České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Martin Meloun**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Zprovoznění OS Linux na mikrokontroléru ColdFire MCF5484**

Pokyny pro vypracování:

1. Seznamte se s architekturou ColdFire firmy FreeScale a dostupnými dokumenty k vývojovému kitu M5484LITE.
2. Vyhledejte maximum informací a rozpracovaných projektů k této architektuře a na základě těchto informací zprovozněte aktuální verzi jádra 2.6 na vývojovém kitu.
3. Začleňte se do vývojové komunity a upravte a doplňte kódy do podoby začlenitelné do hlavní vývojové řady.

Seznam odborné literatury:

[1] MCF548x Reference Manual, FreeScale, 2009
[2] ColdFire Family Programmer's Reference Manual, FreeScale, 2009
[3] Embedded Linux kernel and driver development, http://free-electrons.com/training/drivers
[4] Linux BSP for MCF5484LITE, MCF5475/85EVB, FreeScale 2008

Vedoucí: Ing. Pavel Píša

Platnost zadání: do konce zimního semestru 2011/2012

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 11. 10. 2010

Poděkování

Děkuji Ing. Pavlu Píšovi Ph.D. za hodnotné rady a odborné vedení mé práce. Dále děkuji svému otci, Ing. Michalu Melounovi, za hodnotné rady a poskytnutí firemního hardwaru a softwaru.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

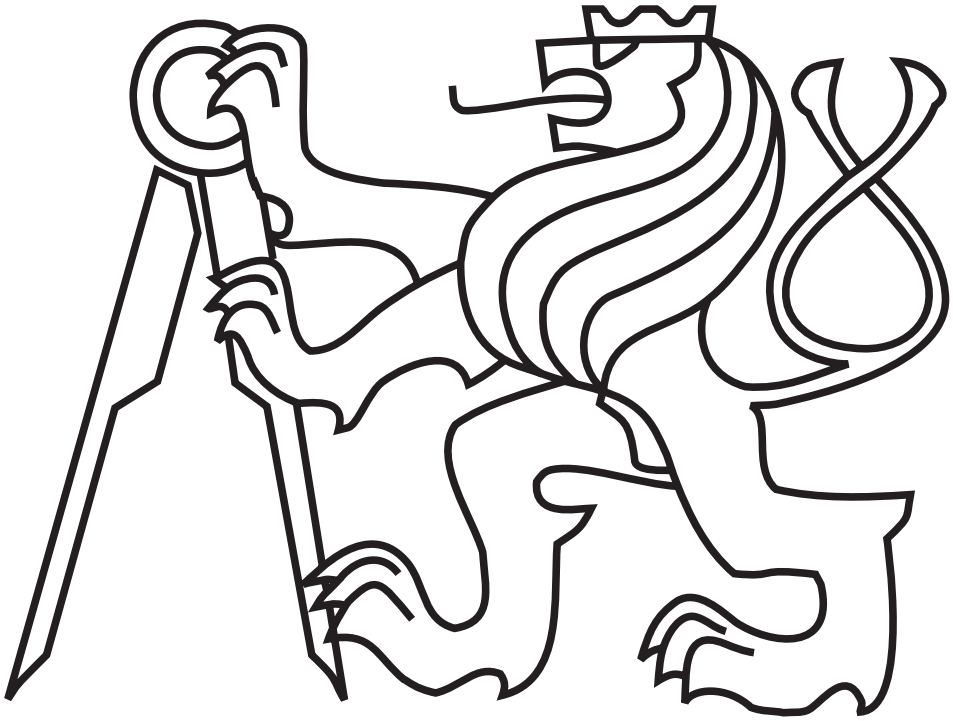
V Praze, dne 27.5.2011 
…….
podpis

Èeské vysoké uèení technické v Praze, Fakulta elektrotechnická, Katedra øídicí techniky

Bachelor Thesis

MCF5484LITE LINUX KERNEL 2.6.37 PORT

Martin Meloun



Academic Year: 2010/2011

Supervisor: Ing. Pavel Píša, Ph.D.

Contents

1

Abstract

1.1 Czech

Cílem práce je naportovat aktuální verzi Linuxového jádra pro Coldfire V4e procesory s podporou jednotky správy paměti. Coldifre V4e procesory byly navrženy pro běh plnohodnotného operačního systému, ale kvůli firemní politice FreeScale žádný operační systém nepodporuje Coldfire V4e ve své hlavní vývojové verzi způsobenou především kvůli nemožné úbržbě kódu portace Linuxu od FreeScale.

1.2 English

The main goal is to port current version of Linux kernel to Coldfire V4e processors including support for Memory Management Unit. Coldfire V4e has been designed to run operating systems but due to FreeScale software developing politics no operating system currently supports this platform in mainline version mainly because their Linux ports are not maintainable.

2

Introduction

2.1 Coldfire Platform

The **ColdFire** derives from the **CISC m68k** microprocessor architecture (Harvard type) manufactured for embedded systems development by **Freescale Semiconductor** (formerly the semiconductor sector of **Motorola**). Main advantage over other m68k processors is their speed: **MC68060** is clocked to 75 MHz (without overclocking) while ColdFire can be clocked up to **300 MHz** (MCF5484LITE is clocked to 200 MHz). MCF548x is from the **v4e** generation launched in 2000, which is using 32-bit instruction set. It features memory management unit, floating point unit and cache (32 kiB for both data and instructions by 8 kiB pages at minimum, virtually accessed, physically tagged) as main advantages over previous versions. MCF5484LITE peripherals also include a programmable serial controllers, CAN, Ethernet, USB and PCI. ColdFire mainly utilize Background Debug Module as their low-level debugging module, newer models can however utilize JTAG as well. MCF5484LITE has only 4 MiB NOR flash memory (unlike regular boards containing one more flash memory sized from 4 MiB to 16 MiB), 64 MiB SDRAM, 32 kiB SRAM.

2.2 Instruction Set

Coldfire is using Harvard architecture having separated instruction and data memory. It is using 8 general purpose 32-bit data registers **D0-D7**, 7 general purpose 32-bit address registers **A0-A6**, 32-bit full descending stack pointer **A7**, 32-bit program counter **PC**, 16-bit status register **SR** and condition control register **CCR**. Unlike for regular m68k processors instructions are only 2, 4 or 6 bytes wide and are characterized as "Variable-Length RISC" by Freescale. Missing variants of many instruction operating with different operand size, many instructions acting only on registers, reduced addressing modes, complete lack of rarely used instructions - this all was a result of plaform optimization based principles in RISC processors and instructions generated in C code translation. Supervisor programming model was also simplified to two stack pointers (there is no interrupt stack pointer). Several instructions behave differently from the m68k, like MULU, MULS (both multiply), ASL and ASR (arithmetic shift left / right) will not set overflow flag, which has to be noted or code sequences working on m68k may yield an incorrect result on Coldfire. Note that m68k assembler instructions are using source as its first operand in contrast to many architectures like ARM.

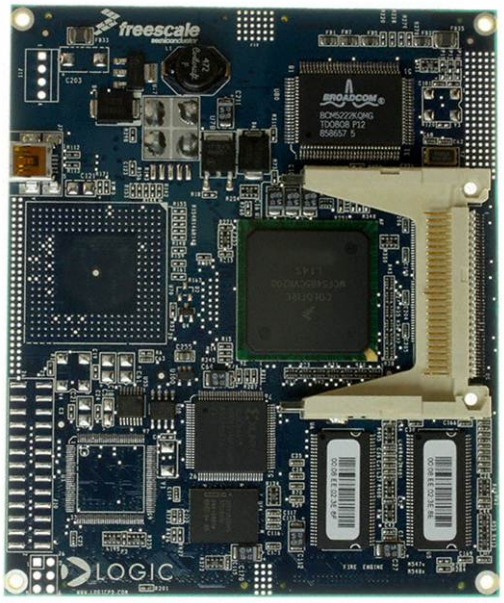


Figure 2.1: MCF5484 Lite Board

2.3 Development Prerequisites

The main prerequisite is low-level knowledge of C language. This means knowing the principles of the compiler and having experience with assembler in general on any platform (then you have to learn m68k instructions and relation between Coldfire and them). Why is knowledge of principles of the compiler so important? Because there is no guarantee that it is going to be flawless and if there is a bug in the compiler you are going to be the one to fix it because **GCC** (GNU Compiler Collection) and **binutils** (binary utilities) are open source projects. From C language you need to know compiler attributes (both as flags passed to gcc or inside C code) and being able to analyse problematic C code in assembler. And of course theoretical knowledge of principles inside Linux kernel is a required knowledge as we are going to manipulate with that.

3

Board Support Package

3.1 Overview

The evaluation kit contains a complete software support package containing technical reference manual and a software development kit including **U-Boot** (Universal Bootlader), **Linux kernel 2.6.25**, root filesystem binaries and a toolchain. The build system is shared throughout the whole family and thus configurable for the specific board. It also contained everything already prebuilt which I've primarily tested with success (there was only a requirement to override command line in U-Boot and to send root filesystem from a **tftp** (trivial ftp) server). Linux kernel in version 2.6.25 was distributed as a set or **incremental patches** which is a better approach unlike just distributing patched code which Motorola (FreeScale is Motorola semiconductor branch) still practices on kernels for their Linux phones. For future reference I shall use "BSP" as Board Support Package in this chapter.

3.2 Technical Reference Manual

Technical reference manual is the most important part of the BSP. It's primary objective is to define the functionality of the MCF548X processors for use by software and hardware developers. It is divided into chapters per subsystem, totalling around 1000 pages where it describes behaviour and all possible settings of all registers present on the board, complete instruction set, signal description and their timing and mechanical data about the board. The manual describes theory of operation in block diagrams with several simple usage examples.

3.3 GNU Build System

Each GNU build system (called toolchain) consists of two necessary parts which are binutils and gcc. Binutils provide support from translating assembler code into raw binary, archiver for static libraries and linker for linking objects files to create an executable and binary. GCC provides support for higher level languages, for our purposes it's C and C++ languages. When choosing components of your build system, one of your priorities is to ensure the build system either fully and supports your platform, to be working under the premise that compiler is creating valid code and thus in case of a bug you search in your code for the cause, or be able to reliably diagnose potential compiler failure. I've already experienced a compiler failure on GCC 4.4.0 for ARM with Android EABI (Extended Application Binary Interface; this defines how for example parameters are passed during function call etc.). In that case the stack protector was bugged and the binary was crashing upon returning from a certain function (stack protector code is automatically generated by GCC unless it's turned off). This one is fairly easy to identify because it crashes exactly at the place where it is bugged. Another relatively good scenario is when GCC creates invalid assembler directive. This mainly draws attention towards GCC C compiler as it's unlikely that binutils would incorrectly translate assembler directive into wrong binary representation. Secondly BSP uses an outdated version of GCC which is likely not supported by current Linux kernel or C library and thus I've automatically discarded BSP toolchain and decided to compile my own. The FreeBSD port by Miracle Group s.r.o., which is described later in chapter 4.2, resulted in several critical patches for GCC for Coldfire platform and these were committed since branch 4.5 (note extended Coldfire support since that version) and thus I've used version **4.5.2**. At the time version 4.6.0 was available as well but having some experience from ARM 4.4.x and 4.5.x versions (another bug I experienced was on 4.4.3 in invalid kernel module structure) I chose the older one with 2 revisions for having potentially less issues.

3.4 Compilation of GNU Toolchain

GNU Toolchain consists from two compulsory compulsory components, binutils and gcc as stated above, and two optional components which is OS userspace headers and C library. At first you configure and build binutils. After that you specify either "without-headers" flag to gcc or import userspace headers from OS environment. This is a function of the Linux kernel (command **make headers_install** after configuring it) or other operating systems. After that you compile gcc (only compiler, command **make all-gcc**). Many tools (including C library or bootloaders)

require you to build **libgcc**. This library consists of mostly arithmetic operations that the specified platform cannot make directly, exception handling and some miscellaneous operations. However if gcc was configured to enable shared libraries then when compiling libgcc it will try to compile a shared version of it as well and failing on the impossibility to link it with C library, which is likely requiring part of the library as well. This is a little pitfall in the build system, another one I've noted is not fully working **fixincludes**, an utility to create default interface if not specified in imported headers, for libgcc. This is used for the case when configuring with "without-headers" flag. Otherwise you export headers from the C library prior compiling libgcc and compile a static version of it (command **make all-target-libgcc**). Then you optionally configure and compile C library reconfigure gcc to support shared libraries and recompile it.

3.5 GCC configuration

The compiler is configured as cross-compiler, meaning that it compiles on a workstation for a different system than it runs on. It can be divided into three parts: configuration triplet cpu-vendor-OS specifying the target environment; disabling unnecessary features and lastly default flags passed to the compiler. Please note that regarding the configuration triplet that vendor part can be anything or omitted and that **linux-gnu** is an option for OS part. At first I decided to look on the configuration of GCC inside the BSP and found:

```
configure -build=i686-pc-linux-gnu -host=i686-pc-linux-gnu
-target=m68k-linux-gnu -enable-threads -disable-libmudflap
-disable-libssp -disable-libgomp -disable-libstdcxx-pch
-with-arch=cf -with-gnu-as -with-gnu-ld -enable-languages=c,c++
-enable-shared -enable-symvers=gnu -enable-__cxa_atexit
-disable-nls -prefix=/opt/.../m68k-linux
-enable-languages=c -disable-libffi
```

I've omitted unimportant parts of the configuration. Note twice and contradictory against each other supplied "enable-languages" parameter and disabling C++ library while enabling C++ language. Also note the lack of sysroot argument (which is used to define the root folder of your cross compiler under which you have the standard directory structure). Mainly specifying "linux-gnu" as OS is not supposed to be used to be built things like U-Boot. Thus I've scratched the supplied build system completely and used my own configuration for building U-Boot and Linux kernel

```
configure -with-sysroot=/coldfire/sysroot/
```

```
-prefix=/coldfire/tools/elf -target=m68k-elf
-enable-languages=c -enable-c99 -disable-nls -disable-libgomp
-disable-libssp -disable-libmudflap -disable-shared
-disable-threads -disable-multilib -without-headers -with-arch=cf
```

and the following one for userspace binaries, with **glibc** as a C library in version 2.13 and with using kernel 2.6.37 userspace headers.

```
configure -with-sysroot=/coldfire/sysroot/linux
-prefix=/coldfire/tools/linux -target=m68k-m68k-linux-gnu -enable-languages=c,c+
-enable-c99 -disable-nls -disable-libgomp -disable-libssp
-disable-libmudflap -enable-shared -enable-threads
-disable-multilib -with-arch=cf
```

3.6 Linux Kernel 2.6.25

Supplied Linux Kernel was the main source of my development despite the flaws it has (described in chapter 6). In this section I shall not analyse functional flaws in the code. As stated in overview the source was supplied as a set of incremental patches which were exported from a development repository. The patches add support for two Coldfire V4e core families MCF5445X and MCF547X / MCF548X. The affected parts of Linux kernel structure is low-level architecture based code and drivers. My first step was examining low-level boot code. Entry points are written in assembler (files **head.S** and **entry.S**). These two files were forked into newly created subdirectory. Comparing them with the generic m68k ones; head.S turned out to be a fork with minimal changes (assembler difference) and which should have been accomplished by patching the original entry.S file to distinguish based on current configuration definitions; head.S was written specially for Coldfire and part of it was rewritten to C code which is a more appropriate solution for maintaining and readability. However I've noticed several major faults in the boot-code. First it was hardcoded to be booted from U-Boot and its non-standard way, by passing a structure, command line and initial ramdisk on stack pointer. Using a different bootloader, like **CoLiLo** (Coldfire Linux Loader) caused the processor to hang indefinitely on bus response (this can be forcibly terminated by XLB arbiter or using BDM (Backgroud Debug Module)) when attempting to read the sturcture. Moreover the structure U-Boot passes when giving control was imported into the kernel and was only used for setting MAC addresses for FEC (Fast Ethernet Controller). Normally Linux kernel reads boot records behind **bss** section, and you're supposed to override them by appending a newly created boot record. Examining the rest of the patches

in low-level system support showed that page size was at its minimum allowed size, which is 8 kiB. I have also found mess in code for cache handling and in confusing merge of the source codes handling cache on MCF547X / MCF548X and MCF5445X. MCF547X / MCF548X cache code has been rewritten unlike MCF5445X but both version were not properly isolated. After that I have examined drivers and narrowed myself to basic peripherials, such as USB, Serial line and Ethernet (not all features, such as secure boot, are available on MCF5484LITE plus they are as an addition to the system). I have found two drivers for Serial line for Coldfire boards. In fact it was one driver which was rewritten by Greg Ungeger who not working in FreeScale, and that revision was applied for m68knommu Coldfire branch, while m68k still was using the old one. Ethernet driver (Fast Ethernet Controller) has its issues as well. The other two peripherals do not have issues. Then I look on design patterns. The m68knommu Coldfire branch was using "mcf" prefix for platform specific headers and these are mostly only mapping platform specific registers but you still have to manually include them in source files. The m68k Coldfire branch didn't follow that pattern and created its own with "cf" prefix and what is worse, some (not all) were forked platform generic headers them and making small changes. Lastly after walking through the patches I've discovered that a portions of code (for example the cache code rewrite noted several lines above) that was rewritten or removed from further patches. Thus I have decided not to develop the port by "pushing every minor change to git repository" but rather pushing blocks of finished code. Concluding this, it serves well as a source of tested and working code but there still lot of things to be fixed.

3.7 U-Boot

U-Boot supports this board in it's working tree I have only decided to use the precompiled binary to verify that supplied binaries are indeed working.

4

Existing Colfdire Linux Projects

4.1 uClinux

Aside from the board support package, which was be described earlier in detail, there exist two public projects; first, the **m68knommu** branch inside Linux kernel and second, **uClinux**, which stands for Linux for microcontrollers (the letter "u" is in fact standing lowercase Greek letter "Mu"). The main disadvantage of both projects is lack of support for memory management unit. Due to relatively low difference of these two projects (most notable would be that uClinux is using its own C library) they have been united since kernel version 2.6.29 and I'll refer to these as uClinux from now on.

uClinux supports Coldfire processors up to MCF540X family which is backwards compatible with previous generations of Coldfire processors and significantly different from MCF548X family. An examination of the source code pointed out several things. There was some code for MCF547X / MCF548X, although just pieces of the code in a manner of unfinished project (missing defconfig for example). Secondly, unlike the board support package, there is a configuration switch for booting from U-Boot. Another major difference between 2.6.25 and 2.6.37 kernels is that architecture specific headers are united in 2.6.37 for m68k and m68knommu systems. A sort of unhappy solution is partially "mixed" design pattern when determining which header belongs to what system. Several headers are distinguishing based on configuration definitions and have a "nommu" and "mmu" version, while platform specific headers are still prefixed in name and respective source code files must include them manually however they do not contain forked code as the board support package. Despite that I don't think that it's good to have a header "cache.h" and another header "mcfcache.h" both containing different pieces of definitions.

uClinux can be run properly on system with memory management unit, having it disabled. Note

that mainstream architectures like ARM are supported even with memory management unit enabled. Originally one of the possibilities for Linux kernel development was to port uClinux properly to MCF548X but it was decided to use full capabilities of Coldfire V4e core and create a system supporting MMU.

4.2 FreeBSD Port

FreeBSD was ported to Coldfire MCF548X by Miracle Group s.r.o. company to be used on their boards. It is utilizing memory management unit and includes support for basic peripherals with the exception of Cache, which is labeled as unfinished yet. FreeBSD is still using monolithic kernel (think of a kernel being as a one large program) unlike Linux, which is using modular kernel. It practically means that is not useful for more than studying techniques that were used when dealing with architectural flaws and apply them on Linux port if possible. This FreeBSD port is licensed under simplified BSD license and is used by Miracle Group s.r.o. privately and thus they do not maintain any open source repository.

4.3 FreeScale Updated Linux Kernel 2.6.31 Port

Around first quarter of 2010 FreeScale updated their kernel from 2.6.25 to 2.6.31 for MCF547X / MCF548X but did not update the board support package with it (and I didn't find any reference towards that kernel on FreeScale website). This kernel used the new serial driver previously noted as being used only m68knommu branch. This project mainly had fixed Fast Ethernet Controller driver and was already using updated serial driver. There also was some update to cache handling code. It also had a cleaner way of updating vanilla 2.6.31 kernel. There are just 7 patches, instead of hundreds, updating the existing files. Rest of the files is distributed in archive. I've also preferred this version for updating KConfig (kernel configuration file, contains list of options and their dependencies).

5

Preliminary Development

5.1 Workstation

The 2.6.37 has been developed on my laptop running Windows 7 operating system. I have used a OpenSUSE distribution, in which I've set up a toolchain compiling for Coldfire, described in section 3 in a virtual machine to compile the Linux kernel (as it's not recommended to compile under cygwin). All other tools for communication and debugging were developed under .NET framework and run Windows.

5.2 Background Debug Module

The Background Debug Module (BDM for future reference) is the most powerful debugging utility, allowing one to break the execution, read and modify register contents and mainly to recover the unit in cases of corrupted bootloader. It uses 26pin connector to communicate with external hardware, however only 6 pins are required to use most of its features. BDM uses three types of commands. Commands such as setting program counter, D0-D7 and A0-A7 registers etc. require interrupting the execution (the processor to be halted). Other commands, mostly setting peripheral registers, steals the processor clocks (that means inject themselves to the execution queue). Lastly there are commands such as program counter tracing that are executed asynchronously with the execution queue and there require full pinout.

The board package supplied a BDM to parallel cable converter by P&E Microcomputer Systems and software for recovery flashing. However I've used a custom board from Miracle Group s.r.o. which they use for developing their Coldfire boards. This board serves as BDM to USB converter, with reduced pinout and having asynchronous commands not avaiable. Additional

functions is a measuring module, which isn't used for this project. This board itself runs on Coldfire MCF52223 (V2 core). Supplied driver and firmware included support for most of BDM commands and flash programming for older chipsets (Coldfire Flash Module). On newer chips you reprogram the flash by first sending the data to RAM and with a simple program to write the contents RAM to the flash memory and executing it. This way of flashing is pretty slow and is recommended only to be used for recovery of the unit.

Another important part of debugging is software support - and how much of it is actually needed. Ideally full blown support for GDB (GNU DeBugger) would be the most powerful option. There exists a port for GDB to run under BDM (m68k-bdm-elf-gdb) requiring a BDM Linux driver, which wasn't written by Miracle Group s.r.o.. The reason is problematic debugging of an operating system. BDM is mainly suitable for stepping through the code however for operating system, with the exception of boot code, this is made practically impossible due to interrupt handling. So you are logging debug information over a serial console. This reduces the need for BDM only for recovery flashing, potentially debugging bootloader in case there are issues with it and debugging OS boot code. Based on the fact that I am not writing these parts from scratch I've decided not to spend time for GDB support and only created a simple .NET program for basic BDM operations, meaning reading and writing registers, reading and writing memory, execution control, dumping stack and recovery functions (actually I did not need to make a recovery flash yet due to technique I've used for developing, that's why there is only button to dump flash memory on the picture). This utility was developed based on board communication interface provided by Miracle Group s.r.o..

Testing the debugger board showed two issues. One of them was non-functioning hardware breakpoint signal, making it impossible to externally break the execution. This is perhaps useful if the processor is stuck in infinite loop as you rather want to have it interrupted on predefined places using halt instruction. Also this is problematic if there was a bootloader malfunction to start the processor in halted mode, but for this case P&E Microcomputer Systems convertor would do it's job and halt the execution at startup (which it does if disconnected from the other

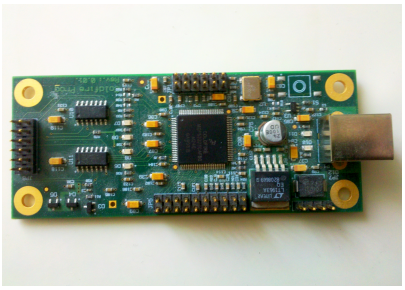


Figure 5.1: BDM Debugging Board

side). A bypass for this situation is to always start in halted mode, and having halt as a first instruction in the bootloader written in flash memory. The other one was lack of support TA (Transfer Acknowledge) signal. This signal forcibly terminates bus cycle by generating an error. This signal is used when you for example address invalid physical memory and the processor indefinitely waits for response (if you address invalid virtual memory than memory managment unit will generate the fault). For this case I've set up the XLB arbiter which will forcibly terminate bus cycle after a certain period. This can be used as a watchdog to restart the processor or just to trigger an interrupt (this requires valid stack pointer).

5.3 Bootloader

The board came with pre-installed with **dBUG** a small monitor program by Freescale. I have found two other bootloader projects, CoLiLo, a simple bootloader designed to boot uClinux and more complex U-Boot bootloader, widely used across all platforms and with support for filesystems. When choosing the bootloader I required the ability for the bootloader to be able to boot from both flash and RAM so I can send it via ethernet from tftp server or load from flash. Also I required for the bootloader to receive the kernel from ethernet. This both is to speed up the development as prevent myself from recovery flashing via BDM. This was supported by CoLiLo as U-Boot, with several excpetions on ARM processors (like Texas Instruments OMAP, where the initial bootloader is called **xloader** which loads U-Boot to RAM and executes it), doesn't support booting from RAM (I discovered the "U-Boot requirement" of 2.6.25 kernel in the BSP package after choosing CoLiLo as my bootloader). Nevertheless I've made several minor modifications to CoLiLo (interrupt handling, default configuration etc.). The reason I didn't want to use U-Boot initially was that it contained some compilation errors which means

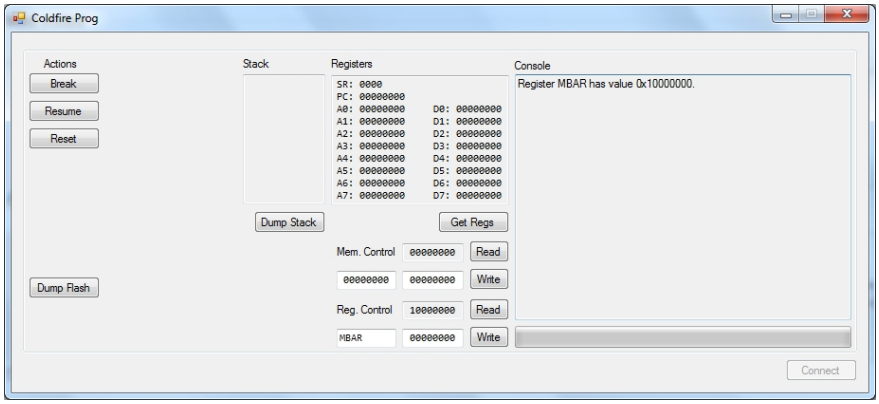


Figure 5.2: BDM Debugger Application

it wasn't maintained for some time and thus possibly could contain bugs. Secondly I have originally misinterpreted the lack of support from booting from RAM, overall this is meant to completely skip any low-level hardware initialization and relocation to RAM, something which U-Boot is not designed for in it's initial code. This would mean slow development from flash memory and that's why I decided to skip it originally. It however later turned out that the only thing I need to do is to set it's text base to SRAM location and disable it's initialization and then I can load it to SRAM in dBUG and execute it. So at the end I have two different bootloaders for my board.

6

Linux 2.6.37 Port

6.1 GIT Repository

A git repository for the port was set up on the Department's pages and with the intention to be used to commit each modification done. This is commonly used when multiple people work on the same project so everybody has his working source code up to date. It is also used to create the final incremental patches to allow porting to later kernel version with ease. I have decided not to use a GIT Repository for my development in initial stage for several reasons as I work on it alone and a lot of my development still was in trial & error style - this would be reflected in the patches as a mess should there be another modifications between the committing patch and removal patch. I have noticed that in cache code in incremental patches supplied from Freescale. This basically voided the possibility to walk through the patches in order to resolve mismatches so I've decided to result in lesser amount of patches having bigger blocks of code. These patches are then used to make the port for current version of Linux kernel, which is 2.6.39 at the time of writing the thesis. Initially I started developing on Linux kernel version 2.6.35 in the phase to look inside the kernel code and study the code more precisely . The actual development started on 2.6.37 kernel version but also many modifications were tested in the old 2.6.25 kernel as it was the bootable system.

6.2 Linux Development

In order of the port to be effective the code must be working correctly and be maintainable. Therefore my main aim was to remove all forking inside the existing code, mainly the headers. I've focused only on the main peripherials, meaning serial line, ethernet etc., there is no support

for PCI, CAN, any hardware cryptographic support etc. Firstly I removed all traces of MCF548X from m68knommu branch to prevent possible ambiguous situations among both of the branches. After that I've analysed what things were changed and why. Typically these were assembler differences in low-level code and different peripherial usage (each platform has its own registers etc.). In some cases I've found problems with lack of cleanness of the code serving as a barrier for any further development (this is the case FEC driver in Board Support Package for example). I've focused on working with code for MCF547X / MCF548X and leaving MCF5445X code as it was keeping the possibility of making the code work on that board as well even though it was discarded mostly in 2.6.31 port. The thing I changed first was removing the hardcoded bootcode expecting U-Boot and using it as configuration option as in m68knommu branch. Interestingly in 2.6.31 kernel the revised FEC driver already didn't need the passed U-Boot structure but this was not fixed. Currently the kernel will only accept command line and init ramdisk image (if not empty) from U-Boot if configured to. This also included. Second thing was reducing the forking to a minimum that is necessary and uniting the headers with current m68knommu branch where possible (this however still has it's pitfalls where m68k and m68knommu in the same header have code serving a bit different purpose). Other things I was editing partially fixed was cache code (which still includes hacks in clearing and pushing). Serial line driver now contains definitions for MCF547X / MCF548X. I've also modified cache code which is at this point only flushing more than asked for (performance loss over maintaining discarded data). What remained unfinished is proper port of memory management handling code including updating it current code in higher layer, common for all architectures. Several commented code examples from 2.6.25 BSP kernel:

Listing 6.1: Entry point with hardcoded U-Boot boot arguments

```
ENTRY(__start)
/* Save the location of u-boot info - cmd line, bd_info, etc. */
  movel %a7,%a4   /* Don't use %a4 before cf_early_init */
  addl  #0x00000004,%a4 /* offset past top */
  addl  #(PAGE_OFFSET-CONFIG_SDRAM_BASE),%a4  /* high mem offset */

/* Setup initial stack pointer */
  movel #CONFIG_SDRAM_BASE+0x1000,%sp

/* Setup usp */
  subl  %a0,%a0
  movel %a0,%usp
```

Listing 6.2: Original U-Boot handling code notice the nested default command line which is not going to work on any other workstation

```
/* ethernet mac addresses from uboot */
unsigned char uboot_enet0[6];
unsigned char uboot_enet1[6];

/*
 * UBoot Handler
 */
int __init uboot_commandline(char *bootargs)
{
  int len = 0, cmd_line_len;
  static struct uboot_record uboot_info;
  u32 offset = PAGE_OFFSET_RAW - PHYS_OFFSET;

  extern unsigned long uboot_info_stk;

  /* validate address */
  if ((uboot_info_stk < PAGE_OFFSET_RAW) ||
      (uboot_info_stk >= (PAGE_OFFSET_RAW + CONFIG_SDRAM_SIZE)))
    return 0;

  /* Add offset to get post-remapped kernel memory location */
  uboot_info.bdi = (struct bd_info *)((*(u32 *)(uboot_info_stk)) +
     offset);
  uboot_info.initrd_start = (*(u32 *)(uboot_info_stk+4)) + offset;
  uboot_info.initrd_end = (*(u32 *)(uboot_info_stk+8)) + offset;
  uboot_info.cmd_line_start = (*(u32 *)(uboot_info_stk+12)) + offset;
  uboot_info.cmd_line_stop = (*(u32 *)(uboot_info_stk+16)) + offset;

  /* copy over mac addresses */
  memcpy(uboot_enet0, uboot_info.bdi->bi_enet0addr, 6);
  memcpy(uboot_enet1, uboot_info.bdi->bi_enet1addr, 6);

  /* copy command line */
  cmd_line_len = uboot_info.cmd_line_stop - uboot_info.cmd_line_start;
  if ((cmd_line_len > 0) && (cmd_line_len < CL_SIZE-1))
    len = (int)strncpy(bootargs, (char *)uboot_info.cmd_line_start,\
          cmd_line_len);

  return len;
```

```
}

/*
 * This routine does things not done in the bootloader.
 */
#if defined(CONFIG_M54451)
#define DEFAULT_COMMAND_LINE "debug root=/dev/nfs rw nfsroot
   =172.27.155.1:/tftpboot/redstripe/rootfs/ ip
   =172.27.155.51:172.27.155.1"
#elif defined(CONFIG_M54455)
#define MTD_DEFAULT_COMMAND_LINE "root=/dev/mtdblock1 rw rootfstype=
   jffs2 ip=none mtdparts=physmap-flash.0:5M(kernel)ro,-(jffs2)"
#define DEFAULT_COMMAND_LINE "debug root=/dev/nfs rw nfsroot
   =172.27.155.1:/tftpboot/redstripe/rootfs/ ip
   =172.27.155.55:172.27.155.1"
#elif defined(CONFIG_M547X_8X)
#define DEFAULT_COMMAND_LINE "debug root=/dev/nfs rw nfsroot
   =172.27.155.1:/tftpboot/rigo/rootfs/ ip=172.27.155.75:172.27.155.1"
#endif
```

Listing 6.3: Cache code flushing more than is needed

```
/* Push n pages at kernel virtual address and clear the icache */
/* RZ: use cpush %bc instead of cpush %dc, cinv %ic */
void flush_icache_range(unsigned long address, unsigned long endaddr)
{
#ifdef CONFIG_COLDFIRE
// JKM -- hack until new cpushl stuff is in
//  cf_icache_flush_range(address, endaddr);
  flush_icache();
#else /* !CONFIG_COLDFIRE */

...
}

void cache_clear (unsigned long paddr, int len)
{
#ifdef CONFIG_COLDFIRE
// JKM -- revise to use proper caching
//  cf_cache_clear(paddr, len);
  flush_bcache();
#else

...
}
```

Listing 6.4: MMU context stealing (ported from different architecture) which is a performance loss

```
/*
 * Steal a context from a task that has one at the moment.
 * This is only used on 8xx and 4xx and we presently assume that
 * they don't do SMP.  If they do then thicfpgalloc.hs will have to
    check
 * whether the MM we steal is in use.
 * We also assume that this is only used on systems that don't
 * use an MMU hash table - this is true for 8xx and 4xx.
 * This isn't an LRU system, it just frees up each context in
 * turn (sort-of pseudo-random replacement :).  This would be the
 * place to implement an LRU scheme if anyone was motivated to do it.
 *  -- paulus
 */
void steal_context(void)
{
  struct mm_struct *mm;
  /* free up context `next_mmu_context' */
  /* if we shouldn't free context 0, don't... */
  if (next_mmu_context < FIRST_CONTEXT)
    next_mmu_context = FIRST_CONTEXT;
  mm = context_mm[next_mmu_context];
  flush_tlb_mm(mm);
  destroy_context(mm);
}
```

6.3 GCC Development

Working with GCC 4.5.2, with the exception of issues with "long long" type (64-bit integer) producing easily diagnosable assembler directive error for which I've found a patch, I did not diagnose any major bugs meaning it was compiling correctly. However the compiled code has a lot of redundant code when it comes to register allocation. Here is a sample C function

Listing 6.5: Sample function in C

```
unsigned long long __attribute__((noinline)) use_ll(unsigned long long
    ll)
{
  return ll*3;
}
```

which GCC translates into assembler as

Listing 6.6: Translated to Assembler

```
use_ll:
  link.w %fp,#-16
  movem.l #60,(%sp)
  move.l 8(%fp),%d2
  move.l 12(%fp),%d3
  move.l %d2,%d0
  move.l %d3,%d1
  move.l %d1,%d4  ;d1 = d3 = d4
  add.l %d4,%d4
  subx.l %d4,%d4
  neg.l %d4
  move.l %d0,%d5  ;no need to use d5 here at all
  add.l %d5,%d5
  move.l %d5,%a0
  move.l %a0,%d5  ;d5 already had a0 value
  or.l %d4,%d5
  move.l %d5,%a0  ;d5 is put into a0,
                  ;which is later put back to d0
  move.l %d1,%d4  ;d3 wasn't used yet
                  ;d1 still has d3 value
```

```
    add.l %d4,%d4
    move.l %d4,%a1
    move.l %a0,%d0
    move.l %a1,%d1  ;a1 isn't used anymore
    add.l %d3,%d1
    addx.l %d2,%d0
    movem.l (%sp),#60
    unlk %fp
    rts
```

in which I commented redundant directives. Fixing this issue would significantly reduce size of the binaries.

7

Conclusion

I examined supplied Linux kernel in version 2.6.25 and started to clean up and fix stated issues in order to prepare the code to be added to mainline repository. I have also gathered information about other projects for Coldfire which were a FreeBSD Port, uClinux and updated FreeScale 2.6.31 kernel. I also checked for proper support in GNU tools and pinpointed issues to be fixed. However memory management unit support was not finished which means to update the properly with code changes on higher layers and bypass architecture flaws in software (for example problem with kernel stacks mapping using translation look-aside buffer which either have to be flushed on context switch, or be mapped as locked (and there might be a need to steal context from another task) both leading to performance loss; alternative is to map a single 16 MiB locked page in high memory behind kernel text pages and use that to distribute space for kernel stacks, here it saves performance but can overflow causing the kernel to panic). Kernel on the supplied CD will not compile itself in its current state (I have excluded all bypasses to make it compile without being functional).

8

References

8.1 References

[1] MCF548x Reference Manual (FreeScale)
[2] Linux BSP for MCF5484LITE (FreeScale)
[3] Linux 2.6.31 Port for MCF547X / MCF548X (FreeScale)
[4] Linux Kernel (www.kernel.org)
[5] uCLinux (www.uclinux.org)
[6] U-Boot (www.denx.de/wiki/U-Boot)
[7] CoLiLo (Ken Treis)
[8] FreeBSD Port (Miracle Group s.r.o.)

8.2 Repositories

[1] Linux 2.6.31 Port for MCF547X / MCF548X
(dev.openwrt.org/browser/trunk/target/linux/coldfire)