr32/mach-at32uc3a/include/mach/irq.h
diff --git a/arch/avr32/mach-at32ap/include/mach/pm.h b/arch/avr32/mach-at32uc3a/include/mach/pm.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/pm.h
copy to arch/avr32/mach-at32uc3a/include/mach/pm.h
diff --git a/arch/avr32/mach-at32ap/include/mach/portmux.h b/arch/avr32/mach-at32uc3a/include/mach/
    portmux.h
similarity index 79%
copy from arch/avr32/mach-at32ap/include/mach/portmux.h
copy to arch/avr32/mach-at32uc3a/include/mach/portmux.h
index 21c7937..ae3b9df 100644
--- a/arch/avr32/mach-at32ap/include/mach/portmux.h
+++ b/arch/avr32/mach-at32uc3a/include/mach/portmux.h
@@ -19,12 +19,11 @@
 #define AT32_GPIOF_OUTPUT      0x00000002      /* (OUT) Enable output driver */
 #define AT32_GPIOF_HIGH                0x00000004      /* (OUT) Set output high */
 #define AT32_GPIOF_DEGLITCH    0x00000008      /* (IN) Filter glitches */
-#define AT32_GPIOF_MULTIDRV    0x00000010      /* Enable multidriver option */
+#define AT32_GPIOF_OPENDRAIN   0x00000010      /* Enable open drain mode option */

-void at32_select_periph(unsigned int port, unsigned int pin,
-                        unsigned int periph, unsigned long flags);
+void at32_select_periph(unsigned int pin, unsigned int periph,
+                        unsigned long flags);
 void at32_select_gpio(unsigned int pin, unsigned long flags);
-void at32_deselect_pin(unsigned int pin);
 void at32_reserve_pin(unsigned int pin);

 #endif /* __ASM_ARCH_PORTMUX_H__ */
diff --git a/arch/avr32/mach-at32ap/include/mach/smc.h b/arch/avr32/mach-at32uc3a/include/mach/smc.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/smc.h
copy to arch/avr32/mach-at32uc3a/include/mach/smc.h
diff --git a/arch/avr32/mach-at32ap/include/mach/sram.h b/arch/avr32/mach-at32uc3a/include/mach/sram.h
similarity index 100%
copy from arch/avr32/mach-at32ap/include/mach/sram.h
copy to arch/avr32/mach-at32uc3a/include/mach/sram.h
diff --git a/arch/avr32/mach-at32ap/intc.c b/arch/avr32/mach-at32uc3a/intc.c
similarity index 100%
copy from arch/avr32/mach-at32ap/intc.c
copy to arch/avr32/mach-at32uc3a/intc.c
diff --git a/arch/avr32/mach-at32ap/intc.h b/arch/avr32/mach-at32uc3a/intc.h
similarity index 100%
copy from arch/avr32/mach-at32ap/intc.h
copy to arch/avr32/mach-at32uc3a/intc.h
diff --git a/arch/avr32/mach-at32ap/pdc.c b/arch/avr32/mach-at32uc3a/pdca.c
similarity index 75%
copy from arch/avr32/mach-at32ap/pdc.c
copy to arch/avr32/mach-at32uc3a/pdca.c
index 61ab15a..17a48e1 100644
--- a/arch/avr32/mach-at32ap/pdc.c
+++ b/arch/avr32/mach-at32uc3a/pdca.c
@@ -11,7 +11,7 @@
 #include <linux/init.h>
 #include <linux/platform_device.h>

-static int __init pdc_probe(struct platform_device *pdev)
+static int __init pdca_probe(struct platform_device *pdev)
 {
        struct clk *pclk, *hclk;

```

```
@@ -34,14 +34,15 @@ static int __init pdc_probe(struct platform_device *pdev)
        return 0;
 }

-static struct platform_driver pdc_driver = {
+static struct platform_driver pdca_driver = {
+       .probe          = pdca_probe,
        .driver         = {
-               .name   = "pdc",
+               .name   = "pdca",
        },
 };

-static int __init pdc_init(void)
+static int __init pdca_init(void)
 {
-       return platform_driver_probe(&pdc_driver, pdc_probe);
+       return platform_driver_register(&pdca_driver);
 }
-arch_initcall(pdc_init);
+arch_initcall(pdca_init);
diff --git a/arch/avr32/mach-at32ap/pm-at32ap700x.S b/arch/avr32/mach-at32uc3a/pm-at32uc3a0xxx.S
similarity index 100%
copy from arch/avr32/mach-at32ap/pm-at32ap700x.S
copy to arch/avr32/mach-at32uc3a/pm-at32uc3a0xxx.S
diff --git a/arch/avr32/mach-at32ap/pm.c b/arch/avr32/mach-at32uc3a/pm.c
similarity index 100%
copy from arch/avr32/mach-at32ap/pm.c
copy to arch/avr32/mach-at32uc3a/pm.c
diff --git a/arch/avr32/mach-at32ap/pm.h b/arch/avr32/mach-at32uc3a/pm.h
similarity index 100%
copy from arch/avr32/mach-at32ap/pm.h
copy to arch/avr32/mach-at32uc3a/pm.h
diff --git a/arch/avr32/mach-at32ap/sdramc.h b/arch/avr32/mach-at32uc3a/sdramc.h
similarity index 100%
copy from arch/avr32/mach-at32ap/sdramc.h
copy to arch/avr32/mach-at32uc3a/sdramc.h
```

D.29 Board support for ATEVK1100

```
commit 6677f489f529d76a17c5ab6900f81dbcfbc8b5d1
Author: Gunnar Rangoy <gunnar@rangoy.com>
Date:   Tue May 5 14:23:43 2009 +0200

    AVR32: Board support for ATEVK1100

diff --git a/arch/avr32/Kconfig b/arch/avr32/Kconfig
index 631d388..fcec5a1 100644
--- a/arch/avr32/Kconfig
+++ b/arch/avr32/Kconfig
@@ -155,6 +155,10 @@ config BOARD_FAVR_32
 config BOARD_MIMC200
        bool "MIMC200 CPU board"
        select CPU_AT32AP7000
+
+config BOARD_ATEVK1100
+       bool "ATEVK1100 Evaluation Kit"
+       select CPU_AT32UC3A0XXX
 endchoice

 source "arch/avr32/boards/atstk1000/Kconfig"
diff --git a/arch/avr32/Makefile b/arch/avr32/Makefile
index ad1dd87..0a8c3eb 100644
--- a/arch/avr32/Makefile
+++ b/arch/avr32/Makefile
@@ -51,6 +51,7 @@ core-$(CONFIG_BOARD_ATSTK1000)              += arch/avr32/boards/atstk1000/
 core-$(CONFIG_BOARD_ATNGW100)          += arch/avr32/boards/atngw100/
 core-$(CONFIG_BOARD_FAVR_32)           += arch/avr32/boards/favr-32/
 core-$(CONFIG_BOARD_MIMC200)           += arch/avr32/boards/mimc200/
+core-$(CONFIG_BOARD_ATEVK1100)         += arch/avr32/boards/atevk1100/
 core-$(CONFIG_LOADER_U_BOOT)           += arch/avr32/boot/u-boot/
 core-y                                 += arch/avr32/kernel/
 core-y                                 += arch/avr32/mm/
diff --git a/arch/avr32/boards/atevk1100/Makefile b/arch/avr32/boards/atevk1100/Makefile
new file mode 100644
index 0000000..beee577
--- /dev/null
+++ b/arch/avr32/boards/atevk1100/Makefile
@@ -0,0 +1 @@
```

```
+obj-y                          += setup.o
diff --git a/arch/avr32/boards/atevk1100/setup.c b/arch/avr32/boards/atevk1100/setup.c
new file mode 100644
index 0000000..4eba3de
--- /dev/null
+++ b/arch/avr32/boards/atevk1100/setup.c
@@ -0,0 +1,121 @@
+/*
+ * Board-specific setup code for the ATEVK1100 Evaluation Kit
+ *
+ * Copyright (C) 2005-2006 Atmel Corporation
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the Free Software Foundation.
+ */
+#include <linux/clk.h>
+#include <linux/etherdevice.h>
+#include <linux/irq.h>
+#include <linux/i2c.h>
+#include <linux/i2c-gpio.h>
+#include <linux/init.h>
+#include <linux/linkage.h>
+#include <linux/platform_device.h>
+#include <linux/types.h>
+#include <linux/leds.h>
+#include <linux/spi/spi.h>
+
+#include <asm/atmel-mci.h>
+#include <asm/io.h>
+#include <asm/setup.h>
+
+#include <mach/at32uc3a0xxx.h>
+#include <mach/board.h>
+#include <mach/init.h>
+#include <mach/portmux.h>
+
+/* Oscillator frequencies. These are board-specific */
+unsigned long at32_board_osc_rates[3] = {
+       [0] = 32768,    /* 32.768 kHz on RTC osc */
+       [1] = 12000000, /* 12 MHz on osc0 */
+       [2] = 0,
+};
+/* Initialized by bootloader-specific startup code. */
+struct tag *bootloader_tags __initdata;
+
+static struct eth_platform_data __initdata eth_data = {
+       .is_rmii        = 1,
+};
+
+static struct spi_board_info spi0_board_info[] __initdata = {
+       {
+               .modalias       = "mtd_dataflash",
+               .max_speed_hz   = 8000000,
+               .chip_select    = 0,
+       },
+};
+
+void __init setup_board(void)
+{
+       at32_map_usart(0, 0);   /* USART 0: /dev/ttyS0, DB9 */
+       at32_setup_serial_console(0);
+}
+
+static const struct gpio_led evk1100_leds[] = {
+       { .name = "led1", .gpio = GPIO_PIN_PB(27), .active_low = 1,
+               .default_trigger = "heartbeat",
+       },
+       { .name = "led2", .gpio = GPIO_PIN_PB(28), .active_low = 1, },
+       /* Disabled, as it sits on the CS2, which is used for SRAM.
+       { .name = "led3", .gpio = GPIO_PIN_PB(29), .active_low = 1, },
+       */
+       { .name = "led4", .gpio = GPIO_PIN_PB(30), .active_low = 1, },
+       { .name = "led5r", .gpio = GPIO_PIN_PB(19), .active_low = 1, },
+       { .name = "led5g", .gpio = GPIO_PIN_PB(20), .active_low = 1, },
+       { .name = "led6r", .gpio = GPIO_PIN_PB(21), .active_low = 1, },
+       { .name = "led6g", .gpio = GPIO_PIN_PB(22), .active_low = 1, },
+};
+
+static const struct gpio_led_platform_data evk1100_led_data = {
+       .num_leds =     ARRAY_SIZE(evk1100_leds),
+       .leds =         (void *) evk1100_leds,
+};
+
```

```
+static struct platform_device evk1100_gpio_leds = {
+       .name =         "leds-gpio",
+       .id =           -1,
+       .dev = {
+               .platform_data = (void *) &evk1100_led_data,
+       }
+};
+
+static int __init atevk1100_init(void)
+{
+       unsigned        i;
+
+       /*
+        * atevk1100 uses 16-bit SDRAM interface, so we don't need to
+        * reserve any pins for it.
+        */
+
+       at32_add_system_devices();
+
+       at32_add_device_usart(0);
+
+       at32_add_device_eth(0, &eth_data);
+
+       at32_add_device_spi(0, spi0_board_info, ARRAY_SIZE(spi0_board_info));
+       at32_add_device_usba(0, NULL);
+
+       for (i = 0; i < ARRAY_SIZE(evk1100_leds); i++) {
+               at32_select_gpio(evk1100_leds[i].gpio,
+                                AT32_GPIOF_OUTPUT | AT32_GPIOF_HIGH);
+       }
+       platform_device_register(&evk1100_gpio_leds);
+
+       return 0;
+}
+postcore_initcall(atevk1100_init);
+
+static int __init atevk1100_arch_init(void)
+{
+       /* set_irq_type() after the arch_initcall for EIC has run, and
+        * before the I2C subsystem could try using this IRQ.
+        */
+       return set_irq_type(AT32_EXTINT(3), IRQ_TYPE_EDGE_FALLING);
+}
+arch_initcall(atevk1100_arch_init);
```

Appendix E

PDCA, SPI and DataFlash support

```
diff --git a/arch/avr32/boards/atevk1100/setup.c b/arch/avr32/boards/atevk1100/setup.c
index 5d8dca0..8505b45 100644
--- a/arch/avr32/boards/atevk1100/setup.c
+++ b/arch/avr32/boards/atevk1100/setup.c
@@ -46,11 +46,17 @@ static struct eth_platform_data __initdata eth_data = {
 };

 static struct spi_board_info spi0_board_info[] __initdata = {
+
+};
+
+static struct spi_board_info spi1_board_info[] __initdata = {
+  /*
         {
                 .modalias       = "mtd_dataflash",
                 .max_speed_hz   = 8000000,
                 .chip_select    = 0,
         },
+  */
 };

 /*
@@ -162,6 +168,7 @@ static int __init atevk1100_init(void)
         set_hw_addr(at32_add_device_eth(0, &eth_data));

         at32_add_device_spi(0, spi0_board_info, ARRAY_SIZE(spi0_board_info));
+        at32_add_device_spi(1, spi1_board_info, ARRAY_SIZE(spi1_board_info));
         at32_add_device_usba(0, NULL);

         for (i = 0; i < ARRAY_SIZE(evk1100_leds); i++) {
diff --git a/arch/avr32/mach-at32ap/include/mach/pdma.h b/arch/avr32/mach-at32ap/include/mach/pdma.h
new file mode 100644
index 0000000..b798dc7
--- /dev/null
+++ b/arch/avr32/mach-at32ap/include/mach/pdma.h
@@ -0,0 +1,9 @@
+/*
+ * Peripheral DMA abstraction layer for AP7000.
+ */
+#ifndef __ASM_ARCH_PDMA_H__
+#define __ASM_ARCH_PDMA_H__
+
+
+
+#endif /* __ASM_ARCH_PDMA_H__ */
diff --git a/arch/avr32/mach-at32uc3a/at32uc3a0xxx.c b/arch/avr32/mach-at32uc3a/at32uc3a0xxx.c
index f6af725..f488ef9 100644
--- a/arch/avr32/mach-at32uc3a/at32uc3a0xxx.c
+++ b/arch/avr32/mach-at32uc3a/at32uc3a0xxx.c
@@ -23,6 +23,7 @@
 #include <mach/at32uc3a0xxx.h>
 #include <mach/board.h>
 #include <mach/hmatrix.h>
+#include <mach/pdma.h>
 #include <mach/portmux.h>
 #include <mach/sram.h>
```

```
57
58 @@ -55,6 +56,18 @@
59                 .name           = _name,                \
60                 .flags          = IORESOURCE_IRQ,       \
61         }
62 +#define PDCA_RX(num)                                    \
63 +       {                                               \
64 +               .start          = num,                  \
65 +               .end            = num,                  \
66 +               .flags          = IORESOURCE_PDCA_RX,   \
67 +       }
68 +#define PDCA_TX(num)                                    \
69 +       {                                               \
70 +               .start          = num,                  \
71 +               .end            = num,                  \
72 +               .flags          = IORESOURCE_PDCA_TX,   \
73 +       }
74
75  /* REVISIT these assume *every* device supports DMA, but several
76   * don't ... tc, smc, pio, rtc, watchdog, pwm, ps2, and more.
77 @@ -707,11 +720,17 @@ static struct clk smc0_mck = {
78         .index          = 6,
79  };
80
81 +static struct resource pdca_resource[] = {
82 +       PBMEM(0xffff0000),
83 +};
84  static struct platform_device pdca_device = {
85         .name           = "pdca",
86         .id             = 0,
87 +       .resource       = pdca_resource,                \
88 +       .num_resources  = ARRAY_SIZE(pdca_resource),    \
89  };
90  DEV_CLK(pclk, pdca, pba, 2);
91 +DEV_CLK(hclk, pdca, hsb, 5);
92
93  /* --------------------------------------------------------------------
94   * HMATRIX
95 @@ -1011,6 +1030,8 @@ at32_add_device_eth(unsigned int id, struct eth_platform_data *data)
96  static struct resource atmel_spi0_resource[] = {
97         PBMEM(0xffff2400),
98         IRQ(9),
99 +       PDCA_RX(7),
100 +       PDCA_TX(15),
101  };
102  DEFINE_DEV(atmel_spi, 0);
103  DEV_CLK(spi_clk, atmel_spi0, pba, 5);
104 @@ -1018,6 +1039,8 @@ DEV_CLK(spi_clk, atmel_spi0, pba, 5);
105  static struct resource atmel_spi1_resource[] = {
106         PBMEM(0xffff2800),
107         IRQ(10),
108 +       PDCA_RX(8),
109 +       PDCA_TX(16),
110  };
111  DEFINE_DEV(atmel_spi, 1);
112  DEV_CLK(spi_clk, atmel_spi1, pba, 6);
113 @@ -1362,6 +1385,7 @@ struct clk *at32_clock_list[] = {
114         &smc0_pclk,
115         &smc0_mck,
116         &pdca_pclk,
117 +       &pdca_hclk,
118         &at32_ocd0_clk,
119         &gpio0_mck,
120         &gpio1_mck,
121 diff --git a/arch/avr32/mach-at32uc3a/gpio.c b/arch/avr32/mach-at32uc3a/gpio.c
122 index 62600f6..c6c9f9f 100644
123 --- a/arch/avr32/mach-at32uc3a/gpio.c
124 +++ b/arch/avr32/mach-at32uc3a/gpio.c
125 @@ -21,7 +21,11 @@
126
127  #include "gpio.h"
128
129 -#define MAX_NR_GPIO_DEVICES            5
130 +#include <mach/chip.h>
131 +#include <mach/pm.h>
132 +#include "pm.h"
133 +
134 +#define MAX_NR_GPIO_DEVICES            4
135
136  struct gpio_device {
137         struct gpio_chip chip;
138 @@ -138,7 +142,7 @@ void __init at32_select_gpio(unsigned int pin, unsigned long flags)
139                         gpio_writel(gpio, PUERS, mask);
140                 else
```

```
                        gpio_writel(gpio, PUERC, mask);
-
+
                if (flags & AT32_GPIOF_DEGLITCH)
                        gpio_writel(gpio, GFERS, mask);
                else
@@ -207,8 +211,15 @@ static void gpio_set(struct gpio_chip *chip, unsigned offset, int value)

        if (value)
                gpio_writel(gpio, OVRS, mask);
-       else
-               gpio_writel(gpio, OVRC, mask);
+       else {
+               void *a;
+               u32 v;
+
+               a = gpio->regs + GPIO_OVRC;
+               v = mask;
+
+               __raw_writel(v, a);
+       }
 }

 static int direction_output(struct gpio_chip *chip, unsigned offset, int value)
@@ -446,6 +457,9 @@ void __init at32_init_gpio(struct platform_device *pdev)

        gpio->pdev = pdev;
        gpio->regs = ioremap(regs->start, regs->end - regs->start + 1);
+       if (!gpio->regs) {
+               dev_err(&pdev->dev, "unable to map memory (%p, %u)\n", (void *)regs->start, regs->end -
    regs->start + 1);
+       }

        /* start with irqs disabled and acked */
        gpio_writel(gpio, IERC, ~0UL);
diff --git a/arch/avr32/mach-at32uc3a/include/mach/pdca.h b/arch/avr32/mach-at32uc3a/include/mach/pdca.h
new file mode 100644
index 0000000..072e119
--- /dev/null
+++ b/arch/avr32/mach-at32uc3a/include/mach/pdca.h
@@ -0,0 +1,35 @@
+/*
+ * PDCA registers and definitions.
+ */
+#ifndef __ASM_ARCH_PDCA_H__
+#define __ASM_ARCH_PDCA_H__
+
+#include <linux/io.h>
+
+#define PDCA_MAR               0x00
+#define PDCA_PSR               0x04
+#define PDCA_TCR               0x08
+#define PDCA_MARR              0x0c
+#define PDCA_TCRR              0x10
+#define PDCA_CR                        0x14
+#define PDCA_MR                        0x18
+#define PDCA_SR                        0x1c
+#define PDCA_SLOT_SIZE         0x40
+
+/* Bits in CR */
+#define PDCA_CR_TEN            0x00000001
+#define PDCA_CR_TDIS           0x00000002
+#define PDCA_CR_ECLR           0x00000100
+
+/* Bits in SR */
+#define PDCA_SR_TEN            0x00000001
+
+extern void __iomem *pdca_regs;
+
+#define pdca_readl(slot, reg) \
+  __raw_readl(pdca_regs + slot * PDCA_SLOT_SIZE + PDCA_##reg)
+#define pdca_writel(slot, reg, value) \
+  __raw_writel((value), pdca_regs + slot * PDCA_SLOT_SIZE + PDCA_##reg)
+
+
+#endif /* __ASM_ARCH_PDCA_H__ */
diff --git a/arch/avr32/mach-at32uc3a/include/mach/pdma.h b/arch/avr32/mach-at32uc3a/include/mach/pdma.h
new file mode 100644
index 0000000..b74a2ce
--- /dev/null
+++ b/arch/avr32/mach-at32uc3a/include/mach/pdma.h
@@ -0,0 +1,43 @@
+/*
+ * Peripheral DMA abstraction layer for UC3A.
```

```
+ */
+#ifndef __ASM_ARCH_PDMA_H__
+#define __ASM_ARCH_PDMA_H__
+
+#include <linux/ioport.h>
+#include <mach/pdca.h>
+
+/* Resource types for PDCA RX peripheral ID and PDCA TX peripheral id. */
+#define IORESOURCE_PDCA_RX     0x00000e00
+#define IORESOURCE_PDCA_TX     0x00000f00
+
+struct pdma_channel {
+       /*
+        * The PDCA slotss we have allocated for RX & TX,
+        * or -1 if no slot is allocated for that purpose.
+        */
+       int rx_slot;
+       int tx_slot;
+};
+
+int pdma_init(struct pdma_channel *channel, struct platform_device *pdev);
+void pdma_release(struct pdma_channel *channel);
+
+void pdma_set_rx(struct pdma_channel *channel, dma_addr_t addr, u32 counter);
+void pdma_set_next_rx(struct pdma_channel *channel, dma_addr_t addr, u32 counter);
+void pdma_set_tx(struct pdma_channel *channel, dma_addr_t addr, u32 counter);
+void pdma_set_next_tx(struct pdma_channel *channel, dma_addr_t addr, u32 counter);
+
+void pdma_enable_rx(struct pdma_channel *channel);
+void pdma_disable_rx(struct pdma_channel *channel);
+int pdma_rx_enabled(struct pdma_channel *channel);
+
+void pdma_enable_tx(struct pdma_channel *channel);
+void pdma_disable_tx(struct pdma_channel *channel);
+int pdma_tx_enabled(struct pdma_channel *channel);
+
+u32 pdma_get_rx_counter(struct pdma_channel *channel);
+u32 pdma_get_tx_counter(struct pdma_channel *channel);
+
+
+#endif /* __ASM_ARCH_PDMA_H__ */
diff --git a/arch/avr32/mach-at32uc3a/pdca.c b/arch/avr32/mach-at32uc3a/pdca.c
index 17a48e1..2ff7cfb 100644
--- a/arch/avr32/mach-at32uc3a/pdca.c
+++ b/arch/avr32/mach-at32uc3a/pdca.c
@@ -6,15 +6,194 @@
  * published by the Free Software Foundation.
  */

+#include <linux/bitops.h>
 #include <linux/clk.h>
 #include <linux/err.h>
 #include <linux/init.h>
+#include <linux/mutex.h>
 #include <linux/platform_device.h>
+#include <mach/pdma.h>
+
+void __iomem *pdca_regs;
+
+static DEFINE_MUTEX(pdca_lock);
+static unsigned long allocated_slots;
+
+#define PDCA_SLOTS 15
+
+static int allocate_slot(void)
+{
+       int slot;
+
+       mutex_lock(&pdca_lock);
+
+       slot = ffz(allocated_slots);
+       if (slot >= PDCA_SLOTS) {
+               slot = -ENOSPC;
+               printk(KERN_ERR "No free pdca slots.\n");
+       } else {
+               __set_bit(slot, &allocated_slots);
+       }
+
+       mutex_unlock(&pdca_lock);
+
+       return slot;
+}
+
+static void free_slot(int slot)
```

```
+{
+       BUG_ON(slot < 0 || slot >= PDCA_SLOTS);
+
+       mutex_lock(&pdca_lock);
+
+       __clear_bit(slot, &allocated_slots);
+
+       mutex_unlock(&pdca_lock);
+}
+
+int pdma_init(struct pdma_channel *channel, struct platform_device *pdev)
+{
+       int ret;
+       struct resource *pdca_rx;
+       struct resource *pdca_tx;
+
+       pdca_rx = platform_get_resource(pdev, IORESOURCE_PDCA_RX, 0);
+       pdca_tx = platform_get_resource(pdev, IORESOURCE_PDCA_TX, 0);
+
+       if (pdca_rx) {
+               ret = allocate_slot();
+               if (ret < 0)
+                       goto out_no_rx_slot;
+               channel->rx_slot = ret;
+               pdca_writel(channel->rx_slot, PSR, pdca_rx->start);
+
+       } else {
+               channel->rx_slot = -1;
+       }
+
+       if (pdca_tx) {
+               ret = allocate_slot();
+               if (ret < 0)
+                       goto out_no_tx_slot;
+               channel->tx_slot = ret;
+               pdca_writel(channel->tx_slot, PSR, pdca_tx->start);
+
+       } else {
+               channel->tx_slot = -1;
+       }
+
+       return 0;
+
+ out_no_tx_slot:
+       if (channel->rx_slot != -1) {
+               free_slot(channel->rx_slot);
+       }
+ out_no_rx_slot:
+       return ret;
+}
+
+void pdma_release(struct pdma_channel *channel)
+{
+       if (channel->rx_slot != -1) {
+               BUG_ON(pdma_rx_enabled(channel));
+               free_slot(channel->rx_slot);
+       }
+
+       if (channel->tx_slot != -1) {
+               BUG_ON(pdma_tx_enabled(channel));
+               free_slot(channel->tx_slot);
+       }
+
+}
+
+void pdma_set_rx(struct pdma_channel *channel, dma_addr_t addr, u32 counter)
+{
+       BUG_ON(channel->rx_slot == -1);
+       BUG_ON(counter > 0xffff);
+       pdca_writel(channel->rx_slot, MAR, addr);
+       pdca_writel(channel->rx_slot, TCR, counter);
+
+}
+void pdma_set_next_rx(struct pdma_channel *channel, dma_addr_t addr, u32 counter)
+{
+       BUG_ON(channel->rx_slot == -1);
+       BUG_ON(counter > 0xffff);
+       pdca_writel(channel->rx_slot, MARR, addr);
+       pdca_writel(channel->rx_slot, TCRR, counter);
+
+}
+void pdma_set_tx(struct pdma_channel *channel, dma_addr_t addr, u32 counter)
+{
+       BUG_ON(channel->tx_slot == -1);
```

```
+	BUG_ON(counter > 0xffff);
+	pdca_writel(channel->tx_slot, MAR, addr);
+	pdca_writel(channel->tx_slot, TCR, counter);
+
+}
+void pdma_set_next_tx(struct pdma_channel *channel, dma_addr_t addr, u32 counter)
+{
+	BUG_ON(channel->tx_slot == -1);
+	BUG_ON(counter > 0xffff);
+	pdca_writel(channel->tx_slot, MARR, addr);
+	pdca_writel(channel->tx_slot, TCRR, counter);
+
+}
+
+void pdma_enable_rx(struct pdma_channel *channel)
+{
+	BUG_ON(channel->rx_slot == -1);
+	pdca_writel(channel->rx_slot, CR, PDCA_CR_TEN);
+}
+void pdma_disable_rx(struct pdma_channel *channel)
+{
+	BUG_ON(channel->rx_slot == -1);
+	pdca_writel(channel->rx_slot, CR, PDCA_CR_TDIS);
+}
+int pdma_rx_enabled(struct pdma_channel *channel)
+{
+	BUG_ON(channel->rx_slot == -1);
+	return !!(pdca_readl(channel->rx_slot, SR) & PDCA_SR_TEN);
+}
+
+void pdma_enable_tx(struct pdma_channel *channel)
+{
+	BUG_ON(channel->tx_slot == -1);
+	pdca_writel(channel->tx_slot, CR, PDCA_CR_TEN);
+}
+void pdma_disable_tx(struct pdma_channel *channel)
+{
+	BUG_ON(channel->tx_slot == -1);
+	pdca_writel(channel->tx_slot, CR, PDCA_CR_TDIS);
+}
+int pdma_tx_enabled(struct pdma_channel *channel)
+{
+	BUG_ON(channel->tx_slot == -1);
+	return !!(pdca_readl(channel->tx_slot, SR) & PDCA_SR_TEN);
+}
+
+u32 pdma_get_rx_counter(struct pdma_channel *channel)
+{
+	BUG_ON(channel->rx_slot == -1);
+	return pdca_readl(channel->rx_slot, TCR);
+}
+
+u32 pdma_get_tx_counter(struct pdma_channel *channel)
+{
+	BUG_ON(channel->tx_slot == -1);
+	return pdca_readl(channel->tx_slot, TCR);
+}

 static int __init pdca_probe(struct platform_device *pdev)
 {
+	struct resource *regs;
	struct clk *pclk, *hclk;

+	regs = platform_get_resource(pdev, IORESOURCE_MEM, 0);
+	if (!regs) {
+		dev_err(&pdev->dev, "no memory defined\n");
+		return -ENXIO;
+	}
+
	pclk = clk_get(&pdev->dev, "pclk");
	if (IS_ERR(pclk)) {
		dev_err(&pdev->dev, "no pclk defined\n");
@@ -27,6 +206,10 @@ static int __init pdca_probe(struct platform_device *pdev)
		return PTR_ERR(hclk);
	}

+	pdca_regs = ioremap(regs->start, regs->end - regs->start + 1);
+	if (!pdca_regs)
+		return  -ENOMEM;
+
	clk_enable(pclk);
	clk_enable(hclk);

diff --git a/drivers/mtd/devices/mtd_dataflash.c b/drivers/mtd/devices/mtd_dataflash.c
```

```
index 6dd9aff..653eea2 100644
--- a/drivers/mtd/devices/mtd_dataflash.c
+++ b/drivers/mtd/devices/mtd_dataflash.c
@@ -867,6 +867,7 @@ static int __devinit dataflash_probe(struct spi_device *spi)
         * capacity using bits in the status byte.
         */
        status = dataflash_status(spi);
+
        if (status <= 0 || status == 0xff) {
                DEBUG(MTD_DEBUG_LEVEL1, "%s: status error %d\n",
                                spi->dev.bus_id, status);
diff --git a/drivers/spi/atmel_spi.c b/drivers/spi/atmel_spi.c
index 8abae4a..f942f98 100644
--- a/drivers/spi/atmel_spi.c
+++ b/drivers/spi/atmel_spi.c
@@ -7,6 +7,8 @@
  * it under the terms of the GNU General Public License version 2 as
  * published by the Free Software Foundation.
  */
+#define DEBUG
+#define VERBOSE

 #include <linux/kernel.h>
 #include <linux/init.h>
@@ -23,9 +25,11 @@
 #include <mach/board.h>
 #include <mach/gpio.h>
 #include <mach/cpu.h>
+#include <mach/pdma.h>

 #include "atmel_spi.h"

+
 /*
  * The core SPI transfer engine just talks to a register bank to set up
  * DMA transfers; transfer queue progress is driven by IRQs.  The clock
@@ -43,6 +47,7 @@ struct atmel_spi {

        void __iomem            *regs;
        int                     irq;
+       struct pdma_channel     dma;
        struct clk              *clk;
        struct platform_device  *pdev;
        unsigned                new_1:1;
@@ -107,6 +112,7 @@ static void cs_activate(struct atmel_spi *as, struct spi_device *spi)

        if (!(cpu_is_at91rm9200() && spi->chip_select == 0))
                gpio_set_value(gpio, active);
+
        spi_writel(as, MR, mr);
 }

@@ -198,19 +204,18 @@ static void atmel_spi_next_xfer(struct spi_master *master,
                xfer = NULL;

        if (xfer) {
-               spi_writel(as, PTCR, SPI_BIT(RXTDIS) | SPI_BIT(TXTDIS));
+               pdma_disable_rx(&as->dma);
+               pdma_disable_tx(&as->dma);

                len = xfer->len;
                atmel_spi_next_xfer_data(master, xfer, &tx_dma, &rx_dma, &len);
                remaining = xfer->len - len;

-               spi_writel(as, RPR, rx_dma);
-               spi_writel(as, TPR, tx_dma);
-
                if (msg->spi->bits_per_word > 8)
                        len >>= 1;
-               spi_writel(as, RCR, len);
-               spi_writel(as, TCR, len);
+
+               pdma_set_rx(&as->dma, rx_dma, len);
+               pdma_set_tx(&as->dma, tx_dma, len);

                dev_dbg(&msg->spi->dev,
                                "  start xfer %p: len %u tx %p/%08x rx %p/%08x\n",
@@ -243,13 +248,11 @@ static void atmel_spi_next_xfer(struct spi_master *master,
                atmel_spi_next_xfer_data(master, xfer, &tx_dma, &rx_dma, &len);
                as->next_remaining_bytes = total - len;

-               spi_writel(as, RNPR, rx_dma);
-               spi_writel(as, TNPR, tx_dma);
-
```

```
560                 if (msg->spi->bits_per_word > 8)
561                         len >>= 1;
562 -               spi_writel(as, RNCR, len);
563 -               spi_writel(as, TNCR, len);
564 +
565 +               pdma_set_next_rx(&as->dma, rx_dma, len);
566 +               pdma_set_next_tx(&as->dma, tx_dma, len);
567
568                 dev_dbg(&msg->spi->dev,
569                         "  next xfer %p: len %u tx %p/%08x rx %p/%08x\n",
570 @@ -257,8 +260,9 @@ static void atmel_spi_next_xfer(struct spi_master *master,
571                         xfer->rx_buf, xfer->rx_dma);
572                 ieval = SPI_BIT(ENDRX) | SPI_BIT(OVRES);
573         } else {
574 -               spi_writel(as, RNCR, 0);
575 -               spi_writel(as, TNCR, 0);
576 +               pdma_set_next_rx(&as->dma, NULL, 0);
577 +               pdma_set_next_tx(&as->dma, NULL, 0);
578 +
579                 ieval = SPI_BIT(RXBUFF) | SPI_BIT(ENDRX) | SPI_BIT(OVRES);
580         }
581
582 @@ -273,7 +277,8 @@ static void atmel_spi_next_xfer(struct spi_master *master,
583          * It should be doable, though. Just not now...
584          */
585         spi_writel(as, IER, ieval);
586 -       spi_writel(as, PTCR, SPI_BIT(TXTEN) | SPI_BIT(RXTEN));
587 +       pdma_enable_rx(&as->dma);
588 +       pdma_enable_tx(&as->dma);
589  }
590
591  static void atmel_spi_next_message(struct spi_master *master)
592 @@ -372,9 +377,11 @@ atmel_spi_msg_done(struct spi_master *master, struct atmel_spi *as,
593         as->current_transfer = NULL;
594         as->next_transfer = NULL;
595
596 -       /* continue if needed */
597 -       if (list_empty(&as->queue) || as->stopping)
598 -               spi_writel(as, PTCR, SPI_BIT(RXTDIS) | SPI_BIT(TXTDIS));
599 +       /* Continue if needed */
600 +       if (list_empty(&as->queue) || as->stopping){
601 +               pdma_disable_rx(&as->dma);
602 +               pdma_disable_tx(&as->dma);
603 +       }
604         else
605                 atmel_spi_next_message(master);
606  }
607 @@ -390,7 +397,6 @@ atmel_spi_interrupt(int irq, void *dev_id)
608         int                     ret = IRQ_NONE;
609
610         spin_lock(&as->lock);
611 -
612         xfer = as->current_transfer;
613         msg = list_entry(as->queue.next, struct spi_message, queue);
614
615 @@ -400,7 +406,6 @@ atmel_spi_interrupt(int irq, void *dev_id)
616
617         if (pending & SPI_BIT(OVRES)) {
618                 int timeout;
619 -
620                 ret = IRQ_HANDLED;
621
622                 spi_writel(as, IDR, (SPI_BIT(RXBUFF) | SPI_BIT(ENDRX)
623 @@ -417,7 +422,9 @@ atmel_spi_interrupt(int irq, void *dev_id)
624                  *
625                  * First, stop the transfer and unmap the DMA buffers.
626                  */
627 -               spi_writel(as, PTCR, SPI_BIT(RXTDIS) | SPI_BIT(TXTDIS));
628 +               pdma_disable_rx(&as->dma);
629 +               pdma_disable_tx(&as->dma);
630 +
631                 if (!msg->is_dma_mapped)
632                         atmel_spi_dma_unmap_xfer(master, xfer);
633
634 @@ -426,16 +433,17 @@ atmel_spi_interrupt(int irq, void *dev_id)
635                         udelay(xfer->delay_usecs);
636
637                 dev_warn(master->dev.parent, "overrun (%u/%u remaining)\n",
638 -                        spi_readl(as, TCR), spi_readl(as, RCR));
639 +                        pdma_get_tx_counter(&as->dma), pdma_get_rx_counter(&as->dma));
640
641                 /*
642                  * Clean up DMA registers and make sure the data
643                  * registers are empty.
```

```
644                  */
645 -                spi_writel(as, RNCR, 0);
646 -                spi_writel(as, TNCR, 0);
647 -                spi_writel(as, RCR, 0);
648 -                spi_writel(as, TCR, 0);
649 +                pdma_set_next_rx(&as->dma, NULL, 0);
650 +                pdma_set_next_tx(&as->dma, NULL, 0);
651 +                pdma_set_rx(&as->dma, NULL, 0);
652 +                pdma_set_tx(&as->dma, NULL, 0);
653 +
654                  for (timeout = 1000; timeout; timeout--)
655                          if (spi_readl(as, SR) & SPI_BIT(TXEMPTY))
656                                  break;
657 @@ -763,12 +771,18 @@ static int __init atmel_spi_probe(struct platform_device *pdev)
658          if (ret)
659                  goto out_unmap_regs;
660
661 +        ret = pdma_init(&as->dma, pdev);
662 +        if (ret)
663 +                goto out_free_irq;
664 +
665          /* Initialize the hardware */
666          clk_enable(clk);
667          spi_writel(as, CR, SPI_BIT(SWRST));
668          spi_writel(as, CR, SPI_BIT(SWRST)); /* AT91SAM9263 Rev B workaround */
669          spi_writel(as, MR, SPI_BIT(MSTR) | SPI_BIT(MODFDIS));
670 -        spi_writel(as, PTCR, SPI_BIT(RXTDIS) | SPI_BIT(TXTDIS));
671 +        pdma_disable_rx(&as->dma);
672 +        pdma_disable_tx(&as->dma);
673 +
674          spi_writel(as, CR, SPI_BIT(SPIEN));
675
676          /* go! */
677 @@ -785,6 +799,8 @@ out_reset_hw:
678          spi_writel(as, CR, SPI_BIT(SWRST));
679          spi_writel(as, CR, SPI_BIT(SWRST)); /* AT91SAM9263 Rev B workaround */
680          clk_disable(clk);
681 +        pdma_release(&as->dma);
682 +out_free_irq:
683          free_irq(irq, master);
684  out_unmap_regs:
685          iounmap(as->regs);
686 @@ -823,6 +839,7 @@ static int __exit atmel_spi_remove(struct platform_device *pdev)
687          dma_free_coherent(&pdev->dev, BUFFER_SIZE, as->buffer,
688                          as->buffer_dma);
689
690 +        pdma_release(&as->dma);
691          clk_disable(as->clk);
692          clk_put(as->clk);
693          free_irq(as->irq, master);
694 diff --git a/drivers/spi/atmel_spi.h b/drivers/spi/atmel_spi.h
695 index 6e06b6a..95dbc0c 100644
696 --- a/drivers/spi/atmel_spi.h
697 +++ b/drivers/spi/atmel_spi.h
698 @@ -23,16 +23,6 @@
699  #define SPI_CSR1                                0x0034
700  #define SPI_CSR2                                0x0038
701  #define SPI_CSR3                                0x003c
702 -#define SPI_RPR                                         0x0100
703 -#define SPI_RCR                                         0x0104
704 -#define SPI_TPR                                         0x0108
705 -#define SPI_TCR                                         0x010c
706 -#define SPI_RNPR                                0x0110
707 -#define SPI_RNCR                                0x0114
708 -#define SPI_TNPR                                0x0118
709 -#define SPI_TNCR                                0x011c
710 -#define SPI_PTCR                                0x0120
711 -#define SPI_PTSR                                0x0124
712
713  /* Bitfields in CR */
714  #define SPI_SPIEN_OFFSET                        0
```

Appendix F

Toolchain patches

F.1 Coverletter

```
From 4912c9e615f5c2fee55838e4895004b3149f08f8 Mon Sep 17 00:00:00 2001
Date: Tue, 26 May 2009 16:21:02 +0200
Subject: [PATCH] Toolchain support for AVR32A UC3 Linux programs

These patches make it possible to compile Linux programs for AVR32A UC3.
A lot of work still remains, but they actually work.
We are able to use the toolchain to compile BusyBox.

What works:
* Compiling statically linked FDPIC ELF programs.

What should be done/What does not work:
* Shared library support
* Some cleanup
* Linking in some program (e.g. BusyBox) does not work entierly correct.
  The PT_GNU_STACKSIZE not always copied.
* Probably other bugs in the code


This patches are developed during a master thesis at NTNU. We hope that
someone else can use them as a starting point for getting full support
for FDPIC ELF into the avr32 toolchain.


We made changes to the following tools:
* GCC-4.2.2-atmel.1.1.3
* Binutils-2.18 with patches from buildroot-avr32-v2.3.0
* uClibc-0.9.30


We attached the script we used to build the toolchain, to show which
options we used to compile the various tools. We have also attached
the script we used to build BusyBox.
```

F.2 GCC changes

```
From 70e27ba6eacd938d86ae366b930b37dd784f364d Mon Sep 17 00:00:00 2001
Date: Tue, 26 May 2009 15:08:26 +0200
Subject: [PATCH] GCC: Add support for FDPIC ELF for AVR32.

This patch makes a few changes to GCC, mostly to add support for the
 -mfdpic flag. There were also a few changes to crti.asm, to prevent it
from replacing the got-pointer during _init and _fini.

Unfortunately, we haven't found a way to compile several variants of
crti.o from crti.asm, so that a single GCC can be used for both fdpic
and normal compiles.

To compile gcc for fdpic, make must be invoked like:
make CFLAGS_FOR_TARGET=-mfdpic

This makes crti.asm compile with __AVR32_FDPIC__ defined.
```

```
---
 gcc/config/avr32/avr32.opt   |    3 +++
 gcc/config/avr32/crti.asm    |    4 ++++
 gcc/config/avr32/linux-elf.h |   15 +++++++++++++-
 3 files changed, 21 insertions(+), 1 deletions(-)

diff --git a/gcc/config/avr32/avr32.opt b/gcc/config/avr32/avr32.opt
index a9a1d5a..d4c62f3 100644
--- a/gcc/config/avr32/avr32.opt
+++ b/gcc/config/avr32/avr32.opt
@@ -84,3 +84,6 @@ Target Report Mask(RMW_ADDRESSABLE_DATA)
 Signal that all data is in range for the Atomic Read-Modify-Write memory instructions, and that
 gcc can safely generate these whenever possible.

+mfdpic
+Target Report Mask(FDPIC)
+Enable Function Descriptor PIC mode
diff --git a/gcc/config/avr32/crti.asm b/gcc/config/avr32/crti.asm
index 4c31f49..634adc3 100644
--- a/gcc/config/avr32/crti.asm
+++ b/gcc/config/avr32/crti.asm
@@ -40,6 +40,7 @@
        .global _init
 _init:
        stm     --sp, r6, lr
+#ifndef __AVR32_FDPIC__
        lddpc   r6, 1f
 0:
        rsub    r6, pc
@@ -47,6 +48,7 @@ _init:
        .align  2
 1:     .long   0b - _GLOBAL_OFFSET_TABLE_
 2:
+#endif /* __AVR32_FDPIC__ */

        .section        ".fini"
 /* Just load the GOT */
@@ -54,6 +56,7 @@ _init:
        .global _fini
 _fini:
        stm     --sp, r6, lr
+#ifndef __AVR32_FDPIC__
        lddpc   r6, 1f
 0:
        rsub    r6, pc
@@ -61,4 +64,5 @@ _fini:
        .align  2
 1:     .long   0b - _GLOBAL_OFFSET_TABLE_
 2:
+#endif /* __AVR32_FDPIC__ */

diff --git a/gcc/config/avr32/linux-elf.h b/gcc/config/avr32/linux-elf.h
index b3223fb..cb206a1 100644
--- a/gcc/config/avr32/linux-elf.h
+++ b/gcc/config/avr32/linux-elf.h
@@ -67,11 +67,22 @@
 #define ENDFILE_SPEC \
   "%{!shared:crtend.o%s} %{shared:crtendS.o%s} crtn.o%s"

+#define DRIVER_SELF_SPECS "\
+ %{mfdpic:%{!fpic:%{!fpie:%{!fPIC:%{!fPIE:\
+            %{!fno-pic:%{!fno-pie:%{!fno-PIC:%{!fno-PIE:-fpie}}}}}}}}} \
+"
+
 #undef ASM_SPEC
-#define ASM_SPEC "%{!mno-pic:%{!fno-pic:--pic}} %{mrelax|O*:%{mno-relax|O0|O1: ;:--linkrelax}} %{mcpu=*
    :-mcpu=%*}"
+#define ASM_SPEC "\
+   %{!mno-pic:%{!fno-pic:--pic}} \
+   %{mrelax|O*:%{mno-relax|O0|O1: ;:--linkrelax}} \
+   %{mcpu=*:-mcpu=%*} \
+   %{mfdpic} \
+"

 #undef  LINK_SPEC
 #define LINK_SPEC "%{version:-v} \
+   %{mfdpic:-mavr32linuxfdpic} \
    %{static:-Bstatic} \
    %{shared:-shared} \
    %{symbolic:-Bsymbolic} \
@@ -122,6 +133,8 @@
         builtin_define ("__AVR32_HAS_BRANCH_PRED__");           \
       if (TARGET_FAST_FLOAT)                                     \
         builtin_define ("__AVR32_FAST_FLOAT__");                 \
```

```
+      if (TARGET_FDPIC)                                                \
+        builtin_define ("__AVR32_FDPIC__");                    \
     }                                                          \
   while (0)

--
1.5.4.3
```

F.3 GNU binutils changes

```
From 4912c9e615f5c2fee55838e4895004b3149f08f8 Mon Sep 17 00:00:00 2001
Date: Tue, 26 May 2009 16:21:02 +0200
Subject: [PATCH] Binutils support for FDPIC ELF on AVR32 UC3

This patch adds support for statically linked FDPIC ELF targets on
AVR32. It mostly works, but there is a lack of error checking on input
file types, which means that if the linker is invoked incorrectly,
it will fail in strange ways.

For example, if one fails to specify -I elf32-avr32fdpic to
strip/objcopy, it will pretend that the file is a normal
elf32-avr32 file, and "ruin" the PT_GNU_STACK program header.

Some functions are (almost) direct copies from elf32-bfin.c and
elf32-frv.c, which are two architectures with FDPIC support. The code
for creating the .rofixup-section is however mostly new.
---
 bfd/config.bfd                   |    2 +
 bfd/configure                    |    1 +
 bfd/configure.in                 |    1 +
 bfd/elf32-avr32.c                |  343 ++++++++++++++++++++++++++++++++++++++
 bfd/targets.c                    |    2 +
 gas/config/tc-avr32.c            |    8 +
 include/elf/avr32.h              |    1 +
 ld/Makefile.am                   |    5 +
 ld/Makefile.in                   |    5 +
 ld/configure.tgt                 |    4 +-
 ld/emulparams/avr32linux.sh      |    1 +
 ld/emulparams/avr32linuxfdpic.sh |   10 +
 12 files changed, 382 insertions(+), 1 deletions(-)
 create mode 100644 ld/emulparams/avr32linuxfdpic.sh

diff --git a/bfd/config.bfd b/bfd/config.bfd
index 90350d7..193fd37 100644
--- a/bfd/config.bfd
+++ b/bfd/config.bfd
@@ -337,6 +337,8 @@ case "${targ}" in

   avr32-*-*)
     targ_defvec=bfd_elf32_avr32_vec
+    targ_selvecs=bfd_elf32_avr32fdpic_vec
+    targ_underscore=yes
     ;;

   c30-*-*aout* | tic30-*-*aout*)
diff --git a/bfd/configure b/bfd/configure
index 92d10b8..90a32d7 100755
--- a/bfd/configure
+++ b/bfd/configure
@@ -19042,6 +19042,7 @@ do
     bfd_elf32_am33lin_vec)     tb="$tb elf32-am33lin.lo elf32.lo $elf" ;;
     bfd_elf32_avr_vec)         tb="$tb elf32-avr.lo elf32.lo $elf" ;;
     bfd_elf32_avr32_vec)       tb="$tb elf32-avr32.lo elf32.lo $elf" ;;
+    bfd_elf32_avr32fdpic_vec)  tb="$tb elf32-avr32.lo elf32.lo $elf" ;;
     bfd_elf32_bfin_vec)                tb="$tb elf32-bfin.lo elf32.lo $elf" ;;
     bfd_elf32_bfinfdpic_vec)   tb="$tb elf32-bfin.lo elf32.lo $elf" ;;
     bfd_elf32_big_generic_vec)         tb="$tb elf32-gen.lo elf32.lo $elf" ;;
diff --git a/bfd/configure.in b/bfd/configure.in
index 0b0cd92..4e5102b 100644
--- a/bfd/configure.in
+++ b/bfd/configure.in
@@ -620,6 +620,7 @@ do
     bfd_elf32_am33lin_vec)     tb="$tb elf32-am33lin.lo elf32.lo $elf" ;;
     bfd_elf32_avr_vec)         tb="$tb elf32-avr.lo elf32.lo $elf" ;;
     bfd_elf32_avr32_vec)       tb="$tb elf32-avr32.lo elf32.lo $elf" ;;
+    bfd_elf32_avr32fdpic_vec)  tb="$tb elf32-avr32.lo elf32.lo $elf" ;;
     bfd_elf32_bfin_vec)                tb="$tb elf32-bfin.lo elf32.lo $elf" ;;
     bfd_elf32_bfinfdpic_vec)   tb="$tb elf32-bfin.lo elf32.lo $elf" ;;
     bfd_elf32_big_generic_vec)         tb="$tb elf32-gen.lo elf32.lo $elf" ;;
```

```
diff --git a/bfd/elf32-avr32.c b/bfd/elf32-avr32.c
index e45134c..f882331 100644
--- a/bfd/elf32-avr32.c
+++ b/bfd/elf32-avr32.c
@@ -59,6 +59,8 @@
 /* The name of the dynamic interpreter. This is put in the .interp section. */
 #define ELF_DYNAMIC_INTERPRETER                 "/lib/ld.so.1"

+#define DEFAULT_STACK_SIZE 0x10000
+
 #define AVR32_GOT_HEADER_SIZE          8
 #define AVR32_FUNCTION_STUB_SIZE       8

@@ -68,6 +70,9 @@

 #define NOP_OPCODE 0xd703

+extern const bfd_target bfd_elf32_avr32fdpic_vec;
+#define IS_FDPIC(bfd) ((bfd)->xvec == &bfd_elf32_avr32fdpic_vec)
+

 /* Mapping between BFD relocations and ELF relocations */

@@ -327,6 +332,10 @@ struct elf_avr32_link_hash_table
   asection *sgot;
   asection *srelgot;
   asection *sstub;
+  asection *rofixup;
+
+  unsigned int rofixup_count;
+  unsigned int rofixup_added;

   /* We use a variation of Pigeonhole Sort to sort the GOT.  After the
      initial refcounts have been determined, we initialize
@@ -547,6 +556,39 @@ avr32_elf_create_dynamic_sections (bfd *dynobj, struct bfd_link_info *info)
   return TRUE;
 }

+static bfd_boolean
+avr32_rofixup_create (bfd *abfd, struct bfd_link_info *info)
+{
+  struct elf_avr32_link_hash_table *htab;
+  flagword flags;
+
+  if (!IS_FDPIC(abfd)) {
+    return TRUE;
+  }
+
+  htab = avr32_elf_hash_table(info);
+  if (htab->rofixup) {
+    /* Already created. */
+    return TRUE;
+  }
+
+  flags = (SEC_ALLOC | SEC_LOAD | SEC_HAS_CONTENTS | SEC_IN_MEMORY
+           | SEC_LINKER_CREATED);
+
+  /* We need a .rofixup-section in FD-PIC ELF files. */
+  htab->rofixup = bfd_make_section_with_flags (abfd, ".rofixup",
+                                               (flags | SEC_READONLY));
+  if (htab->rofixup == NULL ||
+      ! bfd_set_section_alignment (abfd, htab->rofixup, 2)) {
+    return FALSE;
+  }
+
+  htab->rofixup->size = 0;
+
+  return TRUE;
+}
+
+
 /* (2) Go through all the relocs and count any potential GOT- or
     PLT-references to each symbol */

@@ -570,6 +612,10 @@ avr32_check_relocs (bfd *abfd, struct bfd_link_info *info, asection *sec,
   if (info->relocatable)
     return TRUE;

+  if (!avr32_rofixup_create(abfd, info)) {
+    return FALSE;
+  }
+
   dynobj = elf_hash_table(info)->dynobj;
   symtab_hdr = &elf_tdata(abfd)->symtab_hdr;
```

```
154     sym_hashes = elf_sym_hashes(abfd);
155  @@ -577,6 +623,11 @@ avr32_check_relocs (bfd *abfd, struct bfd_link_info *info, asection *sec,
156     local_got_ents = elf_local_got_ents(abfd);
157     sgot = htab->sgot;
158
159  +  if (IS_FDPIC(abfd) && dynobj == NULL) {
160  +    elf_hash_table(info)->dynobj = dynobj = abfd;
161  +  }
162  +
163  +
164     rel_end = relocs + sec->reloc_count;
165     for (rel = relocs; rel < rel_end; rel++)
166       {
167  @@ -727,6 +778,21 @@ avr32_check_relocs (bfd *abfd, struct bfd_link_info *info, asection *sec,
168                 }
169               }
170
171  +          if (IS_FDPIC(abfd) && !info->shared && (sec->flags & SEC_ALLOC))
172  +            {
173  +              htab->rofixup_count++;
174  +              if (h != NULL)
175  +                {
176  +                  pr_debug("Non-GOT reference to symbol %s\n",
177  +                           h->root.root.root.string);
178  +                }
179  +              else
180  +                {
181  +                  pr_debug("Non-GOT reference to local symbol %lu\n",
182  +                           r_symndx);
183  +                }
184  +            }
185  +
186             break;
187
188             /* TODO: GNU_VTINHERIT and GNU_VTENTRY */
189  @@ -1265,6 +1331,23 @@ avr32_elf_size_dynamic_sections (bfd *output_bfd,
190       }
191   #undef add_dynamic_entry
192
193  +  if (IS_FDPIC(output_bfd)) {
194  +    /* Time to find the size of the .rofixup-section. */
195  +
196  +    /* Terminator element. */
197  +    htab->rofixup->size = 4;
198  +
199  +    /* We need one entry for each R_AVR32_32 reloc. */
200  +    htab->rofixup->size += 4 * htab->rofixup_count;
201  +
202  +    /* We also need one entry for each got entry. */
203  +    htab->rofixup->size += htab->sgot->size;
204  +
205  +    htab->rofixup->contents = (bfd_byte *) bfd_zalloc(dynobj, htab->rofixup->size);
206  +    if (htab->rofixup->contents == NULL)
207  +      return FALSE;
208  +  }
209  +
210     return TRUE;
211   }
212
213  @@ -3234,6 +3317,110 @@ avr32_final_link_relocate(reloc_howto_type *howto,
214     return status;
215   }
216
217  +static void
218  +avr32_rofixup_add_entry(bfd *output_bfd, struct bfd_link_info *info,
219  +                        asection *section, bfd_vma section_offset)
220  +{
221  +  struct elf_avr32_link_hash_table *htab;
222  +  bfd_vma offset;
223  +  bfd_vma rofixup_entry_offset;
224  +
225  +  htab = avr32_elf_hash_table(info);
226  +
227  +  BFD_ASSERT(htab->rofixup);
228  +  BFD_ASSERT(htab->rofixup->contents);
229  +
230  +  /* Calculate the offset in the output VMA. */
231  +  offset = section_offset + section->output_section->vma + section->output_offset;
232  +
233  +  /* Add that offset to the .rofixup-section. */
234  +  rofixup_entry_offset = htab->rofixup_added * 4;
235  +  BFD_ASSERT(rofixup_entry_offset < htab->rofixup->size);
236  +  bfd_put_32(output_bfd, offset, htab->rofixup->contents + rofixup_entry_offset);
237  +
```

```
+  pr_debug("Added rofixup entry %u for vma %08lx.\n", htab->rofixup_added, offset);
+
+  htab->rofixup_added++;
+}
+
+static void
+avr32_rofixup_add_relocation(bfd *output_bfd, struct bfd_link_info *info,
+                             asection *input_section,
+                             Elf_Internal_Rela *reloc)
+{
+  struct elf_avr32_link_hash_table *htab;
+  bfd_vma offset;
+
+  htab = avr32_elf_hash_table(info);
+
+  if (!IS_FDPIC(output_bfd))
+    return;
+  if (!(input_section->flags & SEC_ALLOC))
+    return;
+
+  /* Find the offset of the symbol in the output file. */
+  offset = _bfd_elf_section_offset(output_bfd, info,
+                                   input_section,
+                                   reloc->r_offset);
+
+  if (offset == (bfd_vma)-1)
+    return;
+  if (offset == (bfd_vma)-2)
+    return;
+
+  if (input_section->flags & SEC_CODE)
+    {
+      /* This should only occur for three symbols: _GLOBAL_OFFSET_TABLE_,
+       * __ROFIXUP_LIST__ and __ROFIXUP_END__. */
+      pr_debug("Skipping relocation for text segment (vma %08lx).\n", offset);
+      return;
+    }
+
+  avr32_rofixup_add_entry(output_bfd, info, input_section, offset);
+}
+
+static void
+avr32_rofixup_add_got(bfd *output_bfd, struct bfd_link_info *info)
+{
+  struct elf_avr32_link_hash_table *htab;
+  bfd_vma offset;
+
+  htab = avr32_elf_hash_table(info);
+
+  if (!IS_FDPIC(output_bfd))
+    return;
+  if (!htab->sgot)
+    return;
+  if (!(htab->sgot->flags & SEC_ALLOC))
+    return;
+
+
+  for (offset = 0; offset < htab->sgot->size; offset += 4) {
+    avr32_rofixup_add_entry(output_bfd, info, htab->sgot, offset);
+  }
+}
+
+
+static void
+avr32_rofixup_terminate(bfd *output_bfd, struct bfd_link_info *info)
+{
+  struct elf_avr32_link_hash_table *htab;
+  bfd_vma rofixup_entry_offset;
+
+  htab = avr32_elf_hash_table(info);
+
+  BFD_ASSERT(htab->rofixup);
+  BFD_ASSERT(htab->rofixup->contents);
+
+  rofixup_entry_offset = htab->rofixup_added * 4;
+  BFD_ASSERT(rofixup_entry_offset < htab->rofixup->size);
+  bfd_put_32(output_bfd, 0xffffffff, htab->rofixup->contents + rofixup_entry_offset);
+
+  pr_debug("Added rofixup terminator.\n");
+
+  htab->rofixup_added++;
+}
+
 /* (6) Apply relocations to the normal (non-dynamic) sections */
```

```
322
323  static bfd_boolean
324 @@ -3435,6 +3622,9 @@ avr32_elf_relocate_section(bfd *output_bfd, struct bfd_link_info *info,
325           break;
326
327         case R_AVR32_32:
328 +         /* First: FDPIC handling... */
329 +         avr32_rofixup_add_relocation(output_bfd, info, input_section, rel);
330 +
331           /* We need to emit a run-time relocation in the following cases:
332                - we're creating a shared library
333                - the symbol is not defined in any regular objects
334 @@ -3700,6 +3890,8 @@ avr32_elf_finish_dynamic_sections(bfd *output_bfd, struct bfd_link_info *info)
335    if (sgot)
336      elf_section_data(sgot->output_section)->this_hdr.sh_entsize = 4;
337
338 +  avr32_rofixup_add_got(output_bfd, info);
339 +  avr32_rofixup_terminate(output_bfd, info);
340    return TRUE;
341  }
342
343 @@ -3862,6 +4054,136 @@ avr32_elf_grok_psinfo(bfd *abfd, Elf_Internal_Note *note)
344  }
345
346
347 +static bfd_boolean
348 +avr32_fdpic_always_size_sections (bfd *output_bfd,
349 +                                  struct bfd_link_info *info)
350 +{
351 +  if (!info->relocatable)
352 +    {
353 +      struct elf_link_hash_entry *h;
354 +
355 +      /* Force a PT_GNU_STACK segment to be created.  */
356 +      if (! elf_tdata (output_bfd)->stack_flags)
357 +       elf_tdata (output_bfd)->stack_flags = PF_R | PF_W | PF_X;
358 +
359 +      /* Define __stacksize if it's not defined yet.  */
360 +      h = elf_link_hash_lookup (elf_hash_table (info), "__stacksize",
361 +                                FALSE, FALSE, FALSE);
362 +      if (! h || h->root.type != bfd_link_hash_defined
363 +          || h->type != STT_OBJECT
364 +          || !h->def_regular)
365 +       {
366 +         struct bfd_link_hash_entry *bh = NULL;
367 +
368 +         if (!(_bfd_generic_link_add_one_symbol
369 +               (info, output_bfd, "__stacksize",
370 +                BSF_GLOBAL, bfd_abs_section_ptr, DEFAULT_STACK_SIZE,
371 +                (const char *) NULL, FALSE,
372 +                get_elf_backend_data (output_bfd)->collect, &bh)))
373 +           return FALSE;
374 +
375 +         h = (struct elf_link_hash_entry *) bh;
376 +         h->def_regular = 1;
377 +         h->type = STT_OBJECT;
378 +       }
379 +    }
380 +
381 +  return TRUE;
382 +}
383 +
384 +
385 +static bfd_boolean
386 +avr32_fdpic_modify_program_headers (bfd *output_bfd,
387 +                                    struct bfd_link_info *info)
388 +{
389 +  struct elf_obj_tdata *tdata = elf_tdata (output_bfd);
390 +  struct elf_segment_map *m;
391 +  Elf_Internal_Phdr *p;
392 +
393 +  /* objcopy and strip preserve what's already there using
394 +     elf32_avr32fdpic_copy_private_bfd_data ().  */
395 +  if (! info)
396 +    return TRUE;
397 +
398 +  /* Search for the PT_GNU_STACK program header. */
399 +  for (p = tdata->phdr, m = tdata->segment_map; m != NULL; m = m->next, p++)
400 +    if (m->p_type == PT_GNU_STACK)
401 +      break;
402 +
403 +  if (m)
404 +    {
405 +      struct elf_link_hash_entry *h;
```

```
+
+      /* Obtain the pointer to the __stacksize symbol.  */
+      h = elf_link_hash_lookup (elf_hash_table (info), "__stacksize",
+                                FALSE, FALSE, FALSE);
+      if (h)
+       {
+         while (h->root.type == bfd_link_hash_indirect
+                || h->root.type == bfd_link_hash_warning)
+           h = (struct elf_link_hash_entry *) h->root.u.i.link;
+         BFD_ASSERT (h->root.type == bfd_link_hash_defined);
+       }
+
+      /* Set the header p_memsz from the symbol value.  We
+         intentionally ignore the symbol section.  */
+      if (h && h->root.type == bfd_link_hash_defined)
+       p->p_memsz = h->root.u.def.value;
+      else
+       p->p_memsz = DEFAULT_STACK_SIZE;
+
+      p->p_align = 8;
+    }
+
+  return TRUE;
+}
+
+
+static bfd_boolean
+avr32_fdpic_copy_private_bfd_data (bfd *ibfd, bfd *obfd)
+{
+  unsigned i;
+
+  if (bfd_get_flavour (ibfd) != bfd_target_elf_flavour
+      || bfd_get_flavour (obfd) != bfd_target_elf_flavour)
+    return TRUE;
+
+  if (! avr32_elf_copy_private_bfd_data (ibfd, obfd))
+    return FALSE;
+
+  if (! elf_tdata (ibfd) || ! elf_tdata (ibfd)->phdr
+      || ! elf_tdata (obfd) || ! elf_tdata (obfd)->phdr)
+    return TRUE;
+
+  /* Copy the stack size.  */
+  for (i = 0; i < elf_elfheader (ibfd)->e_phnum; i++)
+    if (elf_tdata (ibfd)->phdr[i].p_type == PT_GNU_STACK)
+      {
+       Elf_Internal_Phdr *iphdr = &elf_tdata (ibfd)->phdr[i];
+
+       for (i = 0; i < elf_elfheader (obfd)->e_phnum; i++)
+         if (elf_tdata (obfd)->phdr[i].p_type == PT_GNU_STACK)
+           {
+             memcpy (&elf_tdata (obfd)->phdr[i], iphdr, sizeof (*iphdr));
+
+             /* Rewrite the phdrs, since we're only called after they
+                were first written.  */
+             if (bfd_seek (obfd, (bfd_signed_vma) get_elf_backend_data (obfd)
+                           ->s->sizeof_ehdr, SEEK_SET) != 0
+                 || get_elf_backend_data (obfd)->s
+                 ->write_out_phdrs (obfd, elf_tdata (obfd)->phdr,
+                                    elf_elfheader (obfd)->e_phnum) != 0)
+               return FALSE;
+             break;
+           }
+
+       break;
+      }
+
+  return TRUE;
+}
+
+
 #define ELF_ARCH                        bfd_arch_avr32
 #define ELF_MACHINE_CODE                EM_AVR32
 #define ELF_MAXPAGESIZE                         0x1000
@@ -3913,3 +4235,24 @@ avr32_elf_grok_psinfo(bfd *abfd, Elf_Internal_Note *note)
 #define elf_backend_got_header_size     AVR32_GOT_HEADER_SIZE

 #include "elf32-target.h"
+
+
+/* FDPIC target */
+#undef TARGET_BIG_SYM
+#define TARGET_BIG_SYM                  bfd_elf32_avr32fdpic_vec
+#undef TARGET_BIG_NAME
```

```
+#define TARGET_BIG_NAME                         "elf32-avr32fdpic"
+#undef elf32_bed
+#define elf32_bed                       elf32_avr32fdpic_bed
+
+#undef elf_backend_always_size_sections
+#define elf_backend_always_size_sections \
+               avr32_fdpic_always_size_sections
+#undef elf_backend_modify_program_headers
+#define elf_backend_modify_program_headers \
+               avr32_fdpic_modify_program_headers
+#undef bfd_elf32_bfd_copy_private_bfd_data
+#define bfd_elf32_bfd_copy_private_bfd_data \
+               avr32_fdpic_copy_private_bfd_data
+
+#include "elf32-target.h"
diff --git a/bfd/targets.c b/bfd/targets.c
index 975b9b4..70189ff 100644
--- a/bfd/targets.c
+++ b/bfd/targets.c
@@ -565,6 +565,7 @@ extern const bfd_target bfd_efi_app_x86_64_vec;
 extern const bfd_target bfd_efi_app_ia64_vec;
 extern const bfd_target bfd_elf32_avr_vec;
 extern const bfd_target bfd_elf32_avr32_vec;
+extern const bfd_target bfd_elf32_avr32fdpic_vec;
 extern const bfd_target bfd_elf32_bfin_vec;
 extern const bfd_target bfd_elf32_bfinfdpic_vec;
 extern const bfd_target bfd_elf32_big_generic_vec;
@@ -886,6 +887,7 @@ static const bfd_target * const _bfd_target_vector[] =
 #endif
        &bfd_elf32_avr_vec,
        &bfd_elf32_avr32_vec,
+       &bfd_elf32_avr32fdpic_vec,
        &bfd_elf32_bfin_vec,
        &bfd_elf32_bfinfdpic_vec,

diff --git a/gas/config/tc-avr32.c b/gas/config/tc-avr32.c
index 2703ac2..4f7f610 100644
--- a/gas/config/tc-avr32.c
+++ b/gas/config/tc-avr32.c
@@ -49,6 +49,7 @@
 static int avr32_pic    = FALSE;
 int linkrelax  = FALSE;
 int avr32_iarcompat    = FALSE;
+static int avr32_fdpic = FALSE;

 /* This array holds the chars that always start a comment. */
 const char comment_chars[]             = "#";
@@ -266,6 +267,7 @@ struct option md_longopts[] =
 #define OPTION_LINKRELAX       (OPTION_NOPIC + 1)
 #define OPTION_NOLINKRELAX     (OPTION_LINKRELAX + 1)
 #define OPTION_DIRECT_DATA_REFS (OPTION_NOLINKRELAX + 1)
+#define OPTION_FDPIC           (OPTION_DIRECT_DATA_REFS + 1)
   {"march",            required_argument, NULL, OPTION_ARCH},
   {"mpart",            required_argument, NULL, OPTION_PART},
   {"iar",              no_argument, NULL, OPTION_IAR},
@@ -275,6 +277,7 @@ struct option md_longopts[] =
   {"no-linkrelax",     no_argument, NULL, OPTION_NOLINKRELAX},
   /* deprecated alias for -mpart=xxx */
   {"mcpu",             required_argument, NULL, OPTION_PART},
+  {"mfdpic",           no_argument, NULL, OPTION_FDPIC},
   {NULL,               no_argument, NULL, 0}
 };

@@ -380,6 +383,9 @@ md_parse_option (int c, char *arg ATTRIBUTE_UNUSED)
     case OPTION_NOLINKRELAX:
       linkrelax = 0;
       break;
+    case OPTION_FDPIC:
+      avr32_fdpic = 1;
+      break;
     default:
       return 0;
     }
@@ -3672,6 +3678,8 @@ md_begin (void)
     flags |= EF_AVR32_LINKRELAX;
   if (avr32_pic)
     flags |= EF_AVR32_PIC;
+  if (avr32_fdpic)
+    flags |= EF_AVR32_FDPIC;

   bfd_set_private_flags(stdoutput, flags);

diff --git a/include/elf/avr32.h b/include/elf/avr32.h
index d73943d..00a5f60 100644
```

```
--- a/include/elf/avr32.h
+++ b/include/elf/avr32.h
@@ -25,6 +25,7 @@
 /* CPU-specific flags for the ELF header e_flags field */
 #define EF_AVR32_LINKRELAX              0x01
 #define EF_AVR32_PIC                    0x02
+#define EF_AVR32_FDPIC                  0x04

 START_RELOC_NUMBERS (elf_avr32_reloc_type)
     RELOC_NUMBER (R_AVR32_NONE,                         0)
diff --git a/ld/Makefile.am b/ld/Makefile.am
index 58c3f2c..3b064a6 100644
--- a/ld/Makefile.am
+++ b/ld/Makefile.am
@@ -165,6 +165,7 @@ ALL_EMULATIONS = \
        eavr32elf_uc3b1256es.o \
        eavr32elf_uc3b1256.o \
        eavr32linux.o \
+       eavr32linuxfdpic.o \
        ecoff_i860.o \
        ecoff_sparc.o \
        eelf32_spu.o \
@@ -757,6 +758,10 @@ eavr32linux.c: $(srcdir)/emulparams/avr32linux.sh \
   $(srcdir)/emultempl/elf32.em $(srcdir)/emultempl/avr32elf.em \
   $(srcdir)/scripttempl/elf.sc ${GEN_DEPENDS}
        ${GENSCRIPTS} avr32linux "$(tdir_avr32)"
+eavr32linuxfdpic.c: $(srcdir)/emulparams/avr32linuxfdpic.sh \
+  $(srcdir)/emultempl/elf32.em $(srcdir)/emultempl/avr32elf.em \
+  $(srcdir)/scripttempl/elf.sc ${GEN_DEPENDS}
+       ${GENSCRIPTS} avr32linuxfdpic "$(tdir_avr32)"
 ecoff_i860.c: $(srcdir)/emulparams/coff_i860.sh \
   $(srcdir)/emultempl/generic.em $(srcdir)/scripttempl/i860coff.sc ${GEN_DEPENDS}
        ${GENSCRIPTS} coff_i860 "$(tdir_coff_i860)"
diff --git a/ld/Makefile.in b/ld/Makefile.in
index 1193a74..5cacda9 100644
--- a/ld/Makefile.in
+++ b/ld/Makefile.in
@@ -412,6 +412,7 @@ ALL_EMULATIONS = \
        eavr32elf_uc3b1256es.o \
        eavr32elf_uc3b1256.o \
        eavr32linux.o \
+       eavr32linuxfdpic.o \
        ecoff_i860.o \
        ecoff_sparc.o \
        eelf32_spu.o \
@@ -1583,6 +1584,10 @@ eavr32linux.c: $(srcdir)/emulparams/avr32linux.sh \
   $(srcdir)/emultempl/elf32.em $(srcdir)/emultempl/avr32elf.em \
   $(srcdir)/scripttempl/elf.sc ${GEN_DEPENDS}
        ${GENSCRIPTS} avr32linux "$(tdir_avr32)"
+eavr32linuxfdpic.c: $(srcdir)/emulparams/avr32linuxfdpic.sh \
+  $(srcdir)/emultempl/elf32.em $(srcdir)/emultempl/avr32elf.em \
+  $(srcdir)/scripttempl/elf.sc ${GEN_DEPENDS}
+       ${GENSCRIPTS} avr32linuxfdpic "$(tdir_avr32)"
 ecoff_i860.c: $(srcdir)/emulparams/coff_i860.sh \
   $(srcdir)/emultempl/generic.em $(srcdir)/scripttempl/i860coff.sc ${GEN_DEPENDS}
        ${GENSCRIPTS} coff_i860 "$(tdir_coff_i860)"
diff --git a/ld/configure.tgt b/ld/configure.tgt
index c0c74f3..2012162 100644
--- a/ld/configure.tgt
+++ b/ld/configure.tgt
@@ -111,7 +111,9 @@ avr-*-*)            targ_emul=avr2
                        ;;
 avr32-*-none)       targ_emul=avr32elf_ap7000
             targ_extra_emuls="avr32elf_ap7001 avr32elf_ap7002 avr32elf_ap7200 avr32elf_uc3a0128
                 avr32elf_uc3a0256 avr32elf_uc3a0512 avr32elf_uc3a0512es avr32elf_uc3a1128 avr32elf_
                 uc3a1256 avr32elf_uc3a1512es avr32elf_uc3a1512 avr32elf_uc3a364 avr32elf_uc3a364s
                 avr32elf_uc3a3128 avr32elf_uc3a3128s avr32elf_uc3a3256 avr32elf_uc3a3256s avr32elf_
                 uc3b064 avr32elf_uc3b0128 avr32elf_uc3b0256es avr32elf_uc3b0256 avr32elf_uc3b164
                 avr32elf_uc3b1128 avr32elf_uc3b1256es avr32elf_uc3b1256" ;;
-avr32-*-linux*)     targ_emul=avr32linux ;;
+avr32-*-linux* | avr32-*-uclinux*)     targ_emul=avr32linux
+                       targ_extra_emuls="avr32linuxfdpic"
+                       ;;
 bfin-*-elf)           targ_emul=elf32bfin;
                       targ_extra_emuls="elf32bfinfd"
                       targ_extra_libpath=$targ_extra_emuls
diff --git a/ld/emulparams/avr32linux.sh b/ld/emulparams/avr32linux.sh
index f281f9d..fd36e7d 100644
--- a/ld/emulparams/avr32linux.sh
+++ b/ld/emulparams/avr32linux.sh
@@ -4,6 +4,7 @@ TEMPLATE_NAME=elf32
 EXTRA_EM_FILE=avr32elf
 OUTPUT_FORMAT="elf32-avr32"
 GENERATE_SHLIB_SCRIPT=yes
```

```
+GENERATE_PIE_SCRIPT=yes
 MAXPAGESIZE=0x1000
 TEXT_START_ADDR=0x00001000
 NOP=0xd703d703
diff --git a/ld/emulparams/avr32linuxfdpic.sh b/ld/emulparams/avr32linuxfdpic.sh
new file mode 100644
index 0000000..e5d7f96
--- /dev/null
+++ b/ld/emulparams/avr32linuxfdpic.sh
@@ -0,0 +1,10 @@
+. ${srcdir}/emulparams/avr32linux.sh
+OUTPUT_FORMAT="elf32-avr32fdpic"
+
+OTHER_READONLY_SECTIONS="
+  .rofixup        : {
+    ${RELOCATING+__ROFIXUP_LIST__ = .;}
+    *(.rofixup)
+    ${RELOCATING+__ROFIXUP_END__ = .;}
+  }
+"
--
1.5.4.3
```

F.4 uClibc changes

```
From 4681d0587bcd0d6d300916b80e4a30abe54aa962 Mon Sep 17 00:00:00 2001
Date: Tue, 26 May 2009 15:24:13 +0200
Subject: [PATCH] uClibc: Some support for FDPIC ELF for AVR32

This patch enables uClibc to be linked statically into a FDPIC ELF
binary on AVR32. It doesn't update the parts necessary for dynamic
linking.

There are also a few simple changes to memcmp, memcpy and memmove, which
makes them work on the UC3 (which cannot access unaligned memory.)
---
 Rules.mak                                |    7 +++
 extra/Configs/Config.avr32               |    3 +
 libc/string/avr32/memcmp.S               |   11 ++++
 libc/string/avr32/memcpy.S               |   15 ++++++
 libc/string/avr32/memmove.S              |   16 ++++++
 libc/sysdeps/linux/avr32/Makefile.arch   |    2 +-
 libc/sysdeps/linux/avr32/crt1.S          |   40 ++++++++++++++++-
 libc/sysdeps/linux/avr32/crti.S          |    4 ++
 libc/sysdeps/linux/avr32/crtreloc.c      |   85 ++++++++++++++++++++++++++++++++++++
 libc/sysdeps/linux/avr32/syscall.S       |    6 ++
 libc/sysdeps/linux/avr32/vfork.S         |    4 ++
 11 files changed, 191 insertions(+), 2 deletions(-)
 create mode 100644 libc/sysdeps/linux/avr32/crtreloc.c

diff --git a/Rules.mak b/Rules.mak
index d3cda90..d3a7e15 100644
--- a/Rules.mak
+++ b/Rules.mak
@@ -399,8 +399,15 @@ endif

 ifeq ($(strip $(TARGET_ARCH)),avr32)
        CPU_CFLAGS-$(CONFIG_AVR32_AP7)  += -march=ap
+       CPU_CFLAGS-$(CONFIG_AVR32_UC3)  += -march=ucr1
        CPU_CFLAGS-$(CONFIG_LINKRELAX)  += -mrelax
        CPU_LDFLAGS-$(CONFIG_LINKRELAX) += --relax
+
+ifeq ($(UCLIBC_FORMAT_FDPIC_ELF),y)
+       CPU_CFLAGS-y += -mfdpic -mno-init-got
+       CPU_LDFLAGS-y += -mfdpic
+endif
+
 endif

 ifeq ($(TARGET_ARCH),i960)
diff --git a/extra/Configs/Config.avr32 b/extra/Configs/Config.avr32
index 8d70e6e..4e109ae 100644
--- a/extra/Configs/Config.avr32
+++ b/extra/Configs/Config.avr32
@@ -24,6 +24,9 @@ config CONFIG_AVR32_AP7
        bool "AVR32 AP7"
        select ARCH_HAS_MMU

+config CONFIG_AVR32_UC3
```

```
+       bool "AVR32 UC3"
+
 endchoice

 config LINKRELAX
diff --git a/libc/string/avr32/memcmp.S b/libc/string/avr32/memcmp.S
index ae6cc91..59b799e 100644
--- a/libc/string/avr32/memcmp.S
+++ b/libc/string/avr32/memcmp.S
@@ -20,6 +20,17 @@ memcmp:
        sub     len, 4
        brlt    .Lless_than_4

+#ifdef __CONFIG_AVR32_UC3__
+       /* This CPU cannot do unaligned accesses. */
+       mov     r9, s1
+       andl    r9, 3, COH
+       brne    .Lless_than_4 /* s1 unaligned */
+
+       mov     r9, s2
+       andl    r9, 3, COH
+       brne    .Lless_than_4 /* s2 unaligned */
+#endif /* __CONFIG_AVR32_UC3__ */
+
 1:     ld.w    r8, s1++
        ld.w    r9, s2++
        cp.w    r8, r9
diff --git a/libc/string/avr32/memcpy.S b/libc/string/avr32/memcpy.S
index bf091ab..803fbdc 100644
--- a/libc/string/avr32/memcpy.S
+++ b/libc/string/avr32/memcpy.S
@@ -6,6 +6,8 @@
  * archive for more details.
  */

+#include <features.h>
+
 /* Don't use r12 as dst since we must return it unmodified */
 #define dst r9
 #define src r11
@@ -91,6 +93,18 @@ memcpy:

 .Lunaligned_dst:
        /* src is aligned, but dst is not. Expect bad performance */
+#ifdef __CONFIG_AVR32_UC3__
+       /* This CPU cannot do unaligned accesses. */
+1:
+       sub     len, 1
+       brlt    2f
+       ld.ub   r0, src++
+       st.b    dst++, r0
+       rjmp    1b
+2:
+
+#else /* __CONFIG_AVR32_UC3__ */
+
        sub     len, 4
        brlt    2f
 1:     ld.w    r0, src++
@@ -104,6 +118,7 @@ memcpy:
        ld.ub   r0, src++
        st.b    dst++, r0
        .endr
+#endif /* __CONFIG_AVR32_UC3__ */

        popm    r0-r7, pc
        .size   memcpy, . - memcpy
diff --git a/libc/string/avr32/memmove.S b/libc/string/avr32/memmove.S
index 535f4a2..1b44d84 100644
--- a/libc/string/avr32/memmove.S
+++ b/libc/string/avr32/memmove.S
@@ -6,6 +6,8 @@
  * archive for more details.
  */

+#include <features.h>
+
 #define dst r12
 #define src r11
 #define len r10
@@ -96,6 +98,19 @@ memmove:

 .Lunaligned_dst:
        /* src is aligned, but dst is not. Expect bad performance */
```

```
+
+#ifdef __CONFIG_AVR32_UC3__
+       /* This CPU cannot do unaligned accesses. */
+1:
+       sub     len, 1
+       brlt    2f
+       ld.ub   r0, --src
+       st.b    --dst, r0
+       rjmp    1b
+2:
+
+#else /* __CONFIG_AVR32_UC3__ */
+
        sub     len, 4
        brlt    2f
 1:     ld.w    r0, --src
@@ -109,6 +124,7 @@ memmove:
        ld.ub   r0, --src
        st.b    --dst, r0
        .endr
+#endif /* __CONFIG_AVR32_UC3__ */

        popm    r0-r7, pc
        .size   memmove, . - memmove
diff --git a/libc/sysdeps/linux/avr32/Makefile.arch b/libc/sysdeps/linux/avr32/Makefile.arch
index 44fc01e..0d905f8 100644
--- a/libc/sysdeps/linux/avr32/Makefile.arch
+++ b/libc/sysdeps/linux/avr32/Makefile.arch
@@ -5,7 +5,7 @@
 # Licensed under the LGPL v2.1, see the file COPYING.LIB in this tarball.
 #

-CSRC   := brk.c clone.c mmap.c sigaction.c
+CSRC   := brk.c clone.c mmap.c sigaction.c crtreloc.c

 SSRC   := __longjmp.S setjmp.S bsd-setjmp.S bsd-_setjmp.S      \
                sigrestorer.S syscall.S vfork.S
diff --git a/libc/sysdeps/linux/avr32/crt1.S b/libc/sysdeps/linux/avr32/crt1.S
index ca1fa7a..b4ca2e8 100644
--- a/libc/sysdeps/linux/avr32/crt1.S
+++ b/libc/sysdeps/linux/avr32/crt1.S
@@ -48,7 +48,45 @@ _start:
        st.w    --sp, r10               /* stack_end */
        st.w    --sp, r12               /* rtld_fini */

-#ifdef __PIC__
+#ifdef __AVR32_FDPIC__
+       /* We need to save r10 & r11 until after relocation. */
+       mov     r3, r10
+       mov     r4, r11
+
+       /* FDPIC handing... */
+
+       mov     r12, r0
+
+       /* Find the rofixup address. */
+       lddpc   r11, .L_original_rofixup
+
+       /* Find the got. */
+       lddpc   r10, .L_original_got
+
+       /* Do relocations. */
+       rcall   __self_reloc
+
+       /* Relocated GOT pointer returned in r12. */
+       mov     r6, r12
+
+       /* Restore r10 & r11. */
+       mov     r10, r3
+       mov     r11, r4
+
+       lda.w   r9, _init
+       lda.w   r8, _fini
+       lda.w   r12, main
+
+       /* Ok, now run uClibc's main() -- should not return */
+       call    __uClibc_main
+
+       .align  2
+.L_original_rofixup:
+       .long   __ROFIXUP_LIST__
+.L_original_got:
+       .long   _GLOBAL_OFFSET_TABLE_
+
```

```
+#elif defined(__PIC__)
        lddpc   r6, .L_GOT
 .L_RGOT:
        rsub    r6, pc
diff --git a/libc/sysdeps/linux/avr32/crti.S b/libc/sysdeps/linux/avr32/crti.S
index 660f47c..b39c4bf 100644
--- a/libc/sysdeps/linux/avr32/crti.S
+++ b/libc/sysdeps/linux/avr32/crti.S
@@ -5,12 +5,14 @@
        .type   _init, @function
 _init:
        stm     --sp, r6, lr
+#ifndef __AVR32_FDPIC__
        lddpc   r6, 2f
 1:     rsub    r6, pc
        rjmp    3f
        .align  2
 2:     .long   1b - _GLOBAL_OFFSET_TABLE_
 3:
+#endif /* __AVR32_FDPIC__ */

        .section .fini
        .align  2
@@ -18,9 +20,11 @@ _init:
        .type   _fini, @function
 _fini:
        stm     --sp, r6, lr
+#ifndef __AVR32_FDPIC__
        lddpc   r6, 2f
 1:     rsub    r6, pc
        rjmp    3f
        .align  2
 2:     .long   1b - _GLOBAL_OFFSET_TABLE_
 3:
+#endif /* __AVR32_FDPIC__ */
diff --git a/libc/sysdeps/linux/avr32/crtreloc.c b/libc/sysdeps/linux/avr32/crtreloc.c
new file mode 100644
index 0000000..633e53a
--- /dev/null
+++ b/libc/sysdeps/linux/avr32/crtreloc.c
@@ -0,0 +1,85 @@
+#include <sys/types.h>
+#include <link.h>
+
+
+/* This data structure represents a PT_LOAD segment.  */
+struct elf32_fdpic_loadseg
+{
+  /* Core address to which the segment is mapped.  */
+  unsigned long addr;
+  /* VMA recorded in the program header.  */
+  unsigned long p_vaddr;
+  /* Size of this segment in memory.  */
+  unsigned long p_memsz;
+};
+
+struct elf32_fdpic_loadmap {
+  /* Protocol version number, must be zero.  */
+  unsigned short version;
+  /* Number of segments in this map.  */
+  unsigned short nsegs;
+  /* The actual memory map.  */
+  struct elf32_fdpic_loadseg segs[/*nsegs*/];
+};
+
+static __always_inline void *
+__reloc_pointer (void *p,
+                 const struct elf32_fdpic_loadmap *map)
+{
+  int c;
+
+#if 0
+  if (map->version != 0)
+    /* Crash.  */
+    ((void(*)())0)();
+#endif
+
+  /* No special provision is made for NULL.  We don't want NULL
+     addresses to go through relocation, so they shouldn't be in
+     .rofixup sections, and, if they're present in dynamic
+     relocations, they shall be mapped to the NULL address without
+     undergoing relocations.  */
+  for (c = 0;
+       /* Take advantage of the fact that the loadmap is ordered by
```

```
+         virtual addresses.  In general there will only be 2 entries,
+         so it's not profitable to do a binary search.  */
+       c < map->nsegs && p >= (void*)map->segs[c].p_vaddr;
+       c++)
+    {
+      /* This should be computed as part of the pointer comparison
+        above, but we want to use the carry in the comparison, so we
+        can't convert it to an integer type beforehand.  */
+      unsigned long offset = p - (void*)map->segs[c].p_vaddr;
+      /* We only check for one-past-the-end for the last segment,
+        assumed to be the data segment, because other cases are
+        ambiguous in the absence of padding between segments, and
+        rofixup already serves as padding between text and data.
+        Unfortunately, unless we special-case the last segment, we
+        fail to relocate the _end symbol.  */
+      if (offset < map->segs[c].p_memsz
+         || (offset == map->segs[c].p_memsz && c + 1 == map->nsegs))
+       return (char*)map->segs[c].addr + offset;
+    }
+
+  /* We might want to crash instead.  */
+
+  return (void*)-1;
+}
+
+void* __self_reloc (const struct elf32_fdpic_loadmap *map,
+                    void ***reloc_list, void *got)
+{
+  void ***i;
+  void **e;
+
+  reloc_list = __reloc_pointer(reloc_list, map);
+
+  for (i = reloc_list; (unsigned long)*i != 0xffffffff; i++) {
+    e = __reloc_pointer(*i, map);
+    if (*e != 0) {
+      *e = __reloc_pointer(*e, map);
+    }
+  }
+
+  return __reloc_pointer(got, map);
+}
diff --git a/libc/sysdeps/linux/avr32/syscall.S b/libc/sysdeps/linux/avr32/syscall.S
index 55c1b1f..abea2b5 100644
--- a/libc/sysdeps/linux/avr32/syscall.S
+++ b/libc/sysdeps/linux/avr32/syscall.S
@@ -25,9 +25,13 @@ syscall:
        brlo    .Ldone

 #ifdef __PIC__
+
+#ifndef __AVR32_FDPIC__
        lddpc   r6, .Lgot
 .Lgotcalc:
        rsub    r6, pc
+#endif /* ____AVR32_FDPIC__ */
+
 # ifdef __UCLIBC_HAS_THREADS__
        rsub    r3, r12, 0
        mcall   r6[__errno_location@got]
@@ -55,8 +59,10 @@ syscall:

        .align  2
 #ifdef __PIC__
+#ifndef __AVR32_FDPIC__
 .Lgot:
        .long   .Lgotcalc - _GLOBAL_OFFSET_TABLE_
+#endif /* __AVR32_FDPIC__ */
 #else
 # ifdef __UCLIBC_HAS_THREADS__
 .Lerrno_location:
diff --git a/libc/sysdeps/linux/avr32/vfork.S b/libc/sysdeps/linux/avr32/vfork.S
index 03ca99f..830cba4 100644
--- a/libc/sysdeps/linux/avr32/vfork.S
+++ b/libc/sysdeps/linux/avr32/vfork.S
@@ -32,10 +32,12 @@ __vfork:
        /* vfork failed, so we may use the stack freely */
        pushm   r4-r7,lr
 #ifdef __PIC__
+#ifndef __AVR32_FDPIC__
        lddpc   r6, .L_GOT
        rsub    r4, r12, 0
 .L_RGOT:
        rsub    r6, pc
```

```
+#endif /* __AVR32_FDPIC__ */
        mcall   r6[__errno_location@got]
 #else
        rsub    r4, r12, 0
@@ -46,8 +48,10 @@ __vfork:

        .align  2
 #ifdef __PIC__
+#ifndef __AVR32_FDPIC__
 .L_GOT:
        .long   .L_RGOT - _GLOBAL_OFFSET_TABLE_
+#endif /* __AVR32_FDPIC__ */
 #else
 .L__errno_location:
        .long   __errno_location
--
1.5.4.3
```

F.5 Unsubmitted GCC change

```
From ec741a7b83a01e8f316fbf649cf98da0601f86da Mon Sep 17 00:00:00 2001
From: =?utf-8?q?Gunnar=20Rang=C3=B8y?= <rangoy@mnops.(none)>
Date: Tue, 2 Jun 2009 10:33:17 +0200
Subject: [PATCH] Set -mno-init-gout if -mfdpic is specified.

This patch changes gcc so that specifying -mfdpic flag automatically adds the
-mno-init-got flag.
---
 gcc/config/avr32/linux-elf.h |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

diff --git a/gcc/config/avr32/linux-elf.h b/gcc/config/avr32/linux-elf.h
index cb206a1..5dd7dbf 100644
--- a/gcc/config/avr32/linux-elf.h
+++ b/gcc/config/avr32/linux-elf.h
@@ -70,6 +70,7 @@
 #define DRIVER_SELF_SPECS "\
   %{mfdpic:%{!fpic:%{!fpie:%{!fPIC:%{!fPIE:\
             %{!fno-pic:%{!fno-pie:%{!fno-PIC:%{!fno-PIE:-fpie}}}}}}}} \
+ %{mfdpic:-mno-init-got} \
 "

 #undef ASM_SPEC
--
1.5.4.3
```

Appendix G

Patch for elf2flt

This appendix lists the patch for the modifications done to the elf2flt utility while experimenting with the flat format. The patch is based on a CVS-snapshot (6. March 2009). These changes were not submitted to the maintainers, and probably never will be, since no useful results were achieved.

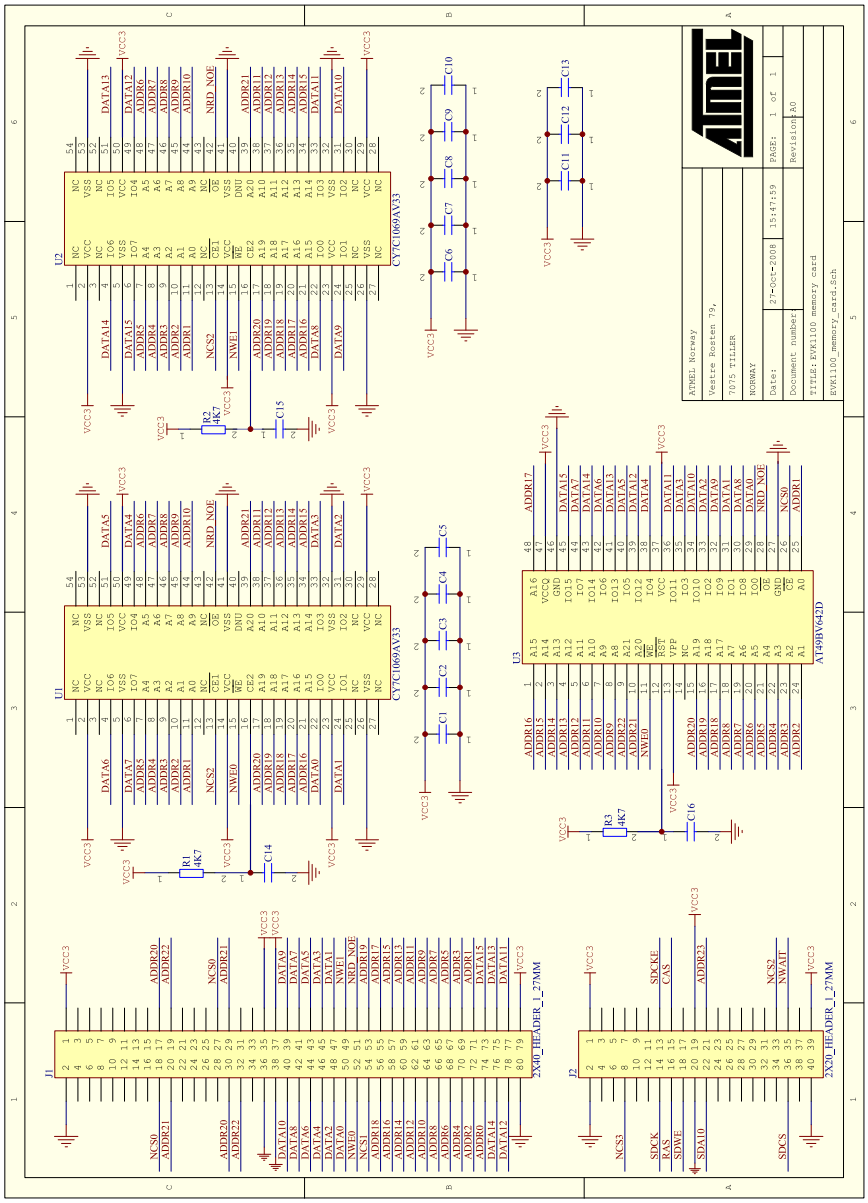
```
diff --git a/config.sub b/config.sub
index 4279c84..ed9cbb6 100755
--- a/config.sub
+++ b/config.sub
@@ -230,6 +230,7 @@ case $basic_machine in
         | alpha | alphaev[4-8] | alphaev56 | alphaev6[78] | alphapca5[67] \
         | alpha64 | alpha64ev[4-8] | alpha64ev56 | alpha64ev6[78] | alpha64pca5[67] \
         | am33_2.0 \
+        | avr32 \
         | arc | arm | arm[bl]e | arme[lb] | armv[2345] | armv[345][lb] | avr \
         | bfin \
         | c4x | clipper \
@@ -425,6 +426,10 @@ case $basic_machine in
                 basic_machine=m68k-apple
                 os=-aux
                 ;;
+        avr32)
+                basic_machine=avr32
+                os=-linux
+                ;;
         balance)
                 basic_machine=ns32k-sequent
                 os=-dynix
diff --git a/elf2flt.c b/elf2flt.c
index 546305f..9d97c39 100644
--- a/elf2flt.c
+++ b/elf2flt.c
@@ -64,6 +64,8 @@
 #include <elf/microblaze.h>    /* TARGET_* ELF support for the BFD library */
 #elif defined(TARGET_bfin)
 #include "elf/bfin.h"
+#elif defined(TARGET_avr32)
+#include "elf/avr32.h"
 #else
 #include <elf.h>      /* TARGET_* ELF support for the BFD library            */
 #endif
@@ -113,6 +115,9 @@
 #define ARCH   "nios"
 #elif defined(TARGET_nios2)
 #define ARCH   "nios2"
+#elif defined(TARGET_avr32)
+#define ARCH   "avr32"
+
 #else
 #error "Don't know how to support your CPU architecture??"
 #endif
@@ -140,7 +145,7 @@
 #endif


-int verbose = 0;       /* extra output when running */
+int verbose = 1;       /* extra output when running */
 int pic_with_got = 0; /* do elf/got processing with PIC code */
```

```
 int load_to_ram = 0;  /* instruct loader to allocate everything into RAM */
 int ktrace = 0;       /* instruct loader output kernel trace on load */
@@ -404,6 +409,7 @@ output_relocs (
    int                  bad_relocs = 0;
    asymbol              **symb;
    long                 nsymb;
+   int                  i;

 #if 0
    printf("%s(%d): output_relocs(abs_bfd=%d,synbols=0x%x,number_of_symbols=%d"
@@ -427,6 +433,7 @@ dump_symbols(symbols, number_of_symbols);
     * Also note that both the relocatable and absolute versions have this
     * terminator even though the relocatable one doesn't have the GOT!
     */
+   printf("pwg: %i  %i\n", pic_with_got, use_resolved);
    if (pic_with_got && !use_resolved) {
      unsigned long *lp = (unsigned long *)data;
      /* Should call ntohl(*lp) here but is isn't going to matter */
@@ -444,6 +451,21 @@ dump_symbols(symbols, number_of_symbols);
 #endif
    }

+#ifdef TARGET_avr32_disable_
+   flat_relocs = realloc(flat_relocs,
+                         (flat_reloc_count + got_size) * sizeof(uint32_t));
+
+   for (i = 0; i < got_size / sizeof(uint32_t); i++) {
+       unsigned long offset = data_vma + i * sizeof(uint32_t);
+       uint32_t value = ntohl(((uint32_t *)data)[i]);
+
+       fprintf(stderr, "Add GOT reloc at 0x%08x (value: 0x%08x)\n", offset, value);
+       flat_relocs[flat_reloc_count] = pflags | offset;
+       flat_reloc_count++;
+   }
+#endif /* TARGET_avr32 */
+
+   fprintf(stderr, "casd: %lu\n", (unsigned long)flat_reloc_count);
    for (a = abs_bfd->sections; (a != (asection *) NULL); a = a->next) {
         section_vma = bfd_section_vma(abs_bfd, a);

@@ -614,7 +636,8 @@ dump_symbols(symbols, number_of_symbols);
                                    the program text.  How this is handled may
                                    still depend on the particular relocation
                                    though.  */
-                                switch (q->howto->type) {
+                           printf("Switching on : %d", q->howto->type);
+                           switch (q->howto->type) {
                                        int r2_type;
 #ifdef TARGET_v850
                                case R_V850_HI16_S:
@@ -708,6 +731,26 @@ dump_symbols(symbols, number_of_symbols);
                                        break;
                                default:
                                        goto bad_resolved_reloc;
+
+#elif defined(TARGET_avr32)
+                               case R_AVR32_32:
+                                   printf("reloacting switch(AVR32_32), typenr: %d\n", q->howto->type);
+                                   relocation_needed = 1;
+                                   break;
+                               case R_AVR32_DIFF32:
+                                   printf("reloacting switch(DIFF32), typenr: %d\n", q->howto->type);
+                                   relocation_needed = 0;
+                                   break;
+                               case R_AVR32_GOTPC:
+                               case R_AVR32_GOT16S:
+                                   printf("reloacting switch(GOT), typenr: %d\n", q->howto->type);
+                                   relocation_needed = 0;
+                                   break;
+
+                               default:
+                                   printf("reloacting switch(DEFAULT), typenr: %d\n", q->howto->type);
+                                   goto bad_resolved_reloc;
+
 #elif defined(TARGET_m68k)
                                case R_68K_32:
                                        goto good_32bit_resolved_reloc;
@@ -818,7 +861,21 @@ dump_symbols(symbols, number_of_symbols);
                                        sym_addr = (sym_addr-q->address)>>(*p)->howto->rightshift;
                                        break;
 #endif
-
+#ifdef TARGET_avr32
+                               case R_AVR32_32:
```

```
+                                       sym_vma = bfd_section_vma(abs_bfd, sym_section);
+                                       sym_addr += sym_vma + q->addend;
+                                       printf("real reloacting switch(AVR32_32), typenr: %d\n", q->
    howto->type);
+                                       relocation_needed = 1;
+                                       break;
+                                case R_AVR32_DIFF32:
+                                       printf("real reloacting switch(AVR32_DIFF32), typenr: %d\n", q->
    howto->type);
+                                       break;
+                                case R_AVR32_GOTPC:
+                                case R_AVR32_GOT16S:
+                                    printf("real reloacting switch(GOT), typenr: %d\n", q->howto->type);
+                                    break;
+#endif
 #ifdef TARGET_v850
                                 case R_V850_32:
                                       relocation_needed = 1;
@@ -1945,6 +2002,18 @@ int main(int argc, char *argv[])
     bfd_size_type sec_size;
     bfd_vma sec_vma;

+    sec_size = bfd_section_size(abs_bfd, s);
+    sec_vma  = bfd_section_vma(abs_bfd, s);
+
+    if(sec_size ==0)
+       continue;
+
+    fprintf(stderr, "name: %-20s %#7lx %#7lx  (%#7lx)  flags: %s%s%s\n", s->name,
+          sec_vma, sec_vma + sec_size, sec_size,
+          (s->flags & SEC_CODE) ? "C" : "",
+          (s->flags & SEC_DATA) ? "D" : "",
+          (s->flags & SEC_ALLOC) ? "A" : "");
+
     if (s->flags & SEC_CODE) {
       vma = &text_vma;
       len = &text_len;
@@ -1957,8 +2026,6 @@ int main(int argc, char *argv[])
     } else
       continue;

-    sec_size = bfd_section_size(abs_bfd, s);
-    sec_vma  = bfd_section_vma(abs_bfd, s);

     if (sec_vma < *vma) {
       if (*len > 0)
@@ -2065,6 +2132,7 @@ int main(int argc, char *argv[])
          | (pic_with_got ? FLAT_FLAG_GOTPIC : 0)
          | (docompress ? (docompress == 2 ? FLAT_FLAG_GZDATA : FLAT_FLAG_GZIP) : 0)
          );
+  printf("load to: %i\n", load_to_ram);
   hdr.build_date = htonl((unsigned long)time(NULL));
   memset(hdr.filler, 0x00, sizeof(hdr.filler));
```

Appendix H

EVK1100 SRAM expansion board



Appendix I

Test source code

I.1 Linux exception tests

I.1.1 Unaligned read

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static void sigbus_handler(int ignored)
{
  fprintf(stderr, "Got SIGBUS exception.\n");
  exit(1);
}

static char buffer[16];

int main()
{
  int *p = (int *)(&buffer[1]); /* Create unaligned pointer. */

  signal(SIGBUS, sigbus_handler);

  fprintf(stderr, "Triggering SIGBUS exception (unaligned read):\n");
  printf("*p is: %d\n", *p);
  fprintf(stderr, "Exception didn't trigger.\n");

  return 0;
}
```

I.1.2 Unaligned write

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static void sigbus_handler(int ignored)
{
  fprintf(stderr, "Got SIGBUS exception.\n");
  exit(1);
}

static char buffer[16];

int main()
```

```
{
  int *p = (int *)(&buffer[1]); /* Create unaligned pointer. */;

  signal(SIGBUS, sigbus_handler);

  fprintf(stderr, "Triggering SIGBUS exception (unaligned write):\n");
  *p = 42;
  fprintf(stderr, "Exception didn't trigger.\n");

  return 0;
}
```

I.1.3 Invalid read

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static void sigbus_handler(int ignored)
{
  fprintf(stderr, "Got SIGBUS exception.\n");
  exit(1);
}

int main()
{
  int *p = (int *)0x100000;

  signal(SIGBUS, sigbus_handler);

  fprintf(stderr, "Triggering SIGBUS exception (invalid read):\n");
  printf("*p is: %d\n", *p);
  fprintf(stderr, "Exception didn't trigger.\n");

  return 0;
}
```

I.1.4 Invalid write

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static void sigbus_handler(int ignored)
{
  fprintf(stderr, "Got SIGBUS exception.\n");
  exit(1);
}

int main()
{
  int *p = (int *)0x100000;

  signal(SIGBUS, sigbus_handler);

  fprintf(stderr, "Triggering SIGBUS exception (invalid write):\n");
  *p = 42;
  fprintf(stderr, "Exception didn't trigger.\n");

  return 0;
}
```

I.1.5 Invalid opcode (aligned)

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static void sigill_handler(int ignored)
{
  fprintf(stderr, "Got SIGILL exception.\n");
  exit(1);
}

int main()
{
  signal(SIGILL, sigill_handler);

  fprintf(stderr, "Triggering SIGILL exception (rsubeq instruction):\n");
  asm(".balignw 4, 0xd703");   /* Align on 4 bytes, pad with NOPs. */
  asm("rsubeq r0, 42"); /* Illegal opcode. */
  fprintf(stderr, "Exception didn't trigger.\n");

  return 0;
}
```

I.1.6 Invalid opcode (unaligned)

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static void sigill_handler(int ignored)
{
  fprintf(stderr, "Got SIGILL exception.\n");
  exit(1);
}

static void sigsegv_handler(int ignored)
{
  fprintf(stderr, "Got SIGSEGV exception.\n");
  exit(1);
}

int main()
{
  signal(SIGILL, sigill_handler);
  signal(SIGSEGV, sigsegv_handler);

  fprintf(stderr, "Triggering SIGILL exception (halfword aligned rsubeq instruction):\n");
  asm(".balignw 4, 0xd703");   /* Align on 4 bytes, pad with NOPs. */
  asm("nop"); /* Make sure that the illegal opcode is aligned at a half-word boundary. */
  asm("rsubeq r0, 42"); /* Illegal opcode. */
  fprintf(stderr, "Exception didn't trigger.\n");

  return 0;
}
```

I.2 Toolchain tests

I.2.1 Simple program

```c
#include <unistd.h>

int main(int argc, char *argv[])
{
  write(1, "Hello!\n", 7);
  return 0;
}
```

I.2.2 More complex program

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  printf("Hello world! %d\n", 42);
  return 0;
}
```

Appendix J

Digital appendices

This appendix lists the digital appendices.

J.1 Linux patches

This is a directory with the patches for Linux

J.2 U-Boot patches

This is a directory with the patches we submitted for U-Boot.

J.3 U-Boot unsubmitted changes

This is a patch with the unsubmitted changes for U-Boot.

J.4 Toolchain patches

This directory contains the patches we submitted for GCC, GNU Binutils and uClibc.

J.5 elf2flt changes

This is a patch with the changes we made to elf2flt.

J.6 SPI DMA changes

This patch contains the changes we made to the SPI driver and the peripheral DMA controller.

J.7 Tests

This directory contains the source code for the tests.