HÖGSKOLAN I BORÅS

INSTITUTIONEN INGENJÖRSHÖGSKOLAN

# Embedded Platform Software Development

## uClinux on a ColdFire v2 platform

**Henric Eriksson, Pär Leandersson**

**2008-05-22**

**The Thesis comprises 15 credits and is a compulsory part in the Master of Science with a Major in Electrical Engineering with specialization in Biomedical Engineering, 60 credits**

**No. 3/2008**

## Acknowledgements

We would like to extend special thanks to Martin Voss who has been our supervisor and made this project possible. We also want to thank Jonas Svennebring and Stanescu Razvan Tudor at Freescale as well as people on the uClinux-dev mailing list for helping us out during the project.

This report has been purposely stripped of product- and technology-specific details because of non-disclosure agreements. A separate document containing such details has been created which is not available to the general public.

## Abstract

This thesis involves working with an embedded hardware platform based on the Freescale ColdFire v2 core, namely the MCF5208EVB development board. It's an architecture that allows the use of running a special branch of Linux called uClinux which is specifically tailored to run on systems without a MMU (Memory Management Unit). It brings certain limitations to the system which needs to be taken into consideration when developing applications. On the target platform a measuring application is running which generates data files that are processed and presented using a web server. The data files are generated as XML files and are in the user's browser combined with an XSLT style sheet to transform the raw data into an easy to use web interface.

# Table of Contents

**Acknowledgements** ..... **i**
**Abstract** ..... **ii**
**Table of Contents** ..... **iii**
**1. Introduction** ..... **1**
**2. Hardware platform** ..... **2**
2.1 MCF5208 and MCF5208EVB ..... 2
2.1.1 Memory ..... 3
2.1.2 Ethernet ..... 3
2.1.3 BDM/JTAG Port ..... 4
2.1.4 Jumpers ..... 4
2.1.5 I/O Header (CN1) ..... 5
2.1.6 Power/reset switches ..... 6
2.1.7 DIP switch ..... 6
2.1.8 Watchdog Timer ..... 7
2.1.9 ZigBee ..... 7
2.2 Custom MCF5208 platform ..... 7
**3. Embedded Linux** ..... **8**
3.1 MMU based CPU ..... 8
3.2 Non-MMU based CPU ..... 8
3.3 Limitations in uClinux ..... 9
3.4 File systems in uClinux ..... 10
3.4.1 ROMfs ..... 10
3.4.2 RAMfs ..... 10
3.4.3 JFFS2 ..... 10
3.5 Boot loader ..... 11
3.6 Console ..... 12
**4. Application Development** ..... **14**
4.1 Virtual Machine ..... 14
4.2 Tool chains ..... 14
4.2.1 2.95.3 ..... 15
4.2.2 4.1.1 ..... 15
4.2.3 CodeSourcery 4.2.1 ..... 15
4.3 uClibc ..... 15
4.4 Transfer executable to target ..... 16
4.4.1 Samba ..... 16
4.4.2 NFS ..... 17
4.4.3 FTP ..... 18
4.5 Stack usage profiling ..... 18
4.6 SBCTools and Eclipse ..... 21
4.6.1 Debugging with gdb ..... 22
4.7 Custom makefile ..... 22
4.8 CodeWarrior ..... 23
4.8.1 Debugging with AppTRK ..... 25
**5. Customizing the uClinux image** ..... **28**
5.1 Kernel versions ..... 28
5.1.1 2.4.x ..... 28
5.1.2 2.6.x ..... 28

5.2 Distributions ..... 28
5.3 Build example ..... 29
5.4 Downloading image to target ..... 29
5.5 Debugging the kernel ..... 30
**6. Web interface ..... 31**
6.1 Boa web server ..... 31
6.2 Dynamic web pages ..... 32
6.2.1 CGI ..... 33
6.2.2 XML and XSLT ..... 33
6.2.3 Graphics using SVG ..... 34
6.3 Practical example ..... 36
**7. Issues encountered ..... 39**
7.1 Issues kernel 2.6.x ..... 39
7.2 Issues kernel 2.4.x ..... 40
7.3 Issues in user land ..... 41
**8. Conclusions ..... 43**
**References ..... 44**
**Appendix A ..... 1**

**Appendix A ..... M52277EVB On-Board BDM Setup and Usage**

# 1. Introduction

The scope of this report is to present how a modern low cost ColdFire based system can be used to run Linux as an embedded operating system to add advanced features such as web interface, ftp server etc. The practical work was done in spring 2008 at Br. Voss Ingenjörsfirma AB located in Borås, Sweden (www.brv.se). The main purpose of the thesis work was to find a new way of presenting data produced by a measurement-system running uClinux on a ColdFire v2 MCU. This required an upgrade of the currently used uClinux version, tool chain and user application executing in the system. The new presentation of measurement data should be presented in a way so it would contain all the information the system produces, and at the same time be easy to overview and easy to access. As there already was a custom made program designed for handling the presentation of data from the instrument, it was not a question if it would be possible to design the new interface; instead the problem was how to do it in the best possible way to meet the given requirement of easy access and simplicity. The main technique used is a web-based approach where XML played an important role. To be able to incorporate all the wanted functionality SVG was added to expand the functionality to include graphics drawing.

# 2. Hardware platform

The ColdFire family plays a key role for Freescale's (Motorola's semiconductor division changed name to Freescale) 32-bits microprocessor family which has been frequently used in the electronic industries for around ten years. They have gained trust and popularity due to flexibility in terms of memories, system modules and communications peripherals. Freescale gives the opportunity to buy product solutions ready to integrate, use and debug. Many devices support several connection alternatives such as Ethernet, USB, CAN and PCI, coupled with sophisticated development tools. And in a broad range of price and performance, these things will most likely have ColdFire stay in the front and keep expanding. A summary of the different ColdFire versions can be seen in Table 2.1.

| Version | Launch year | Description |
|---|---|---|
| **v1** | 2006 – 12 years after the original ColdFire | A cut-down version of the v2. It is designed to easily replace the 8-bit Freescale 68HC08 processors and compete with low-end ARM chips. |
| **v2** | 1994 | The original ColdFire core.<br>Single-issue pipeline, no MMU, no FPU. |
| **v3** | | Added an optional MAC unit. |
| **v4** | | Limited superscalar core. |
| **v4e** (also called ev4) | 2000 | Enhanced version of the v4. Adds optional MMU, FPU, and enhanced MAC unit to the architecture. |
| **v5** | | Fully superscalar core. |

**Table 2.1** Coldfire version summary

## 2.1 MCF5208 and MCF5208EVB

MCF5208EVB is a development board kit, equipped with the MCF5208 processor (member of the ColdFire v2 family) and shipped with all development tools necessary for hands on development of both uClinux and bare-metal applications.

The card has lots of functionalities and during this project just a couple where tested. In this chapter a short description of the MCF5208EVB will be given. Figure 2.1 illustrates the layout of the board.

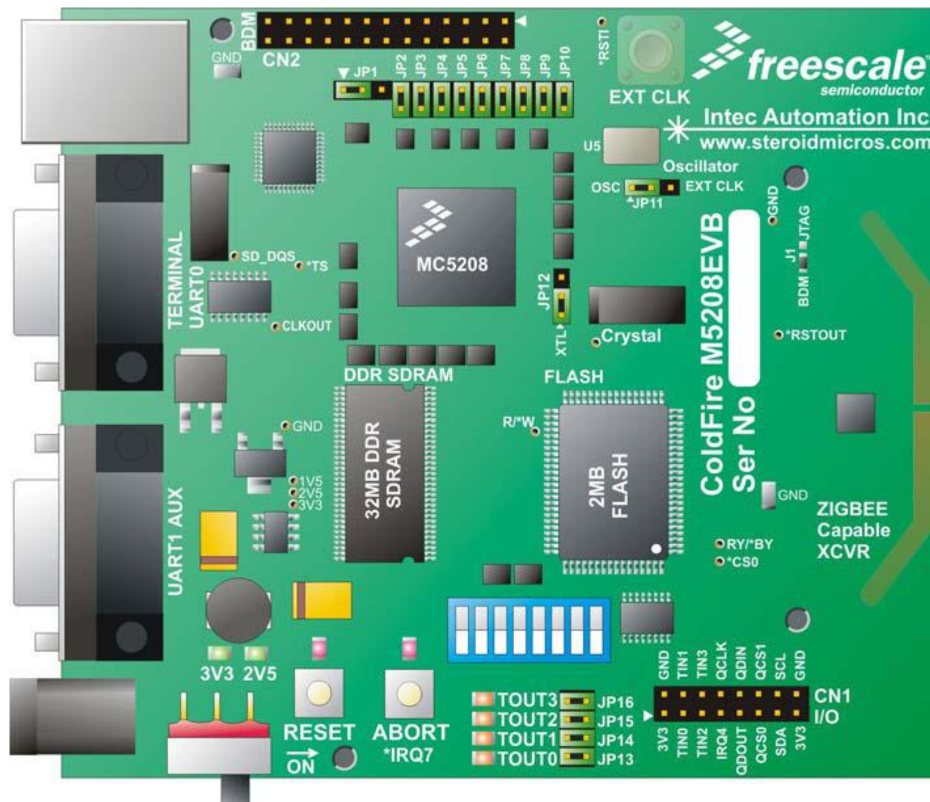


**Figure 2.1** M5208EVB development board layout

## *2.1.1 Memory*

The M5208EVB board has four memory types on board:

- 16KB Internal (on-chip) SRAM
- 128KB External SRAM (optional)
- 32MB External DDR SDRAM
- 2MB External Flash which is shipped pre-loaded with the dBUG monitor, the uClinux kernel and uClinux services. Depending on jumper settings, the dBUG either check if there is an executable OS in flash and execute (uClinux) or enter a dBUG prompt.

All the external memory devices run and interface at 2.5V. This allows a single bus to interface all devices without the need for buffers and level translators. (1)

## *2.1.2 Ethernet*

The board supports 10/100 Ethernet with the feature of auto-negotiate connection speed and RX/TX switch to adapt to the polarity of the connection. With this feature no consideration regarding network cables has to be made. The traffic can be overviewed with 2 LEDs on the front of the connector which is summarized in Table 2.2. (1)

| LED | State | Significance |
|---|---|---|
| **Speed** | On | 100Mbps |
| | Off | 10Mbps |
| **Link/Traffic** | On | Good link |
| | Blink | Traffic |

| Off | No link |
|---|---|

**Table 2.2** Ethernet indicator LEDs

### *2.1.3 BDM/JTAG Port*

There is two possible ways for debugging the M5208EVB board. One is using serial connection (RS232) and the on-board dBUG monitor which is stored in the first few sectors of the Flash memory. The other possible way is to use the background debug mode (BDM) that in comparison with dBUG can have total control over the processor even after a program has crashed. BDM can also be used to monitor the processor status without interfering with the execution. The same header that is used for BDM can also optionally be transformed and used as a JTAG interface. (1)

### *2.1.4 Jumpers*

There are 16 "jumpers" placed on the EVB which is used together with the switches for deciding all configurations that can be done, listed in Table 2.3. (1)

**BDM/JTAG selection**

| | |
|---|---|
| JP1 | Selects either pin 6 or pin 24 of the CN2 (BDM/JTAG Port) to be connected to the TCLK/PSTCLK signal from the M5208EVB. TCLK is used when JTAG mode is enabled; PSTCLK is used when BDM mode is enabled (see BDM/JTAG Port section). |
| JP2 | This jumper is required for some of the legacy BDM cables that connect pins 9 & 25 of the BDM interface internally. More recent cables support both core & I/O voltages. Please check with your BDM cable supplier. The BDM cable supplied with the M5208EVB supports both core and I/O voltages and requires JP5 to be fitted. |

**dBUG mode selection**

| | |
|---|---|
| JP3 | When fitted this jumper causes dBUG monitor to automatically load a program and run it. Typically this program is uClinux. When this jumper is open, dBUG monitor will not run any user program and instead display the dBUG prompt on the terminal and wait for input. |

**Test mode ON/OFF**

| | |
|---|---|
| JP4 | When fitted this jumper disables the factory test mode of the MCF5208. This jumper is normally always fitted. |

**Reset configuration switches ON/OFF**

| | |
|---|---|
| JP5 | When fitted this jumper asserts the *RCON signal and causes the MCF5208 to load the CCR register based on the signals D9, D[7:1]. These signals are conditioned out of reset by the DIP switch. |

**Memory selection**

| | |
|---|---|
| JP6 | This jumper selects between DDR and SDR mode for the SDRAM module. |

| | |
|---|---|
| | Should always be fitted to support DDR mode. |

**Power jumpers**

| | |
|---|---|
| JP7 | When fitted connects 1.5V to the PLL filter and to JP8. |
| JP8 | When fitted connects the processor core voltage to 1.5V. JP7 must also be fitted. |
| JP9 | When fitted connects the processor I/O voltage to 3.3V. Also connects JP11 to 3.3V. |
| JP10 | When fitted connects the processor external bus voltage to 2.5V. |

**Oscillator modes**

| | |
|---|---|
| JP11 | This jumper selects between an external oscillator and an off-board frequency source. This jumper is only relevant if JP15 is in position 1-2. By default it is set to position 2-3. |
| JP12 | This jumper selects between the on board crystal or an external oscillator. It is in position 1-2 by default to select the on-board crystal with a 16 MHz frequency. |

**Timer jumpers**

| | |
|---|---|
| JP13 | This jumper connects TOUT3 to a LED. It is normally fitted. |
| JP14 | This jumper connects TOUT2 to a LED. It is normally fitted. |
| JP15 | This jumper connects TOUT1 to a LED. It is normally fitted. |
| JP16 | This jumper connects TOUT0 to a LED. It is normally fitted. |

**Table 2.3** Jumper summary

### *2.1.5 I/O Header (CN1)*

On the EVB a 2x8 connector is located that contains signals for DMA timer, QSPI, IRQ, and I²C signals but it can also be configured as general purpose I/O pins.

| Pin | Signal | GPIO | Alt 1 | Alt 2 |
|---|---|---|---|---|
| 1 | 3V3 | - | - | - |
| 2 | GND | - | - | - |
| 3 | DT0IN | PTIMER0 | DT0OUT | U2TXD |
| 4 | DT1IN | PTIMER1 | DT1OUT | U2RXD |
| 5 | DT2IN | PTIMER2 | DT2OUT | *U2RTS |
| 6 | DT3IN | PTIMER3 | DT3OUT | *U2CTS |
| 7 | *IRQ4 | PIRQ4 | *DREQ0 | - |
| 8 | QSPI_CLK | PQSPI0 | I2C_SCL | - |
| 9 | QSPI_DOUT | PQSPI1 | I2C_SDA | - |
| 10 | QSPI_DIN | PQSPI2 | *DREQ0 | *U2CTS |
| 11 | U0CTS | PUARTL3 | DT0IN | QSPI_CS0 |
| 12 | U1CTS | PUARTL7 | DT1IN | QSPI_CS1 |
| 13 | I2C_SDA | PFECI2C0 | U2RXD | - |
| 14 | I2C_SCL | PFECI2C1 | U2TXD | - |
| 15 | 3V3 | - | - | - |
| 16 | GND | - | - | - |

**Table 2.4** CN1 signal summary

When handled in I/O mode the pins 7, 11 and 12 have a maximum source/sink current of 4mA while it is 8/16mA for the other ones (except 3V and ground). (1)

## *2.1.6 Power/reset switches*

All four switches are summarized in Table 2.5. (1)

| Switches | Function |
|---|---|
| **Power Switch** | M5208EVB main power switch. |
| **Reset Button** | Asserts the *RSTI signal forcing the MCF5208 and peripheral systems to reset. |
| **Abort Button** | Asserts the *IRQ7 signal causing an interrupt in the MCF5208. This interrupt is handled by dBUG monitor. |
| **Configuration Switch** | Determines the out of reset configuration of the MCF5208. |

**Table 2.5** Power/reset switch summary

## *2.1.7 DIP switch*

The switch can be used for reset configuration of the processor. All options available are shown in Table 2.6 where the red markings are default values. (1)

| Switches | | Function |
|---|---|---|
| SW1-1 | | PLL Mode |
| OFF | | 166.67MHz Core bus, 83.33MHz External Bus operation |
| ON | | 88MHz Core bus, 44MHz External Bus operation |
| SW1-2 | | Oscillator Mode |
| OFF | | Crystal oscillator mode |
| ON | | Oscillator bypass mode |
| SW1-3 | SW1-4 | Boot Port Size |
| OFF | OFF | 16-bit port |
| OFF | ON | 32-bit port |
| ON | OFF | 32-bit port |
| ON | ON | 8-bit port |
| SW1-5 | | Output Pad Drive Strength |
| OFF | | High drive strength |
| ON | | Low drive strength |
| SW1-6 | | LIMP Mode |
| OFF | | Normal operation; PLL drives internal clocks. |
| ON | | LIMP mode; low-power clock divider drives internal clocks. |
| SW1-7 | | Oscillator Frequency Select |
| OFF | | 16MHz is used as input to processor |
| ON | | 16.67MHz is used as input to processor |
| SW1-8 | | Chip Select Configuration |
| OFF | | A[23:22] = A[23:22] |
| ON | | A[23:22]=*FB_CS[5:4] |

**Table 2.6** Summary of the different DIP switches

### *2.1.8 Watchdog Timer*

In case a program enters an unexpected state there is a watchdog timer (WDT) implemented in the MCF5208 to time out and generate a reset. If the watchdog is enabled it cannot be disabled without resetting the MPU. And an enabled WTD also has to be serviced periodically by writing a 0x5555 and 0xAAAA sequence to a watchdog service register to avoid the reset. (1)

### *2.1.9 ZigBee*

ZigBee is a wireless communication protocol with low power consumption. A MC13192 ZigBee Capable Transceiver chip and printed circuit board antenna have been integrated on the M5208EVB with the purpose to have the possibility of evaluating it together with the MCF5208 processor. (1)

## 2.2 Custom MCF5208 platform

Because of the non-disclosure agreements regarding this project no detailed information about the measurement platform is given in this report. The hardware is built around the same microprocessor as the EVB card and in a very similar manner, with some exceptions in memory size and peripherals. An overview of the platform is shown in Figure 2.2.

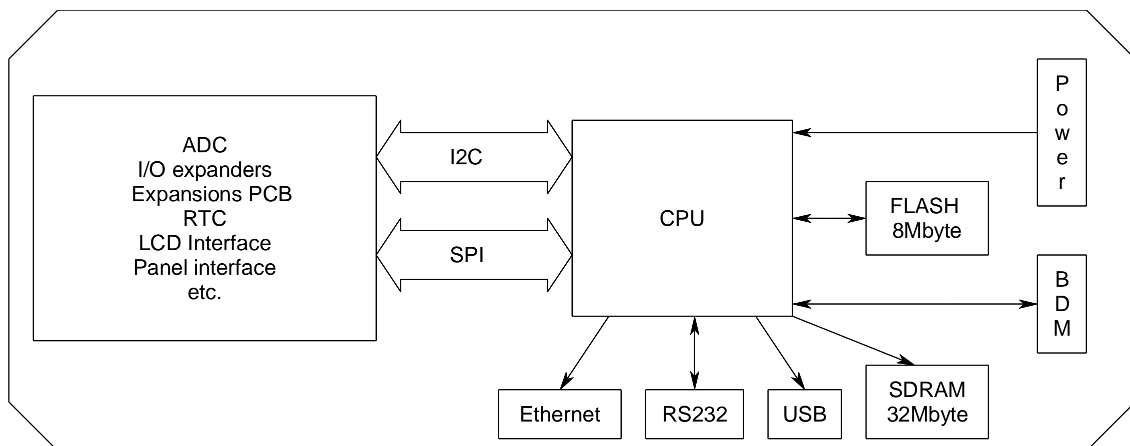


**Figure 2.2** Project platform

# 3. Embedded Linux

Linux has been in use in PCs around the world for many years. Even from its early days, Linux was ported to be run on the handheld computer PalmPilot and has since been ported to a whole range of different architectures. It has turned out be a mature, high-performance and stable alternative to other operating systems. Nowadays you can find Linux running on a myriad of different embedded products such as mobile phones, PDAs, networking products, printers etc and its usage areas continues to expand.

Linux is an extremely flexible operating system; it can be tailored to almost any architecture. Since the only thing that is in direct contact with the hardware is the kernel, that's what has been customized to work on a myriad of different hardware platforms each requiring slight adjustments to work. The source code for all this is openly available and can be modified to support new hardware given enough knowledge of C programming and of the kernel's internal workings. All this flexibility makes it possible to run Linux on virtually any hardware platform you can think of, be it an off the shelf platform or a custom design.

In addition to being a flexible operating system, it is also scalable and can be used in any device, from small consumer-oriented devices to large, heavy-iron, carrier-class switches and routers. As opposed to many of the commercial proprietary embedded operating system, Linux can be used royalty-free. (2)

## 3.1 MMU based CPU

An MMU, or Memory Management Unit, is a hardware device used to allow the kernel to manage and control the address space it uses. The MMU itself is a very complex device, allowing the use of so-called virtual memory. Virtual memory is in the simplest of terms, the physical available memory plus any additional storage medium which has been designated for use as swap-space. In an MMU based system memory allocation does not have to be contiguous, i.e. memory does not have to be allocated in one big chunk; it can instead be allocated by using bits and pieces of the available physical memory. Another important aspect of having an MMU is that the kernel can set access rights to memory segments. Certain memory segments can be set as read-only for instance. This is especially useful when you have multiple running user applications at the same time, because they cannot access the other applications memory and thus run the risk of causing a system-wide crash.

User applications in an MMU based system can, thanks to the virtual memory, be run anywhere in the virtual memory. The base address of the application can be well beyond the physical memory address space and the memory used by the application can grow and shrink as necessary. (2)

## 3.2 Non-MMU based CPU

An MMU is a very complex device and are only available in the larger, more complex, CPUs. Because of this you have to consider a few things when choosing a CPU. More often than not it would not be required to use a powerful enough CPU that implements an MMU because the applications being run on it won’t make use of the available

performance. Another issue to consider is the memory protection features of having an MMU. If the only applications running on the platform will be developed by you and made sure to work within the limitations of not having an MMU, why spend the extra money to use a CPU with an MMU?

While on the subject on non-MMU based CPUs and Linux there is actually a distribution of Linux that will run on CPUs without an MMU. This distribution of Linux is called uClinux, pronounced as “you see Linux”. uClinux is a distribution specifically tailored to work on microcontrollers without the luxury of having access to an MMU. It has been around since 1998, starting around the time Linux kernel 2.0 was actively used. The core of uClinux along with all of the available tools and utilities has been slimmed down immensely compared to regular Linux and a fully working image can be generated which is less than 1 MB in size using kernel 2.4.x, all the while keeping much of the functionality available to users of regular Linux distributions. (3)

For more information regarding uClinux visit the official website at http://www.uclinux.org.

## 3.3 Limitations in uClinux

uClinux is primarily developed to be run on MMU-less devices, that is to say, devices without a built-in memory management unit. The implications of having a Linux platform without a MMU is quite severe, for instance memory in an uClinux platform cannot be protected. There is no virtual memory and as such, running applications cannot dynamically allocate more memory during runtime because doing so might use another application's memory. This limitation also applies to the application’s stack. In turn this means that a poorly written application could potentially crash the entire system. Because of not having virtual memory, memory fragmentation is a big problem. Each application needs a contiguous memory space to work with and this gets increasingly difficult the longer the system is running and the more applications get started.

Another very important limitation in uClinux can be found in use of the *fork()* command often used in Linux applications, especially in daemons. *fork()* is a system call used in Linux to clone the application calling it, creating an identical version of it running independently on the system. The parent (the one calling *fork()*) can create many children and each child run completely independently from the parent. In uClinux this system call is not available, instead a similar system call is available called *vfork()*. While similar in many ways, there are a few important differences. *vfork()* does create a child process but it does not make it run independently of the parent. Instead the parent is suspended until the child exits or calls *exec()*, a system call used to start a new application. In addition it does not have its own memory and space but instead runs on the parent’s stack and is using the parent’s memory and data. In essence, this means that a child can corrupt the parent process so that when the child exits, the parent will crash. To avoid this, the child needs to use a special exit call *_exit* when finished as well as making sure not to make changes to any existing global data structures or variables.

These limitations, especially the *fork()*/*vfork()* issue, can make porting applications to uClinux incredibly easy or almost impossible. (4)

## 3.4 File systems in uClinux

### *3.4.1 ROMfs*

Romfs is a read-only minimalistic file system that was developed for Linux, but is most common when talking about uClinux nowadays. The reason is that all Linux systems, including uClinux needs an initial file system to run some programs at start up. In the case of uClinux where the amount of memory is limited romfs is a good choice. Romfs is therefore included in almost every uClinux distribution, but not always compiled in (enabled). The fact that it is read-only means that the image has to be built beforehand and have to be rebuilt when changes are done to it. It also eliminates the possibilities to use UNIX permissions. But these “drawbacks” has proved to be not all that crucial in the benefit of space-efficiency. (5)

### *3.4.2 RAMfs*

Linux has a number of disk caching mechanisms that it uses to cache files to memory before writing the data to disk. These features have been reused to create a completely RAM-based file system. In the case of a normal hard drive data is read from a backing store (usually the hard drive itself) and is then stored in memory, although marked as free-able. Similarly data which is to be written to files are also stored in memory for caching purposes. This data remains there until the memory subsystem reallocates memory or claims it to be used for something else.

Ramfs makes use of this feature but instead takes away the backing store. What this means is that data is cached in RAM but is never marked as free-able and as such is never able to be reallocated or claimed by the memory subsystem, thus creating a dynamically resizable RAM-based file system. (6)

### *3.4.3 JFFS2*

Flash memory is used extensively when talking about embedded systems today. And the application in this thesis work is not an exception. The flash memory is a block device, which uses sectors of same or different sizes.

Flash-memories is a non-volatile memory, i.e. when power is lost the information is not lost as in RAM memories. Flash memories have relatively fast read access-time, but very slow write performance. Flash memories have better resistance against kinetic shock than ordinary hard drives, since it’s a pure semiconductor device.

The flash memories have two major drawbacks. The first is caused by what mentioned earlier about the block architecture where the blocks tends to be a lot larger than most of the files to be stored, in fact it turns out that almost all of the files to be stored from the system is rather small. The problem here lies within that when a flash memory shall change one of these small files then a whole block has to be copied and rewritten after that the referred file has been changed. This is very time-consuming in comparison with e.g. a hard disk-drive and with that comes an increased time-window for data corruption from power loss which is more common in embedded systems.

The second issue is the fact that the blocks/sectors of a flash memory gets worn out when they are being written to and erased. Today a common number for write-cycles is

around 100000 times, which is quite a lot but definitely something to take into consideration. (2)

To solve these issues JFFS (Journaling Flash File System) was designed for Linux 2.0 and later ported to newer versions. JFFS was developed by a Swedish company called Axis Communications AB. The improvements listed below are a summary of improvements that came with the JFFS that eliminated the problems mentioned earlier in the text.

- Wear levelling
- No data corruption on sudden power loss
- Direct use of the MTD-level APIs instead of going through the flash translation layers

Wear levelling improves the problem with a limited number of write-cycles by distributing the writings equally over the blocks. Despite this fact, flash-memories should only be used as write-occasional and special care should be taken with processes that writes frequently e.g. systems logs.

A functionality that makes these things possible is the log-structures that store all changes made to files in the system. Below is a list of what's logged for making it possible to re-create files when a read is intended, but also used for block erase.

- Identification of the file to which the log belong
- Version that is unique per log belonging to particular file
- Metadata such as timestamp
- Data and data size
- Offset of the data in the file

JFFS2 (second-generation Journaling Flash File System) were released with Linux kernel version 2.4 to overcome an issue in JFFS regarding compression and were soon the file system of choice.

JFFS2 manages erase blocks by using logs to assign them to lists:

- A clean list only contains valid logs if they have not been invalidated by newer ones
- A dirty list contains the logs which are obsolete and can be garbage collected
- A free list does not contains any logs

Erases are handled by a garbage collector who is initialized as a separate thread when JFFS2 is mounted. It reserves five blocks for doing garbage collection and erase blocks from the dirty list with an exception in one percent of the cases, where a block is picked from the clean list to ensure wear levelling. JFFS2 are also able to give compression possibilities using zlib and rubin. (7)

## 3.5 Boot loader

There are lots of boot loaders on the market today; both open source and commercial versions. As this project started with the MCF5208EVB development board from Freescale, shipped with the "dBUG" boot loader that was the one chosen.

In general the first task for a boot loader after power up of the processor board is to initialize hardware elements needed for the system to run and also to fetch start-up code from predefined on-board storage device (often flash-memory). This start-up address is often predefined and used for the initial boot loader code with the purpose for configuring the memory interfaces, SDRAMs etc. When the initialization steps are accomplished the boot loader's next task is to locate, load, and pass execution to the primary operating system. There are similarities between a boot loader and the BIOS known from a traditional PC platform. One thing that is special for the boot loaders in embedded systems is that it´s often overwritten when OS takes control.

dBUG is a program that in the case of the development board mentioned above resides in the bottom of flash taking approximately 256KB in possession. At startup it maps 16MB onto SDRAM as follows. The lower 128KB of SDRAM is used for dBUG's vector table, data and stack (0x40000000 to 0x4001FFFF). Consequently this memory space is out of bounds to the user program. It expects the user program to always start at (0x40020000) in SDRAM. A user program can use the MCF5208's on-chip SRAM for stack space, since it is very fast and dBUG does not use this memory.

On start-up dBUG check if the JP3 is connected. If JP3 is fitted (ON) dBUG automatically load and run a program from a defined memory start address. If the jumper is open (OFF) one will instead enter a dBUG prompt from terminal for entering commands to dBUG.

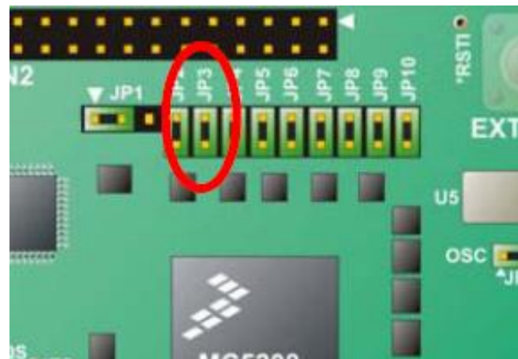


**Figure 3.1** Example picture of on board jumper appearance

dBUG can be serviced directly through any serial consol program using RS232 interface e.g. Windows HyperTerminal (115200bps). With the dBUG command interface running on the console one is enable to load, debug and run programs without the use of specialized debug programs. But also set up a couple of board parameters such as baud rate server IP and watchdog. (2)

## 3.6 Console

dBUG does not have any graphical user interface (GUI) and although uClinux can be configured to run a minimalistic GUI this is not commonplace. So the communication between board and client machine was in this project managed via RS232 and a client terminal program named PuTTY which is shown in the Figure 3.2.

```
COM3 - PuTTY

dBUG> set server 192.168.100.69
dBUG> set baud 115200
dBUG> show
    watchdog: off
        base: 16
        baud: 115200
      server: 192.168.100.69
      client: 192.168.100.200
     gateway: 192.168.100.1
     netmask: 255.255.255.0
    filename: test.s19
    filetype: S-Record
     ethaddr: 00:50:C2:65:20:03
         dns: 192.168.2.1
dBUG> dnfl imagez.bin
Offset:  0x00000000
Downloading Image 'imagez.bin' from 192.168.100.69
TFTP transfer completed
Read 845143 bytes (1651 blocks)
Must erase complete sectors (0x00720000 to 0x007EFFFF)
Continue (yes | no)? y
```

**Figure 3.2** Showing putty console window with some basic dBUG commands

When the flash is programmed one uses the command "gfl" to run the system. If everything is done properly something like Figure 3.3 shall be displayed, of course depending on system and versions chosen.

```
COM3 - PuTTY

TCP: Hash tables configured (established 2048 bind 4096)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
VFS: Mounted root (romfs filesystem) readonly.
Freeing unused kernel memory: 28k freed (0x400fa000 - 0x40100000)
eth0: config: auto-negotiation on, 100FDX, 100HDX, 10FDX, 10HDX.
Welcome to
          ____ _  _
         /  __| ||_|
    _   _| |  | | _ ____  _   _  _  _
   | | | | |  | || |  _ \| | | |\ \/ /
   | |_| | |__| || | | | | |_| |/    \
   |  ___\____|_||_|_| |_|\____|\_/\_/
   | |
   |_|

For further information check:
http://www.uclinux.org/


BusyBox v1.5.1 (2008-04-25 17:14:30 CEST) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

#
```

**Figure 3.3** The actual uClinux console after the system has fully booted

# 4. Application Development

In Linux you have two contexts of application execution; user space and kernel space. In essence what this means is that in order for a user application to be able to access kernel features it has to work through an API which itself operates in kernel space. The reason to have this separation between user and kernel space is two-fold. First of all to make sure that user applications does not have access to critical kernel data structures and secondly to hide the complexities of accessing the hardware directly to the user application. For a user application it is not necessary to have knowledge about how to access for example a hard drive prior to opening and writing data to a file, instead you let the device driver handle the complex operations necessary to do that and rely on the much easier to use API which is available.

Application development for embedded Linux can be done on either a Linux or Windows host. On a Linux host you have a wider variety of choices on how you wish to develop the application because of the fact that you are working in a native Linux environment with native Linux application development. There are however also IDEs for Windows specifically designed to facilitate development of user application for embedded Linux.

## 4.1 Virtual Machine

The idea of running multiple machines on the same hardware is something that has been taken more seriously in recent years. Previously if you wanted to run a different operating system or wanted to run a number of applications on a clean newly installed PC you had to have a physical PC available to install on. With the advent of virtualization technologies that is a thing of the past. Although virtualization has been around since around the 1960s when IBM used it for mainframe partitioning, it has really been taken to a whole new level in the last two decades. Some of the more prominent companies in virtualization technologies are VMware, Connectix (later acquired by Microsoft) and Microsoft, each having developed applications that provide more or less full virtualization of x86 architecture hardware. (8)

As can be seen, virtualization can be a very powerful tool. The ability to run multiple different machines with different operating systems on one and the same physical machine is unrivalled. In the case of this thesis, Linux was run as a virtual machine on top of Windows Vista with the help of an application called VirtualBox. That allows making use of Linux-based compilers and IDEs without having to switch to another computer.

## 4.2 Tool chains

A tool chain is a package of tools that is used when developing applications and operating systems. For the subject issued in this report the tool chain should be tailored for development in uClinux. Usually including what’s listed in Table 4.1. Even though the development platform limits the choices, there proved to be some dilemma according to which of the different versions that should be used. There had to be taken in consideration that the tool chain should fit together with the uClinux distribution. In the

end there were three versions of interest and each of them is the described in the chapters below.

| Utility | Description |
|---|---|
| **GNU make** | Automation tool for compilation and build |
| **GNU Compiler Collection (GCC)** | Suite of compilers for several programming languages |
| **GNU Binutils** | Suite of tools including linker, assembler and other tools; |
| **GNU Debugger (GDB)** | Code debugging tool |
| **GNU build system (autotools)** | Autoconf, Autoheader, Automake, Libtool |

**Table 4.1** List of the tools used that comprises a tool chain

### *4.2.1 2.95.3*

The 2.95.3 tool chain was used when compiling the application that was already running on the measurement-system initially. This is an old tool chain that doesn't support compilation of 2.6.x kernels, but has been around for a long time and is therefore well tested.

The 2.95.3 tool chain can be retrieved at:

http://uclinux.org/pub/uClinux/m68k-elf-tools/

### *4.2.2 4.1.1*

The 4.1.1 tool chain was the latest release at the start of this project. It is therefore up to date and one can hope for a better response in case of trouble, but it would not be as widely tested as 2.95.3 although that does not necessarily mean it has more issues.

The 4.1.1 tool chain can be retrieved at the same web address as the 2.95.3 tool chain.

### *4.2.3 CodeSourcery 4.2.1*

This tool chain was included in the package available with CodeWarrior available from Freescale. It was the tool chain later used during the project because it was the only tool chain that made it possible to debug with the AppTRK application distributed alongside with CodeWarrior.

The CodeSourcery 4.2.1 tool chain can be retrieved at their official website:

http://www.codesourcery.com/

## 4.3 uClibc

uClibc and uC-libc is C runtime libraries that is developed and ported from the original GNU C library (glibc) used for Linux. These two libraries are supposed to be used for developing embedded Linux systems were memory usage is of great importance. uClibc is smaller then glibc but still support almost every function including support for shared libraries and threading. It is also compatible with a lot of the popular

processors on the market today. uC-libc is even more slimmed-down with the trade-off that only ARM and m68k processors are supported. uC-libc's shared library functionality is also more limited comparing with uClibc. (9) (10)

## 4.4 Transfer executable to target

Cross compiling is best done one a Linux based machine. At the client machine side, a basic text editor of choice can be used to write the program. When finished the cross complier is used to compile a binary flat file suitable for executing on the uClinux platform. See Code 4.1, Code 4.2 and Code 4.3 below for a simple example. The executable file can be transferred to the target with several different methods, each described in the chapters below.

```
# include <stdio.h>

int main(int argc, char* argv[])
{
  printf("Hello World!");
  return(0);
}
```

**Code 4.1** main.c located on the host PC

```
# m68k-uclinux-gcc -m5307 -o main main.c
```

**Code 4.2** Compile command line on host PC. Making an executable file called main which can be executed on the target board

```
/mnt/workspace> ./main
Hello World!
```

**Code 4.3** Execution command line and result from the terminal window of the target board will look like this. This of course assume that /mnt/workspace has been mounted using NFS or Samba beforehand

When a project grows lager it is almost a necessity to have a professional IDE which is being described in separate section.

### *4.4.1 Samba*

To enable Linux and Windows systems to share files between them a tool called Samba is commonly used. Samba is a server which runs on Linux and allows sharing of files and printers to Windows-based systems. It works in both directions, allowing Linux-based systems to access Windows shares. But that is only one part of what Samba is capable of. Samba is also a Linux equivalent of the business servers available for Windows. It can act as an NT Domain Controller in a business network, being in charge of user accounts and authentication of all the users on the network be it Windows, Mac or Linux. (11)

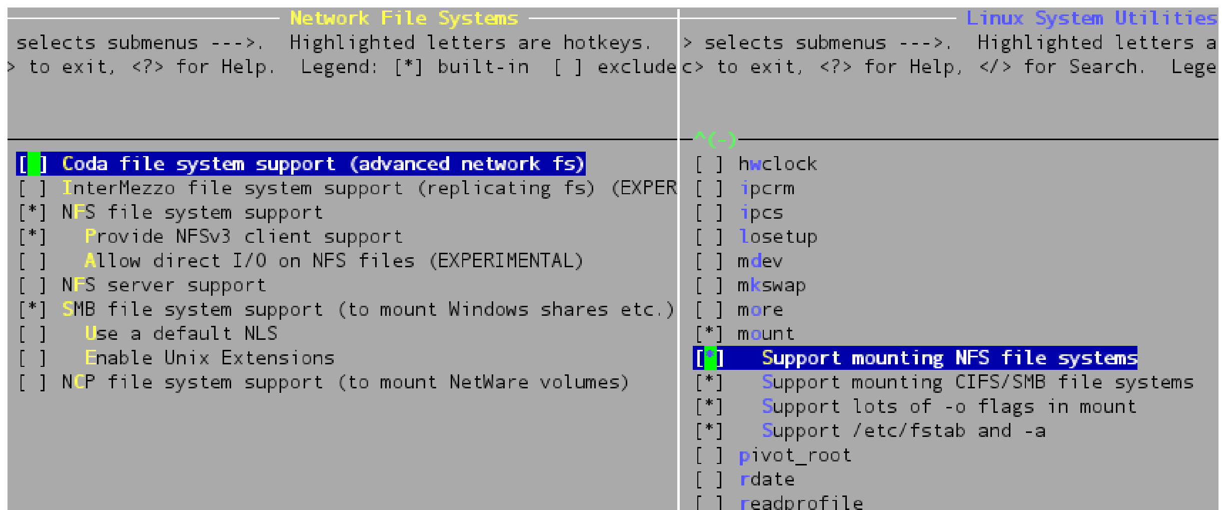


**Figure 4.1** Kernel and user application configuration settings to enable NFS/Samba support

To enable support for mounting Windows shares from Linux one need to make changes to the Linux kernel to enable the SMB file system as well as enabling support for the SMB file system in the *mount*-command. The options in question can be seen in Figure 4.1. Once the appropriate options have been enabled and the image has been generated one can mount Windows share. To do so, the command below is used.

```
/> mount -t smb -o username=User,password=Password \
   //computername/sharename /mnt/workspace
```

**Code 4.4** Command used to mount Windows/Samba shares

### *4.4.2 NFS*

NFS stands for "Network File System" and is commonly used in Linux based systems. The reason why it was added to the application platform handled in this report was from the start the benefit of using cross-compiling between client machine and the target board under the development-phase. Later during the project CodeWarrior + AppTRK which are described in a separate chapter proved to be the development environment to use. Still NFS is useful to share directories between two platforms, in this case the host PC and the target board.

Before NFS can be used on the platform it has to be enabled and configured on both the host and target. The first thing is to enable the NFS functionality on the target (see Figure 4.1 ), which is optional in 20080305 uClinux distribution.

The directories that should be shared using NFS have to be configured on the host PC and this is done in the file */etc/exports* as shown in Code 4.5.

```
# /etc/exports: NFS file systems being exported. See exports(5).
/home/directory_name
192.168.100.0/255.255.255.0(async,rw,no_subtree_check)
```

**Code 4.5** /etc/exports showing an example of how to export a file system using NFS. Note that each entry should be on a single line without newlines

After these steps are completed, the last thing to do is to mount NFS on the target board as can be seen in Code 4.6.

```
/> mount -t nfs -o nolock,rsize=1024,wsize=1024 \
   192.168.100.100:/home/directory_name /mnt/workspace
```

**Code 4.6** Command used on target board to mount the exported NFS file system on the host PC

### *4.4.3 FTP*

Ftp stands for "File Transfer Protocol" and is a standardized way to transfer files. A deep explanation of how it works is outside the scope of this report and here just some basics of how to work with ftp and uClinux is given. The first thing to do is enabling FTP in uClinux-distribution menu configuration by enabling ftpd as shows in Figure 4.2. Ftpd is short for ftp daemon and is a name to differ ftp server from client.

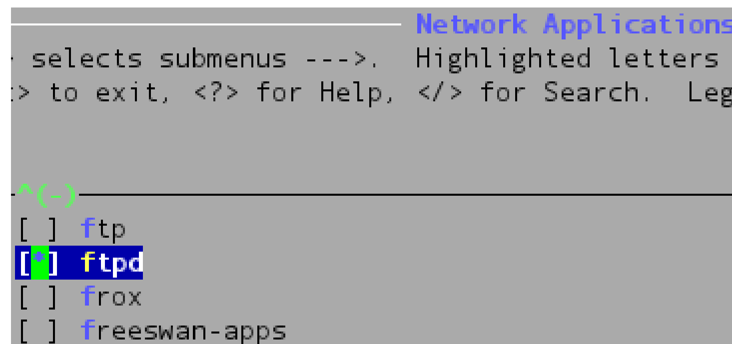


**Figure 4.2** Screen dump from uClinux configuration, enabling ftpd under network applications

When the system is up and running with the ftpd enabled, the best way to test is to get an ftp-client to handle a simple file transfer. A proper IP-address, together with login name and password has to be used to access the ftp server. IP-address is changed using the command *ifconfig* which is used to configure network interface settings, see Linux manual pages for more info. Code 4.7 below is an example how to change the IP address.

```
ifconfig eth0 192.168.100.10
```

**Code 4.7** Command used to set the IP address of a network interface. In this case, network interface eth0 is given the IP address 192.168.100.10

The user related configuration on the ftp server shall preferably be stored at some place on flash memory and linked from /etc to flash to the actual system when it´s running. The passwd-file can be found under /etc/ and is has a format which can be seen in Code 4.8.

```
username:password:UID:GID:Full Name:/home/dir:/bin/shell
```

**Code 4.8** Format of the /etc/passwd file. UID and GID are numeric representations of the user and group ID associated with the user

On the client machine, but not that often in embedded systems, there is a shadow file that is built and can be used in a similar manner as the passwd file except this file is used to contain the encrypted passwords for all users on the system.

## 4.5 Stack usage profiling

One critical issue when working with uClinux is the growth of an user application's stack. In uClinux the stack size for each application is fixed in size, with a default value of 4096 bytes. The stack size cannot grow during program execution, due to the absence of

the MMU, and if the application requires more than the allocated stack space, a potential system crash is at hand. This makes stack profiling a very interesting topic. Erwin Authried has developed a small patch for 2.4.x uClinux kernels, together with a small application that works like the *ps* command. These tools initialize the stack area with a known value, and then check how deep the stack has overwritten these values during runtime. When profiling the stack of an application, one can override the 4096 default size, to a much larger value using the *m68k-uclinux-flthdr* command, and then profile the actual stack usage for all possible runtime scenarios and then adjust the stack down as needed. The kernel patch is shown in Code 4.9.

```
diff -u linux-2.4.20/fs/proc/array.c.orig linux-2.4.20/fs/proc/array.c
--- linux-2.4.20/fs/proc/array.c.orig Mon Nov 17 20:57:46 2003
+++ linux-2.4.20/fs/proc/array.c   Sun Nov 16 23:10:30 2003
@@ -413,7 +413,7 @@
  read_unlock(&tasklist_lock);
  res = sprintf(buffer,"%d (%s) %c %d %d %d %d %d %lu %lu \
 %lu %lu %lu %lu %lu %ld %ld %ld %ld %ld %ld %lu %lu %ld %lu %lu %lu %lu
%lu \
-%lu %lu %lu %lu %lu %lu %lu %lu %d %d\n",
+%lu %lu %lu %lu %lu %lu %lu %lu %d %d %lu\n",
     task->pid,
     task->comm,
     state,
@@ -456,7 +456,8 @@
     task->nswap,
     task->cnswap,
     task->exit_signal,
-    task->processor);
+    task->processor,
+    mm ? mm->end_brk : 0);
  if(mm)
     mmput(mm);
  return res;
diff -u linux-2.4.20/fs/binfmt_flat.c.orig linux-2.4.20/fs/binfmt_flat.c
--- linux-2.4.20/fs/binfmt_flat.c.orig   Thu Nov 27 09:15:29 2003
+++ linux-2.4.20/fs/binfmt_flat.c  Thu Nov 27 09:22:20 2003
@@ -751,9 +751,11 @@
  /* zero the BSS,  BRK and stack areas */
  memset((void*)(datapos + data_len), 0, bss_len +
         (memp + ksize((void *) memp) - stack_len -   /* end brk */
-        libinfo->lib_list[id].start_brk) +     /* start brk */
-        stack_len);
+        libinfo->lib_list[id].start_brk));     /* start brk */

+    /* fill stack with 0xa5, starting at end_brk */
+ memset((void*)(memp + ksize((void *)memp)) - stack_len, 0xa5,
stack_len);
+
  return 0;
 }
```

**Code 4.9** The patch written by Erwin Authried. Saved as stackcheck.patch

To apply the patch the following command is executed:

```
uClinux-dist/linux-2.4.x # patch -p1 < stackcheck.patch
```

**Code 4.10** Command line to patch the 2.4 kernel to work with the stackcheck application

```
#include <stdio.h>
#include <ctype.h>
#include <dirent.h>

int main(int argc,char **argv)
{
    FILE *f;
    char fname[80],prog[100];
    char *q;
    int i,pid,stack;
    struct dirent *entry;
    DIR *dir;

    dir=opendir("/proc"); if (!dir) exit(1);

    printf("   PID %-30sFree Stack\n","Command");

    while ((entry = readdir(dir)) != NULL) {
        if (!isdigit(*entry->d_name)) continue;
        sprintf(fname,"/proc/%s/stat",entry->d_name);

        f=fopen(fname,"r"); if (f==NULL) exit(1);

        fscanf(f,"%d %s %*s %*d %*d %*d %*d %*d %*d %*d "
               "%*d %*d %*d %*d %*d %*d %*d %*d %*d %*d "
               "%*d %*d %*d %*d %*d %*d %*d %*d %*d %*d "
               "%*d %*d %*d %*d %*d %*d %*d %*d %*d %d ",
               &pid,prog,&stack );
        fclose(f);

        printf("%6d %-30s",pid,prog);
        if ((q=(char*)stack)){
            i=0;
            while(*q++==0xa5) i++;
            printf("%d",i);
        }
        printf("\n");
    }
    closedir(dir);
    return 0;
}
```

**Code 4.11** Stackcheck.c application code.

```
# m68k-uclinux-gcc -m5307 -o stackcheck stackcheck.c
```

**Code 4.12** Compile line for stackcheck.c

After these steps are executed the kernel is built and downloaded (see section 5). When the new kernel is up and running, stackcheck is run from the target terminal preferably over NFS.

```
/mnt/workspace> ./stackcheck
   PID Command                          Free Stack
     1 (init)                              3182
     2 (keventd)
     3 (ksoftirqd_CPU0)
     4 (kswapd)
     5 (bdflush)
     6 (kupdated)
```

```
     7 (mtdblockd)
    10 (jffs2_gcd_mtd1)
    29 (sh)                          18420
    30 (inetd)                        2979
    31 (boa)                         14239
    34 (rpciod)
    39 (stackcheck)                   2562
```

**Code 4.13** Output of the stackcheck application after running on the target board

This original source and kernel patch can be found at:

http://home.at/cgi-bin/viewcvs.cgi/midori/sources/debugtools/src/

## 4.6 SBCTools and Eclipse

SBC Tools is put together by Intec Automation from various number of tools listed below in Table 4.2, creating a development environment that is separated into two branches.

| Software | Description | License |
|---|---|---|
| **Eclipse IDE** | Eclipse is an open source project which can be found at http://www.eclipse.org. Eclipse has been modified by Intec to better support embedded development with the ColdFire microprocessor family. The unique plug-ins added by Intec are licensed under the Intec software license and are not open source. | Eclipse Public License (EPL) and Intec Automation Software License |
| **GCC** | SBCTools includes two distinct versions of the GCC compiler. One for bare board applications and one for uClinux applications. GCC is compiled by Intec against the mingw libraries for standalone execution on windows. | GNU Public License (GPL) |
| **GDB** | Two versions of GDB are included, one for serial debugging and one for TCP/IP debugging under uClinux with gdbserver. | GNU Public License (GPL) |
| **uClinux** | uClinux is a port of the Linux operating system to microprocessors without Memory Management Units (MMU). uClinux has been modified and enhanced by Intec to support features of their products and all modifications are subject to the GPL. | GNU Public License (GPL) |
| **dBUG** | The dBUG monitor allows a target board to download, run, and debug applications on the target without the use of a Background Debug Module (BDM). Intec has enhanced the dBUG monitor for tighter integration with SBCTools. The base dBUG source code can be downloaded from Freescale. | Intec Software License |
| **TCP/IP Stack (Bare board** | The TCP/IP stack for bare board applications is a port of the OpenTCP stack with some | OpenTCP License |

| | | |
|---|---|---|
| **only)** | enhancements and modifications by Intec. | |
| **Run Time Library (RTL)** | The runtime library is a set of C functions which can be called by both bare board applications and from within uClinux. | Intec Software License |

**Table 4.2** From "SBCTools User Manual Version 2.0.0"

One of the branches is supposed to be used for development of applications which run without the need of uClinux. The other one is to be used for user application development running under uClinux.

SBCTools was only used initially during the project, hence why there is only given a short summary and a glimpse of the environment (see Figure 4.3 ). More information can be found in the user manual for SBCTools distributed by Intec Automations.

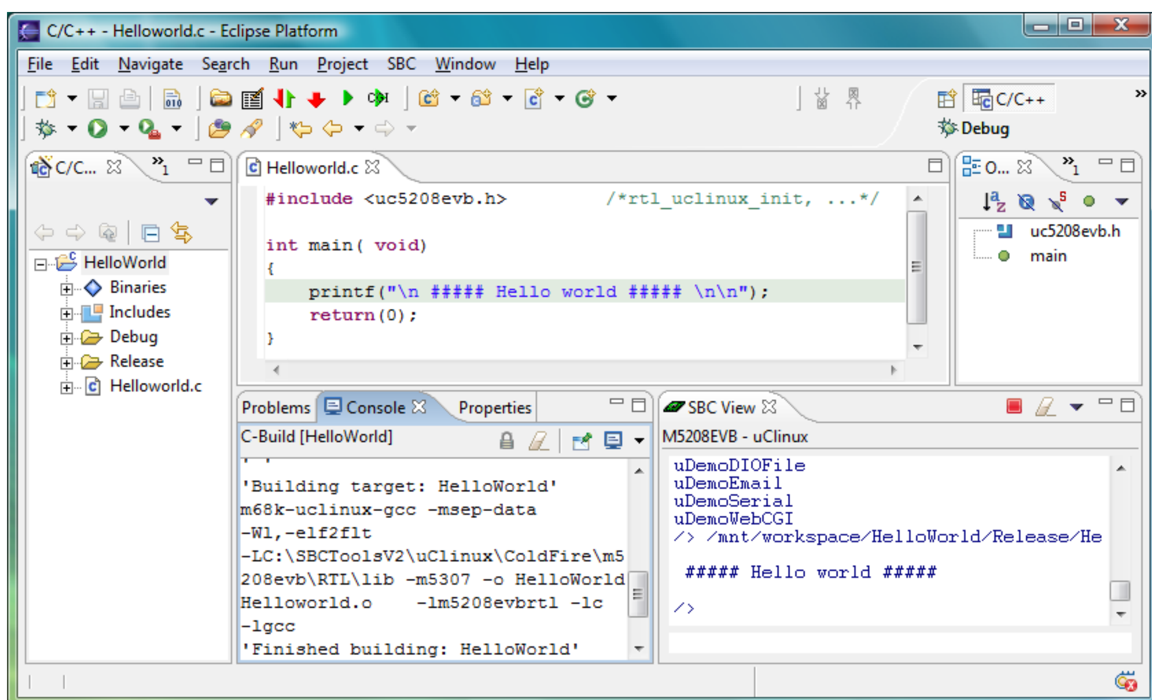


**Figure 4.3** Figure showing the IDE of SBC Tools

### *4.6.1 Debugging with gdb*

SBCTools have a conveniently integrated debugging interface. It makes use of *gdb*, the GNU Project Debugger, but hides all the inner workings of *gdb* and provides a neat and easy to use interface. It requires that *gdbserver* is available on the target platform and that the application is compiled in debug mode. Once those requirements are met, debugging an application is pretty straightforward. It allows you to set breakpoints, single step through the source code, view contents of variables and registers and much more. All in all it's a solid debugging environment which is easy to get used to and easy to use.

## 4.7 Custom makefile

When a project grows larger it´s definitely justified using an IDE, but it is not always necessary. One alternative is to use a text editor to write both programming code and

makefiles. A makefile is written to link and compile all the project files in a structured way by just executing the command *make*.

```
# Makefile example one
# compiler: m68k-elf-gcc

EXEC = main
CC = m68k-elf-gcc
LDFLAGS = -m5307 -Wl,-elf2flt
LDLIB = -lc
OBJS = main.o loop.o

${EXEC}: ${OBJS}
	${CC} ${LDFLAGS} -o ${EXEC} ${OBJS} ${LDLIB}

main.o: main.c
	$(CC) -c -m5307 main.c

loop.o: loop.c
	${CC} -c -m5307 loop.c

clean:
	-rm -f ${EXEC} *.elf *.gdb *.o
```

**Code 4.14 Example of a basic Makefile**

If either of the object to the right (source) of ":" is changed later in time then the object on the left side (target) at make time the target is rebuilt.

The first execution line implies that main.o and loop.o shall be built before comparing their timestamp with main. Which in this case will mean that if main.c is newer than main.o, then main.o will be rebuilt, leading to a rebuild of main. And the same thing applies for loop.

## 4.8 CodeWarrior

CodeWarrior is an integrated development environment for embedded systems. It's available for Macintosh, Windows, Linux and Solaris-based PCs. CodeWarrior was originally developed for the Macintosh platform, by a company called Metrowerks. There has been versions including a various number of programming languages such as Pascal, Object Pascal, Objective-C, and Java but the main focus for the tool has always been C and C++. Around year 1999 Metrowerks was acquired by Freescale and since then they have developed and distributed CodeWarrior. Freescale chose to concentrate their continued development of CodeWarrior towards embedded systems and in 2005 the last Macintosh version was released.

For this particular project the Linux version of CodeWarrior IDE (Version 5.9.0 Build 2482) was the one used. As CodeWarrior is a sophisticated development IDE, a complete description will be out of the scope of this report, but the following example will show the basic functions and benefits.

**Figure 4.4** Main menu of the CodeWarrior Linux version

Under the tab “File” in the main menu showed in Figure 4.4 there is an option for creating a new project and also a new file.

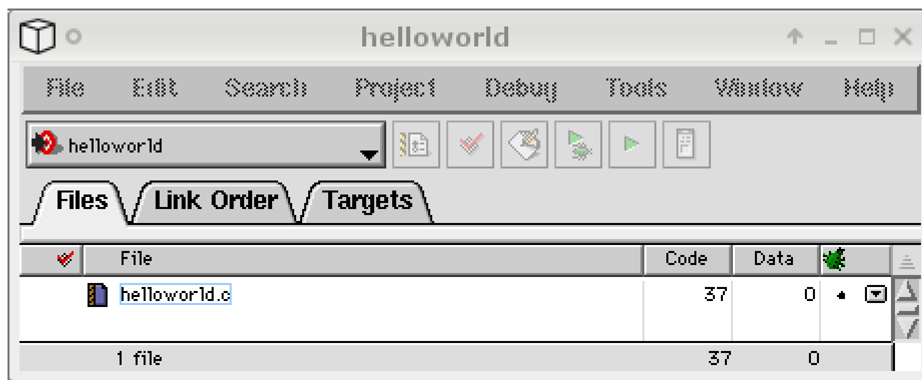


**Figure 4.5** Main window for the project called “helloworld”

To add the new file into the project as in Figure 4.5 it is first necessary to make sure a linker is selected in the target settings (see Figure 4.6) which can be found under the Edit tab and “<project_name> settings”.

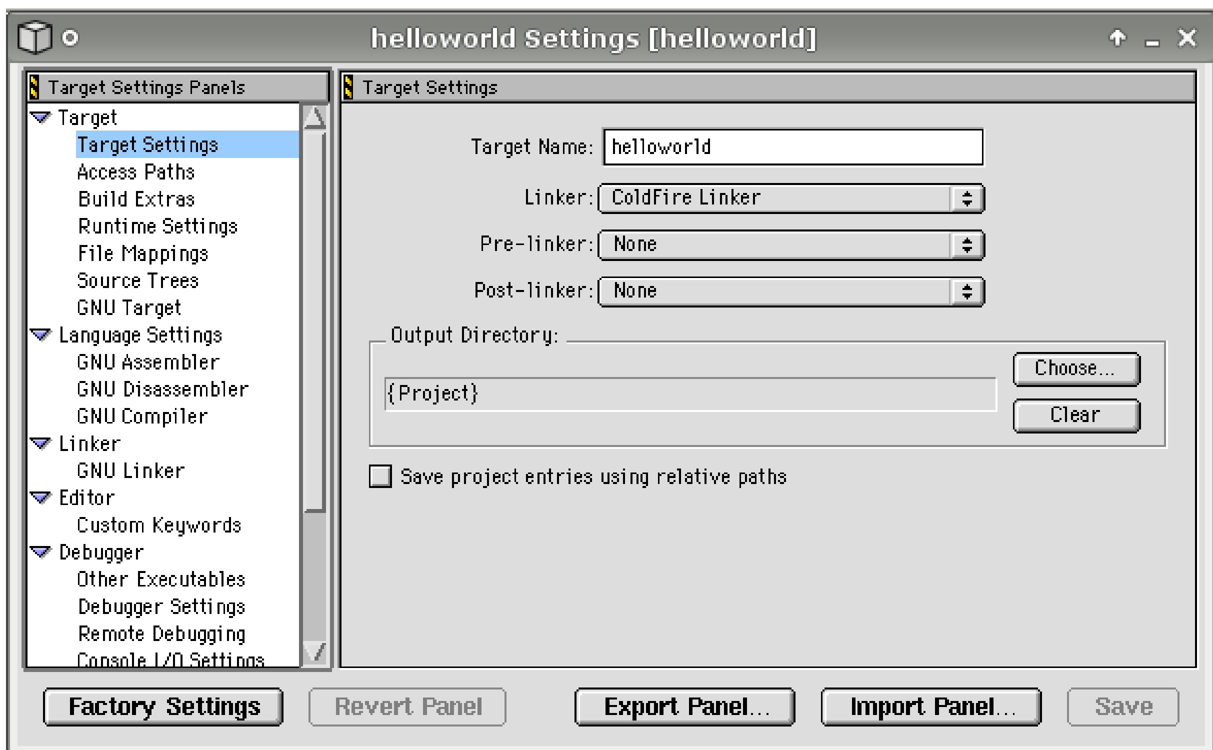


**Figure 4.6** Configuration window for the current project

Central things to set this early in the project could look something like Table 4.3.

| Category | Option | Recommended Setting |
|---|---|---|
| **Target Settings** | Linker | ColdFire Linker |
| **GNU Target** | Output File name | Optional |
| **GNU Compiler** | Command Line Arguments | -Wall -m5307 |
| **GNU Linker** | Linker/Archiver Flags | -m5307 –elf2flt |
| **GNU Tools** | Use Custom Tool Commands | Select Tool chain to use |

**Table 4.3** Recommended settings for a CodeWarrior project

When all settings are saved and the code is finished, the next thing to do is compiling, run and hopefully end up with a working program. An example program source and output can be seen in Figure 4.7.

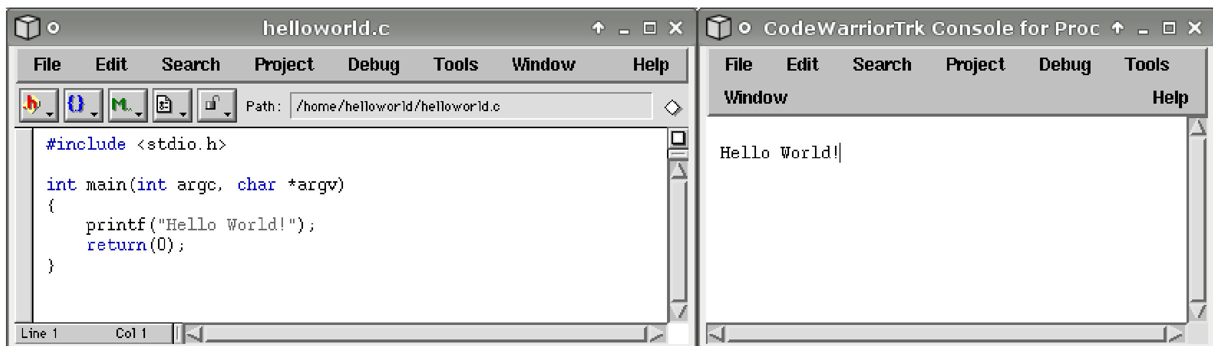


**Figure 4.7** Editor / Console window

### *4.8.1 Debugging with AppTRK*

Application development is not always as simple as it may seem when looking at the previous chapters and often one will end up with errors and warnings that needs to be sorted out. Freescale have equipped CodeWarrior with a tool to simplify debugging on uClinux based platforms. The debugging system used is called AppTRK. By downloading the AppTRK program to the target board, using NFS or similar it is possible to run the AppTRK program as a background process. This will give access to debug any uClinux application directly on target, much like *gdbserver*.

The following example expects the AppTRK to already be compiled and downloaded to the flash directory on target. In “<project_name> settings” there has to be some changes done under the Debugger tab as shown in Table 4.4.

| Debugger | Option | Recommended Setting |
|---|---|---|
| **Remote Debugging** | Connection | Edited as Figure 4.8 |
| | Remote download path | /tmp |
| **Console I/O Settings** | Stdout | Console I/O |

**Table 4.4** Recommended settings for the debugger

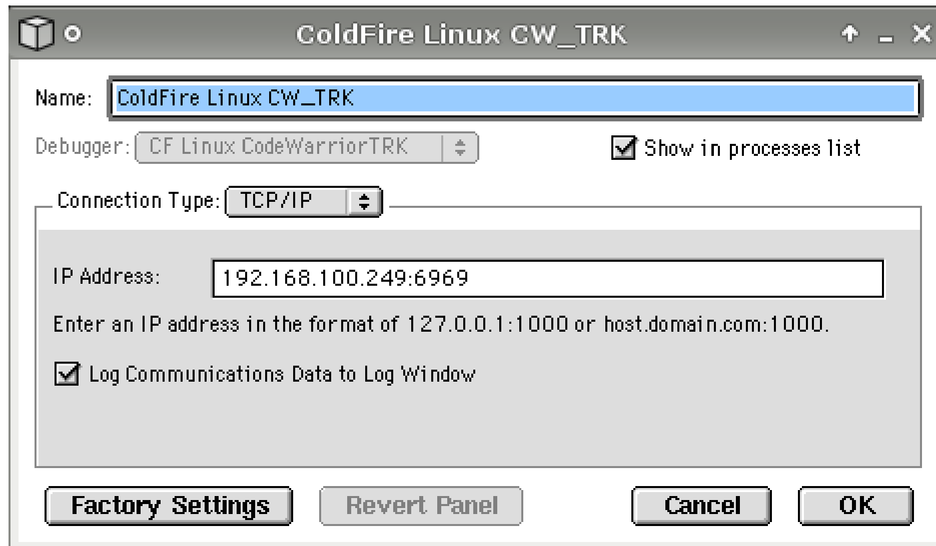


**Figure 4.8** Connection settings to target

In the target terminal the following command line is used to run the AppTRK application as a background process and with port number in this case set to be 6969.

```
/mnt/flash> ./AppTRK :6969 &
[34]
```

**Code 4.15** Command line to run AppTRK as a background process on port 6969

When running in debug mode, a new window will appear giving a huge amount of helpful options. A simple “Hello World”-example, which can be seen in Figure 4.9, is only scratching the surface of the possibilities offered when using AppTRK.

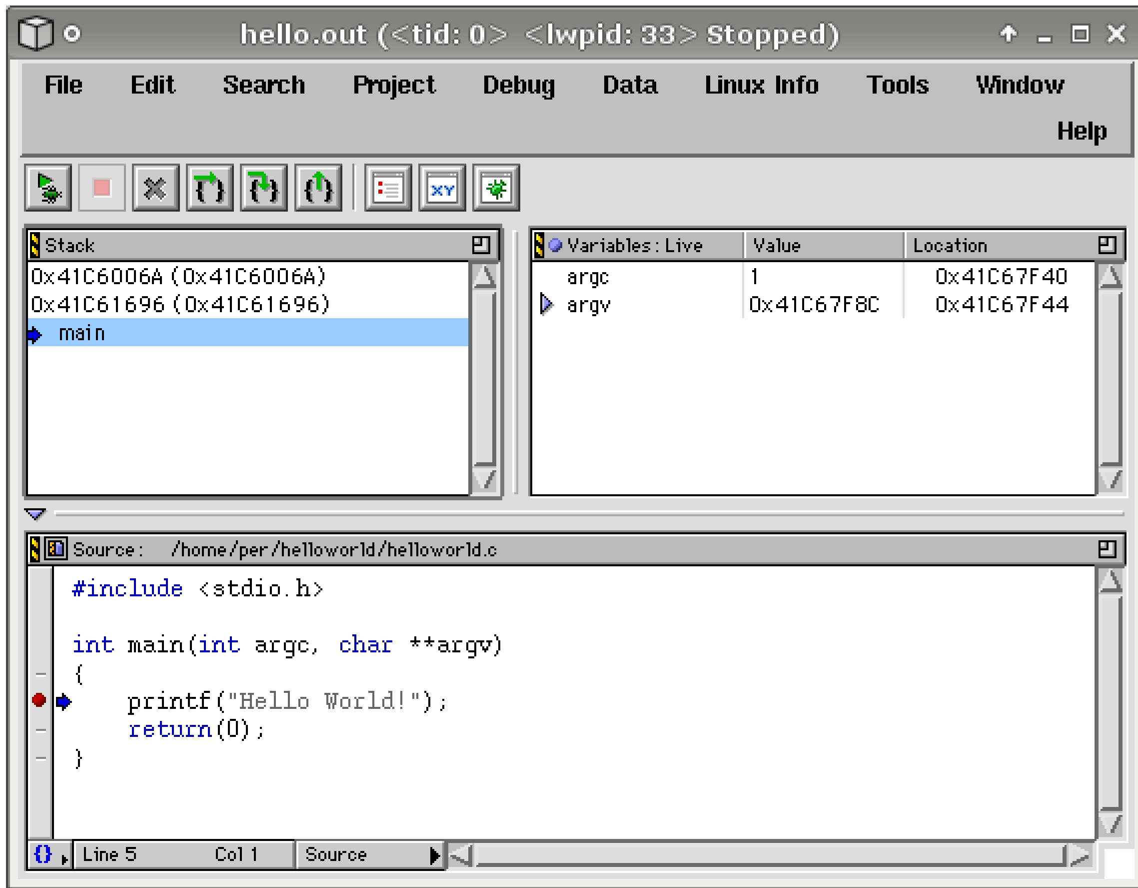


**Figure 4.9** Debugging window in CodeWarrior showing a breakpoint in the code as well as various other information relating to program execution

With the debugger there is also a possibility to get the output printed in the target console which was chosen in Table 4.3 and the result is given in Code 4.16.

```
/mnt/flash> Connected on :6969
Hello World!
Disconnected
```

**Code 4.16** Example showing console I/O output of the “Helloworld” program

# 5. Customizing the uClinux image

Each uClinux distribution contains a number of things. First off you have the kernel which is the core of the operating system. The kernel itself has many different settings relating to what type of hardware you are running it on. For example it is possible to decide what type of flash device is used, if you wish to have Ethernet support and in that case what type of Ethernet device you are using, if you wish to have USB/SPI/I2C support and much more. In addition to drivers for various hardware platforms there are also settings that affect how the kernel behave and operate. Of course, all these settings are different depending on what version of the kernel you choose. In short the currently most common kernel versions are the 2.4 and 2.6 branches.

Also included in the distribution are a number of different user applications that can be enabled or disabled based on what type of functionality you would like to have in the final image. These include a choice of shell, various system utilities as well as daemons such as web server, ftp server etc.

Once all the choices have been made, what you get is a single image file which can easily be downloaded using e.g. TFTP or similar method to the target board.

## 5.1 Kernel versions

### *5.1.1 2.4.x*

The kernel development began around the 20$^{th}$ century just slightly after the original Linux 2.4 kernel. It was a continuation of uClinux 2.0.3x since the 2.2 Linux kernel where never really “rewritten” into uClinux. The 2.4 kernel had numerous of additional features like USB support, IrDA, and QoS (quality of service) support to mention some. As the version status is said to be final at 2.4.32 up until now that’s the kernel most widely used and popular, due to stability. The version upgrades that is done to the 2.4.3x from now is only reported fatal bugs and similar.

### *5.1.2 2.6.x*

At the time of this writing 2.6.19 is the latest kernel and the one tested during this project work, but it was not elected for reasons handled in section 7.1.

Additional processor support, improved security, upgrading networking and support for hardware acceleration are some of the things added to the 2.6.x kernel in comparison with the 2.4.x. This kernel is still under development and evaluation, why it will continuously be improved.

## 5.2 Distributions

A distribution contains kernel, vendor tree, libraries and application code. The latest stable package distributed when this report is written also includes all three latest kernel versions 2.0.39, 2.4.32 and 2.6.19, together with both the older uC-libc and newer uClibc-0.9.27 libraries. The packages are also able to support various CPU architectures like M68k, ColdFire, ARM, Sparc, Altera NIOS, Xilinx Microblaze to mention some. As most parts in the distribution is under continuous development there is also beta

distributions released, which is not recommended to be used in production systems. One recommended way to get a distribution up to date is to visit the website http://www.uclinux.org/pub/uClinux/dist/. When the distribution is in hand, next thing to do is to get a tool chain used for building the uClinux image.

## 5.3 Build example

The first step is to install a suitable tool chain depending on which kernel one intends to build. The distribution package is downloaded and unpacked using the following command line:

```
tar xzf uClinux-dist-xxxx.tar.gz
```

**Code 5.1** Command line to unpack the distribution on host system

This will extract the source into an uClinux-dist directory. Code 5.2 shows how to add the tool chain path of the system PATH environment variable.

```
cd uClinux-dist
export PATH="$PATH:/(search path to the compiler of choice)"
```

**Code 5.2** Commands used for enter the distribution directory and add the compiler path

When this is done a configuration and build can be done as follows:

```
make clean          (clean out the source tree)

make menuconfig     (kernel configuration alternatives)
make config
make xconfig

make dep            (builds the dependencies)
make                (trying to build kernel image)
```

**Code 5.3** Commands used when building an image file on host PC

The third command *make dep* is not needed if the choice of kernel is 2.6.x. If the building process is done properly (can take a while depending on machine, 10-30 minutes) the finished image can be found in the subdirectory called images.

## 5.4 Downloading image to target

There are possibly several ways to handle this step, but the one used in this case was tftp which was enabled by assuring that the proper configuration were set on the host machine:

```
/etc/hosts.allow:

all : all

/etc/conf.d/in.tftpd:

INTFTPD_PATH="/var/tftp"
INTFTPD_OPTS="-R 4096:32767 -s ${INTFTPD_PATH}"
```

**Code 5.4** Configuration done on host PC to enable tftp. This particular configuration is done on a Gentoo Linux distribution

Next step would be to copy the imagez.bin file into the shared directory */var/tftp/* using command *cp*. The kernel image will be available over the network and by using dBUG on the target board (see section 3.6) it is a straight forward task to download the new image as can be seen below.

```
dBUG> set server <host IP>      (setting correct IP to host)
dBUG> dnfl imagez.bin           (downloading image to flash)
.                               (processing)
.
.
dBUG> gfl                       (used when finished to start the system)
```

**Code 5.5** User commands in the target board console for downloading image

## 5.5 Debugging the kernel

During the project some debugging on the uClinux 2.4 kernel was done using a BDM pod and *gdb*. This effort was taken when trying to track down and gain information about the issue discussed in section 7.2, the MAP_SHARED kernel message. There is also a possibility to use CodeWarrior for kernel debugging purposes but this way of debugging has not been tested during this project.

For more detailed information about how to do debugging with the BDM pod see Appendix A as well as the article about *gdb* which is available at:

http://www.ucdot.org/article.pl?sid=03/01/30/0548223

# 6. Web interface

The thesis involved creating a web interface containing all the information generated by the application. To do this a number of different techniques of presenting the data were examined. First of all a web server was needed and Boa appeared to suit the needs. It's one of the web servers available in the official uClinux distribution and as such is readily available to implement. Secondly a technique for presenting the data had to be selected. Basically there were three options available:

- Generating complete HTML through the application
- Create a CGI application to parse and present data
- Use the extensive capabilities of XML files which are generated by the application and then transformed using XSLT

The first option was quickly dismissed as it would require that too much formatting and layout information had to be hardcoded into the application. Preferably the data had to be completely separate from any layout or formatting. The next technique was to use CGI to parse a data file generated by the application and then present it to the user. CGI itself presents certain security issues however as it is basically an application started by the web server and any application have the potential to crash the system. In the end XML and XSLT was the technique that was deemed as the most viable given that it would be the most flexible way to add data to the website. It would also completely separate any formatting and layout from the data as well as not requiring having an application launch every time someone accessed the website (except for Boa itself).

## 6.1 Boa web server

Boa is an open source HTTP web server, working with single-tasking and internal multiplexing to handle multiple connections. It has support for common gateway interface (CGI) that allows web servers to run programs (which must be separate processes), automatic directory generation, and automatic file gunzipping (which involves unzipping files on the fly). Boa was created in 1991 with the primary design goals to advocate speed and security.

Boa is enabled in the configuration which was used in this project, shown in Figure 6.1.

```
                                 Network Applications
> selects submenus --->.  Highlighted letters a
c> to exit, <?> for Help, </> for Search.  Lege

-^(-)-----------------------------------------
 [ ] fnord web server
 [ ] fnord uses PAM for auth
 [*] boa
 [ ]   boa uses SSL
 [ ]   emergency syslog
 [ ]   enable log files
 [ ] bpalogin
```

**Figure 6.1** Screen dump showing boa enabled using menuconfig

Before building the uClinux image it can be good having a look into boa.conf which can be found at */... /uClinux-dist/vendors/Generic/httpd/* where boa can be configured. For this project the lines marked in the left column in Code 6.1 were changed or added to meet functionalities needed.

```
  #
  # A minimal config that makes the home page
  # an unauthenticated CGI
  #
  ServerName uClinux
* DocumentRoot /var/httpd
* Alias /img /var/httpd/img
* #ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
* #Auth /cgi-bin/cgi_demo /etc/config/config
+ DirectoryIndex index.xml
  AddType text/plain   txt
  AddType image/gif    gif
  AddType text/html    html
  AddType text/html    htm
  AddType text/xml     xml
+ AddType text/xsl     xsl
+ AddType text/css     css
  AddType image/jpeg   jpe
  AddType image/jpeg   jpeg
  AddType image/jpeg   jpg
  AddType image/x-icon ico
  AddType image/svg+xml   svg
```

**Code 6.1** Changed boa.conf, asterisk marks modified lines and plus marks added lines

## 6.2 Dynamic web pages

A website can be done in many different ways, the most simple being plain HTML documents. Such documents are static in nature and can only be changed by editing the files directly. Today the web is more interactive, more dynamic, and as such websites which have some form of dynamic content generation is much more common than the old ways of having static HTML documents. Generating content in such websites can be done in many different ways but can be divided into two categories: server-side processing and client-side processing.

There are many techniques that can be used for server-side content generation. The most common techniques are programming languages such as PHP, ASP and various CGI languages. All of them offer similar capabilities but is carried out in different ways. PHP and ASP both require separate software which work alongside the web server to process the code. CGI on the other hand are simply executable files which are executed by the server and generate content based on a specification. Which one of these techniques are used depend mostly on personal preference but also on the capabilities of the platform and web server it's run on. Embedded systems rarely have the capabilities of using one of the more powerful techniques such as PHP or ASP but instead rely on CGI.

Client-side content generation is often combined with some sort of server-side technique. Examples of client-side content generation can be seen with JavaScript applications or perhaps more commonly used; Flash. Even Java is sometimes used, although more for very specific applications rather than whole websites.

### *6.2.1 CGI*

Common Gateway Interface (CGI) was agreed on by the WC3 organization in 1995. CGI is a protocol that is used for interfacing external applications with an information server such as web-servers. As the name implies it's not a programming language, but just a way to execute programs and scripts in real-time that dynamically produces information for the web browser to process. The method is commonly used to process databases onto the web, but almost any program can be translated. There are eventually two important things to take into consideration when using CGI, which is the time and workload effort together with security aspects. The first issue demands inquiry of intended amount of traffic and complexity of the transaction with a trade-off for the functionality. There are some alternatives available, such as FastCGI and developed third-party tools which will not be described in-depth here. The security has to be handled by the web server administrator and is case dependent.

As a simple explanation of the procedure, say a user selection will give an input URL to the web-server. That will execute the corresponding program giving an output back to the server, augmented with the proper CGI standards and handled back to the user. (12) (13)

### *6.2.2 XML and XSLT*

XML and XSLT are markup languages which have been derived from the ancestor of all markup languages "Standard Generalized Markup Language" (SGML). To start with, the XML (eXtensible Markup Language) were founded by World Wide Web Consortium (W3C) to transfer data over the web but soon gained popularity among other user groups such as economical businesses to mention one of many. The strength of XML is simplicity of transferring information to suit your needs and to various types of devices. XML documents are used to categorize and organize information under user defined tags of individual choice, but with a strict specification of rules to follow. When you got an XML document that complies with the specifications there are several ways to transform the information into e.g. a graphical view using style-sheets or synthetic speech for vision-impaired people (as an example).

The following properties are all met by the XML standard and give a good summary of the benefits:

- **Extensible:** It can be tailored or customized for specific applications
- **Open:** It's well documented and widely available
- **Non-proprietary:** It uses standard form of notations that's free for all and widely used

```
<?xml version="1.0" ?>

<persons>
   <person username="JS1">
      <name>John</name>
      <family-name>Smith</family-name>
   </person>
   <person username="MI1">
      <name>Morka</name>
      <family-name>Ismincius</family-name>
   </person>
</persons>
```

**Code 6.2** Code example showing a simple XML document structure. In this case, <persons> is the root element which have two children which in turn have two children. Each root child describes a person with a name and a family-name.

Code 6.2 covers the XML-document well. Everything needed to create a basic XML sheet is the declaration, root element and handling the opening and closing tags with great care according to uppercase sensitivity, opening/closing at the same level and in case of empty elements prefer to leave the stand alone closing tag and end with a slash "/".

As mentioned earlier the benefit with XML is the ability to transform in so many ways to best suit the use. In this case the usage were decided to be web based, why the XML document was transformed using XSLT in a combination with CSS style sheets to produce a webpage.

By adding XSLT style sheets it is possible to generate/transform markup, in this case XML into HTML. In combination with CSS style sheet that's enough to make rather complex web presentations for the web browser to know how to use the translated content. (14) (15)

### *6.2.3 Graphics using SVG*

SVG (Scalable Vector Graphics) is a language for describing two-dimensional graphics and graphical applications in XML. As the name implies, SVG describes vector graphics. Vector graphics is different compared to the more commonly used raster graphics such as that which can be seen with e.g. JPEG-, GIF- or PNG-formats, as can be seen in Figure 6.2. The idea behind vector graphics is that each object is described in a way that no matter how much you zoom, the object will always be reconstructed perfectly without any of the artefacts, such as jagged edges, usually seen when zooming raster images. (16)

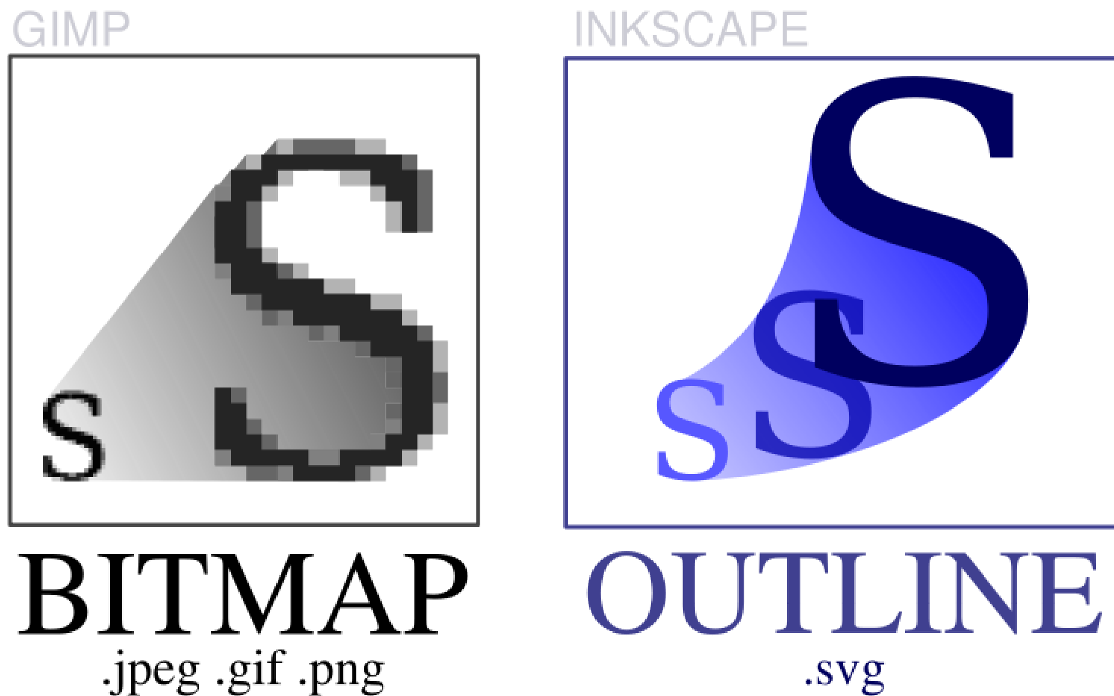


**Figure 6.2** A figure displaying the differences between raster graphics (left) and vector graphics (right).

The layout of an SVG file is an XML document.

```
<?xml version="1.0" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
   width="467" height="462">

  <!-- This is for the red square -->
  <rect x="80" y="60" width="250" height="250" rx="20" fill="red"
     stroke="black" stroke-width="2px" />
  <!-- This is for the blue square -->
  <rect x="140" y="120" width="250" height="250" rx="40" fill="blue"
     fill-opacity="0.7" stroke="black" stroke-width="2px" />

</svg>
```

**Code 6.3** Code example showing the XML behind a very simple SVG image. The two *rect* elements each draw a square, one with red and one with blue fill.

The result of the SVG image described in Code 6.3 can be seen below. The different attributes set on the two *rect* objects describe how each square should look as well as positioning it in the coordinate system set up by the root element. In this case we see that the first square has a width and height of 250 pixels with 20 pixels radius rounded corners, has a fill colour of red and a 2 pixel wide black stroke line around it. The second square has the same size and stroke properties, but is positioned slightly more to the right and slightly below compared to the red square. Its fill colour is blue, the rounded corners are 40 pixels radius and there's an additional new property here which sets the opacity of the square to 70%, hence why the blue square is slightly transparent in the image below. (17)

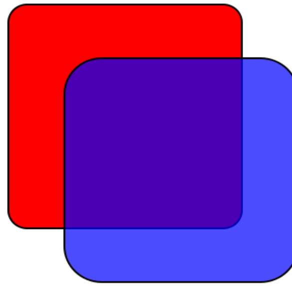

**Figure 6.3** Figure showing the end result of the XML-based SVG image.

SVG itself is a very capable graphics format, but since it is based on XML it can also be combined with most of the available XML technologies. One such example is the ability to generate dynamic graphs such as bar charts or line graphs. To do that one can use various number of server-side languages, such as PHP, ASP or even Java, to generate the SVG file, but to really take advantage of the fact that it is XML based one can use XSL transformations. With an XML file coupled with a XSL style sheet it is possible to transform the XML data into a SVG-based graph. The transformation is not very difficult to accomplish and the result can be amazing. (18) (19)

## 6.3 Practical example

To showcase some of the capabilities of using XML and XSL transformations an example is in order:

Imagine you have a measuring application running on a target board. It measures a number of temperatures at a regular interval and these temperatures need to be presented on a website. For this you need to add a means to output an XML data file from the application itself. This can be done pretty easily by using simple file I/O operations as can be seen below.

```
/***************************************************************
* FUNCTION: writeXML
* INPUT: iTemp1, iTemp2, iTemp3; Temperatures in degrees
*         celcius
* OUTUT: None
* NOTES: Requires header-file stdio.h
***************************************************************/
void writeXML(int iTemp1, int iTemp2, int iTemp3)
{
   FILE* fp;

   fp = fopen("temps.xml", "w");
   if(fp == NULL)
      return; // Unable to open file

   // Write XML header and stylesheet definition
   fprintf(fp, "<?xml version=\"1.0\" ?>\n");
   fprintf(fp, "<?xml-stylesheet type=\"text/xsl\" href=\"style.xsl\"
?>\n");

   // Write root element and each of the three temperatures
   fprintf(fp, "<temps>\n");
   fprintf(fp, "\t<temp>%d</temp>\n", iTemp1);
```

```
    fprintf(fp, "\t<temp>%d</temp>\n", iTemp2);
    fprintf(fp, "\t<temp>%d</temp>\n", iTemp3);
    fprintf(fp, "</temps>\n");

    // Flush buffer and close file
    fflush(fp);
    fclose(fp);
}
```

**Code 6.4** Function writeXML which is used to write three temperatures to an XML file

The function shown in Code 6.4 generates an XML file with the XML header and style sheet instructions followed by a root element with three children, each containing a temperature reading. The result of this function can be seen below.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>

<temps>
   <temp>30</temp>
   <temp>25</temp>
   <temp>-5</temp>
</temps>
```

**Code 6.5** Generated XML file

Now that we have an XML file with the necessary data we can start with the accompanying XSL transformation file. Say that we want all temperatures to be in a table with a header for each temperature reading. The necessary transformations can be seen in Code 6.6.

```
<?xml version="1.0" ?>

<xsl:stylesheet
 version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/1999/xhtml">

<xsl:output method="html" />

<!-- Root template -->
<xsl:template match="/temps">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Temperature Readings</title>
</head>
<body>
   <h1>Temperature readings</h1>
   <table>
      <tr>
         <th>Temp 1</th>
         <th>Temp 2</th>
         <th>Temp 3</th>
      </tr>
      <tr>
         <xsl:for-each select="temp">
         <td><xsl:value-of select="." /></td>
         </xsl:for-each>
      </tr>
```

```
    </table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

**Code 6.6** XSLT style sheet used to transform an XML file

Combining the XML data file with the XSLT file you get the result as seen in Figure 6.4. This can with more elaborate HTML and combined with CSS styling be made to look anyway you want.

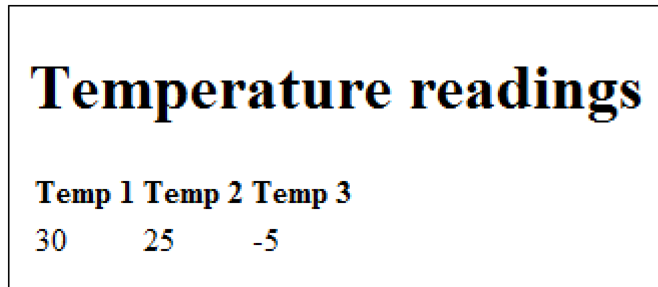


**Figure 6.4** Result of transforming the XML file

# 7. Issues encountered

Linux, although extremely powerful and flexible, does have its quirks. For a beginner it can be seen as a daunting task to learn to operate. It is a very complex operating system and before you understand how to get around in Linux it is very possible to make some disastrous mistakes. Someone once said *"I love Linux because it lets you shoot yourself in the foot. There is nothing like pain because it makes you learn."* and that is very true. The surest way to learn your way around Linux is by making mistakes.

When you finally do learn it however, you have a very powerful operating system at your fingertips. People who have been around Linux for a while and especially the open source community have gotten to know the hard way that if you happen to find a problem and no one else have encountered it yet, you probably have to fix it yourself. There's a famous expression: Open source = you own the problem. There are many discussion forums and mailing lists where people can ask for advice, but in the end you are in a way expected to find the solution to the problem yourself. When doing a project involving Linux this is a big risk that has to be taken into consideration. There are usually no 3rd party vendors where you can get a 24/7 support agreements without paying huge fees.

## 7.1 Issues kernel 2.6.x

An issue has been observed in the 20070130 distribution when used with the 5208 platform or any other ColdFire platform as well. The problem was observed when a JFFS2 partition was written to. Occasionally the following error message was printed on the console:

```
Data CRC 3f1aceac != calculated CRC 91126571 for node at 0014551c
```

**Code 7.1** Error message printed on console when accessing JFFS2 partition

The source for the problem could be narrowed down to the case where it was observed that a spurious read to the FLASH memory was done, in the middle of the busy poll of dq6 after a word write/buffer write. This toggle dq6 in the FLASH and fooled the code checking for a completed operating to exit prematurely. Disabling the interrupts in the busy poll parts in cfi_cmdset_0002.c made the error messages disappear. This was however not a solution, just an observation.

Measurements on the CS signal on the FLASH memory showed that the spurious access was made randomly when accessing files in the ROMfs (stored in SDRAM). Just by running for instance *ifconfig* caused the CS to show activity a number of times. When doing the same measurements on the 2.4.27 kernel, the FLASH CS was always high (as expected).

A semi-dirty way of generating a break when the "illegal" access to the FLASH was made was to hook the FLASH CS signal to the BKPT pin on the BDM-port on the board. With a simple 2-way switch and a couple of resistors everything could be booted and gdb started (using the USB Multilink pod) and then flip the switch to route the FLASH CS signal to BKPT instead of letting the BDM pod control BKPT. Executing something in */bin* caused a break to be triggered as can be seen below.

```
Program received signal SIGTRAP, Trace/breakpoint trap. 0x40022e9a in
timer_interrupt (irq=68, dummy=0x0, regs=0xe4a0) at
arch/m68knommu/kernel/time.c:54
54              update_process_times(user_mode(regs));
(gdb)
```

**Code 7.2** A break being triggered while debugging with gdb

The source of the problem was caused by the *regs* pointer not being given a valid value in the timer interrupt function in *arch/m68knommu/kernel/time.c*, due to the general change of how the *regs* pointer is handled in the later kernel versions. A dereference of the *regs* pointer was made, with *regs* uninitialized which coincided with the FLASH address range, causing the spurious FLASH access. The *regs* pointer bug was fixed in a patch from Greg Ungerer on February 2007-02-09. When applying this patch, everything works perfectly. No more "spurious" FLASH accesses causing problems with the dq6 polling. JFFS2 works without any problems.

For more information about the process of tracking down and fixing this bug, see the following posts from the uClinux-dev mailing list:

http://mailman.uclinux.org/pipermail/uclinux-dev/2007-August/043886.html
http://mailman.uclinux.org/pipermail/uclinux-dev/2007-November/044695.html
http://mailman.uclinux.org/pipermail/uclinux-dev/2007-December/044973.html
http://mailman.uclinux.org/pipermail/uclinux-dev/2008-January/045076.html

When the uClinux distribution was patched to 20080305 another problem occurred when using JFFS2. During file system idling error messages like *"Newly-erased block contained word 0xXXXXXX at offset 0xYYYYYYYY"* appeared at random. This could occur at any time and the nature of the error is severe, basically meaning that an erase of a block did not succeed successfully. It was speculated that the error message originated from the JFFS2 garbage collector which runs while there is no or low activity on the file system. A solution to this issue was never found.

## 7.2 Issues kernel 2.4.x

During boot of the 2.4 kernel a disturbing error message was posted in the beginning of the boot process stating *"BUG: wrong zone alignment, it will crash"*. The origin of this error message was from PAGE_OFFSET_RAW which is defined in *linux-2.4.x/include/asm-arch/page_offset.h*. For the M5208EVB board this was set to 0x40020000. This value needs to be alignment to a step of a power of 2 such that $2^{(PAGE_SHIFT+MAX_ORDER-1)}$ which in this case was not true. A simple change of the value to 0x40000000 put it in line with the alignment requirements and the error message disappeared.

Another error message that appeared especially during heavy FTP activity was *"MAP_SHARED not completely supported on !MMU"*. The origin of this error message was traced to *linux-2.4.x/mmnommu/mmap.c* on line 1452 and was generated because of a function call to *mmap* in the ftpd application. Usage of *mmap* is toggled using the #define HAVE_MMAP in *user/ftpd/config.h*. A question about this was sent to the uClinux-dev mailing list and the response from Jamie Lokie can be seen below.

```
Using mmap is just an optimisation - it can save data copying time and
RAM when serving files.  (sendfile is even better than mmap, but I think
that's not supported in 2.4 nommu).

This message means mmap is refused, and presumably ftpd falls back to
read/write.  That's completely safe.

So if you get this message every time, disabling mmap in ftpd makes sense
- it's pointless trying to mmap.

If you don't get it for some files, it may be worth keeping the mmap
optimisation in ftpd and commenting out the printk instead (keep the
return -EINVAL).  (Though, it might not really improve anything - mmap
'optimisations' are not always an improvement).
```

**Code 7.3** Reply from Jamie Lokie about the MAP_SHARED issue found in the 2.4 kernel

## 7.3 Issues in user land

With the 20080305 distribution a few issues were encountered in user land. Another issue in the ftpd application was a bug which appeared while transferring many small files rapidly. It would return odd socket errors and cause the ftpd application to crash. This bug was quickly fixed and a patch for it can be found in Code 7.4.

```
--- uClinux-dist-20080305/user/ftpd/ftpd.c   2004-08-06
08:32:16.000000000 +0200
+++ uClinux-dist/user/ftpd/ftpd.c  2008-04-21 11:50:56.000000000 +0200
@@ -198,7 +198,7 @@
   static char bufs[NUM_SIMUL_OFF_TO_STRS][80];
   static char (*next_buf)[80] = bufs;

-  if (next_buf > (bufs+NUM_SIMUL_OFF_TO_STRS))
+  if (next_buf >= (bufs+NUM_SIMUL_OFF_TO_STRS))
     next_buf = bufs;

   if (sizeof (off) > sizeof (long))
```

**Code 7.4** Patch for ftpd to correct a pointer issue

In addition, the busybox have a version of the *mount* command and in the source was added a function to be able to daemonize the mount operation to perform a mount in the background. This operation relied on the *fork()* system call which, as previously discussed, is not available on non-MMU based systems. It was not a simple task to replace *fork()* with *vfork()* so instead a workaround was devised where a separate empty function was defined if it was compiled without MMU support. This new function simply returns 0, in effect disabling the functionality of the daemonize feature.

```
--- uClinux-dist-20080305/user/busybox/util-linux/mount.c   2008-03-25
14:59:50.000000000 +0100
+++ uClinux-dist/user/busybox/util-linux/mount.c  2008-03-25
14:27:19.000000000 +0100
@@ -726,6 +726,7 @@
  return &p;
 }

+#if defined(__UCLIBC__) && defined(__UCLIBC_HAS_MMU)
 static int daemonize(void)
 {
```

```
  int fd;
@@ -745,6 +746,12 @@
  logmode = LOGMODE_SYSLOG;
  return 1;
 }
+#else
+static int daemonize(void)
+{
+ return 0;
+}
+#endif

 // TODO
 static inline int we_saw_this_host_before(const char *hostname)
```

**Code 7.5** Patch for the busybox mount command to support non-MMU architectures

Debugging is an important tool to be able to use. Previously mentioned is the AppTRK application which is distributed alongside of CodeWarrior. Attempts was made to use AppTRK with the 4.1.1 tool chain, but there was an issue with the application debug information generated by that tool chain which caused the application to fall through the debugging traps. In effect this meant that debugging with AppTRK using the 4.1.1 tool chain was not possible.

An issue that was encountered late in the project was that the CodeSourcery tool chain did not take into account certain environment variables during compilation of the uClinux distribution package. An important environment variable called FLTFLAGS is set for various user applications to tell the compiler what header flags to use for the flat binary format. A flag that was commonly set using that environment variable was the stack size flag which is used to increase or decrease the stack size for the given application from the default value of 4096 bytes. This means that all applications in the uClinux distribution were using the default stack size when compiled using the CodeSourcery tool chain which is a very bad thing.

# 8. Conclusions

In the weeks we have been working on this project there have been many difficulties to get around as well as a lot of different things to take into consideration. It has definitely not been a straightforward process but instead been a path filled with obstacles. Most of them have been solved but some of them have been too technical and time consuming that we simply haven't had the time to investigate. There has been a lot of reading and learning in order to successfully complete the project. Things like learning about the workings of uClinux in order to make the best out of the resources at hand or learning about the different technologies available for web based presentation.

The biggest challenge in the project has probably been getting the uClinux kernel and user applications up and running without issues. As can be seen above, there were a number of different problems that we had to face and that's what took the majority of the time. Developing the actual web based interface took only a fraction of the total time. The end result however is something that we're proud of and it is working very well.

## References

1. **Intec Automation Inc.** M5208EVB-RevB: 32-bit Microcontroller User Manual. *Freescale Semiconductor.* [Online] 7 September 2005. [Cited: 8 May 2008.] http://www.freescale.com/files/32bit/doc/ref_manual/M5208EVBUM.pdf.

2. **Hallinan, Christopher.** *Embedded Linux Primer: A Practical Real-world Approach.* s.l. : Prentice Hall, 2006. ISBN: 978-0-13-167984-9.

3. **Drabik, John.** uClinux: World's most popular embedded Linux distro? *linuxdevices.com.* [Online] 24 September 2002. [Cited: 12 May 2008.] http://www.linuxdevices.com/articles/AT3267251481.html.

4. **McCullough, David.** uClinux for Linux Programmers. *Linux Journal.* [Online] 1 July 2004. [Cited: 14 May 2008.] http://www.linuxjournal.com/article/7221.

5. romfs information. *SourceForge.* [Online] 26 June 2007. [Cited: 12 May 2008.] http://romfs.sourceforge.net/.

6. **Landley, Rob.** *Linux Kernel 2.6 Documentation: ramfs, rootfs and initramfs.* 2005.

7. **Raghavan, Pichai, Lad, Amol and Neelakandan, Sriram.** *Embedded Linux System Design and Development.* s.l. : Auerbach Publications, 2006. ISBN: 978-0-8493-4058-1.

8. **Singh, Amit.** An Introduction to Virtualization. *kernelthread.com.* [Online] 9 April 2006. [Cited: 13 May 2008.] http://www.kernelthread.com/publications/virtualization/.

9. **McCullough, David.** Should I use uClibc or uC-libc. *uCdot: Embedded Linux Developer Forum.* [Online] 30 September 2002. [Cited: 14 May 2008.] http://www.ucdot.org/article.pl?sid=02/09/30/0523232.

10. **Andersen, Erik.** A C library for embedded Linux. *uClibc.* [Online] 3 January 2001. [Cited: 14 May 2008.] http://uclibc.org/about.html.

11. **Hertel, Chris, Samba Team and jCIFS Team.** Samba: An Introduction. *Samba.* [Online] 27 November 2001. [Cited: 13 May 2008.] http://www.samba.org.

12. **NCSA HTTPd Development Team.** The Common Gateway Interface. *NCSA HTTPd.* [Online] 1 January 1998. [Cited: 13 May 2008.] http://hoohoo.ncsa.uiuc.edu/cgi.

13. **Wikipedia.** Common Gateway Interface. *Wikipedia.* [Online] February 2008. [Citat: den 13 May 2008.] http://en.wikipedia.org/wiki/Common_Gateway_Interface.

14. **Shepherd, Devan.** *Teach Yourself XML in 21 Days.* 2nd edition. s.l. : SAMS, 2001. ISBN: 0-672-32093-2.

15. **Wikipedia.** XSL Transformations. *Wikipedia.* [Online] 2008. [Cited: 5 May 2008.] http://en.wikipedia.org/wiki/XSL_Transformations.

16. **Adobe Systems Incorporated.** Scalable Vector Graphics: SVG Zone. *Adobe.* [Online] 30 March 2007. [Cited: 5 May 2008.] http://www.adobe.com/svg/.

17. **Wikipedia.** Scalable Vector Graphics. *Wikipedia.* [Online] May 2008. [Cited: 5 May 2008.] http://en.wikipedia.org/wiki/Scalable_Vector_Graphics.

18. **Venn, Brian.** Render dynamic graphs in SVG. *IBM DeveloperWorks.* [Online] 29 October 2004. [Cited: 15 April 2008.] http://www-128.ibm.com/developerworks/xml/library/x-svggrph/.

19. **Surguy, Inigo.** Client-side image generation with SVG and XSLT. [Online] [Cited: 25 April 2008.] http://surguy.net/articles/client-side-svg.xml.

# Appendix A

## Written by Matt Waddel, Freescale

```
M52277EVB On-Board BDM Setup and Usage

Using the following procedure I was able to setup a host to communicate with the
on-board BDM.

Host Setup:
-----------

(1) Download the publically available Linux/Ethernet Device Drivers for P&E
    devices from:

    http://www.pemicro.com/support/download_processor.cfm

(2) Untar the device driver.  In this case the downloaded file was named
    pe_driver_ver_324_811.tar.gz and it untared to the pe_driver_ver_324_811
    directory.

(3) Build and install the driver as root.  (This version of the driver was written
    for kernels later than 2.6.17.)  I had to copy the utsrelease.h file into a
    directory that was accessible from my compiler.  After these changes the device
    driver installed.

            cd pe_driver_ver_324_811/
            sudo ./setup.sh
                        (lots of build messages)
                        ...
                        Installing the WinDriver kernel module
                        --------------------
                        mkdir -p /lib/modules/2.6.18.8-0.7-default/kernel/drivers/misc
                        cp LINUX.2.6.18.8-0.7-default.i386/windrvr6.ko
                                    /lib/modules/2.6.18.8-0.7-default/kernel/drivers/misc
                        ./wdreg windrvr6 no
                        Waiting 15 seconds for windriver's kernel module to load...
                        wdreg windrvr6 auto
                        If you want everyone to access the module :
                        chmod 666 /dev/windrvr6
                        ********* ********** ********** ******** *********
                        ********* ********** ********** ******** *********
                        ********* ********** ********** ******** *********
                        ********* P&E Install has been a success *********
                        ********* ********** ********** ******** *********
                        ********* ********** ********** ******** *********
                        ********* ********** ********** ******** *********

            lsmod
                        Module                  Size  Used by
                        windrvr6              160192  0
                        ...

(4) Connect the P&E-USB connector to the host.

(5) Make sure the m68knommu uClinux toolchain is in your PATH.  As of this document
    it's at:

            /opt/freescale/usr/local/gcc-4.2.47-uclibc-0.9.47/m68k-uclinux/bin/

(6) On the host determine the USB id.

            m68k-uclinux-sprite -i
                        CodeSourcery ColdFire Debug Sprite (Sourcery G++ Lite 4.2-47)
                        pe: [speed=<n:0-31>&memory-timeout=<n:0-99>] P&E Adaptor
                        pe://USBMultilink/PE6 - USB1 : USB-ML-CF REF :
                        Embedded ColdFire Debug (PE6)
```

```
                    ccs: [timeout=<n>&speed=<n>] CCS Adaptor
                    ccs://$Host:$Port/$Chain_position - CCS address
```

(7) The "pe://USBMultilink/PE6" is the important information you need from the previous step. You'll use it in the next step.

(8) Start the CodeSourcery debugger included with the toolchain.

```
        m68k-uclinux-gdb
                    GNU gdb (Sourcery G++ Lite 4.2-47) 6.6.50.20070821-cvs
                    Copyright (C) 2007 Free Software Foundation, Inc.
                    GDB is free software, covered by the GNU General Public License,
                    and you are welcome to change it and/or distribute copies of it
                    under certain conditions.
                    Type "show copying" to see the conditions.
                    There is absolutely no warranty for GDB.  Type "show warranty" for
                    details.
                    This GDB was configured as "--host=i686-pc-linux-gnu
                    --target=m68k-uclinux".
                    For bug reporting instructions, please see:
                    <https://support.codesourcery.com/GNUToolchain/>.
        (gdb) target remote | m68k-uclinux-sprite pe://USBMultilink/PE6
                    Remote debugging using | m68k-uclinux-sprite pe://USBMultilink/PE6
                    m68k-uclinux-sprite: Opening P&E USBMultilink port 1 (USB1 :
                    USB-ML-CF REF : Embedded ColdFire Debug (PE6))
                    m68k-uclinux-sprite: Missing config file; this may not work
                    m68k-uclinux-sprite: Target reset
                    0x00000000 in ?? ()
```

(9) Run the dBUG bootloader installed in flash. It should run to a dBUG> prompt.

```
        (gdb) set $pc=0x400
        (gdb) c
        Continuing.

        dBUG>
```

(10) If the code is binary you can convert it to an srec file with this command.

```
        m68k-uclinux-objcopy -O srec -I binary vmlinux.bin vmlinux.srec
        m68k-uclinux-objcopy -O srec -I binary image.bin image.srec

        cp vmlinux.srec /tftpboot/
        cp image.srec /tftpboot/
```

(11) Halt execution with a <cntl>c and download the kernel or kernel+romfs in srec format.

```
        <cntl>c
        Program received signal SIGINT, Interrupt.
        0x40000a60 in ?? ()
        (gdb) load /tftpboot/vmlinux.srec 0x40020000
        Loading section .sec1, size 0x105000 lma 0x40020000
        Start address 0x0, load size 1069056
        Transfer rate: 40 KB/sec, 7803 bytes/write.
        (gdb) c
```

(12) From this point you can run the recently downloaded code from the dBUG> prompt. (Ignore any error messages that are displayed on the dBUG screen after continuing, they don't seem to affect kernel operation.)

```
        dBUG> go 0x40020000
        Linux version 2.6.22-uc1-gb1f6286b (mattw@loa) (gcc version 4.2.1 (Sourcery
        G++ Lite 4.2-47)) #19 Tue Nov 6 15:01:25 MST 2007

        uClinux/COLDFIRE(m5227x)
        COLDFIRE port done by Greg Ungerer, gerg@snapgear.com
        Flat model support (C) 1998,1999 Kenneth Albanowski, D. Jeff Dionne
        Built 1 zonelists.  Total pages: 8128
        Kernel command line: rootfstype=romfs
        PID hash table entries: 128 (order: 7, 512 bytes)
```