Order Number: AN2005/D
Rev. 0, 7/2000

Application Note

Embedded ColdFire®–Taking Linux On-Board

Nigel Dick
Netcomm Applications Group

1.1 Introduction

Linux is probably one of the most talked about operating systems recently and has rapidly taken up a major amount of column space in many of the computing journals and periodicals. In the embedded arena, Linux is growing at an equally rapid rate. How is it the old saying goes? The only thing in life that is free is the air that we breathe – not true! Linux is free; it works and has all the benefits of being an open-source software suite.

In the past, existing operating systems have often caused developers great frustration having to "just live with system bugs". Linux gives developers the power to identify and easily solve bugs by writing their own software "patches" which are then added to the software library for the benefit of all Linux users. The main growth of Linux in the embedded marketplace is undoubtedly due to this open nature. This allows developers to quickly modify the Linux kernel code to suit their own particular applications, which can then be uploaded, into the common open-source software pool which can be used and accessed by others.

This application note looks at the historical development of Linux and the features that make it suitable for embedding onto a ColdFire® Processor core. This document also details setting up a Linux debugging environment on the ColdFire 5206e LITE development board using freeware software. Similar references for other ColdFire development boards are also given. Finally, an illustrative example on the MCF5206 Arnewsh development board shows the uClinux kernel running.

1.2 Linux—A Brief History

The Linux operating system gets its name from its inventor, Linus Torvalds. Becoming dissatisfied with the operating system that came with his new IBM 386 PC hardware, Linus wanted to run a UNIX operating system instead. UNIX proved an expensive operating system and so he set out with a team of programmers to develop a new version of UNIX. From this work, a new core operating system, or kernel, named Linux was created. In early spring of 1994, three years after development first started, the first public released version of Linux was made available. This was Linux V1.0.

At the very outset of the Linux project, it was decided to make the code distribution freely available. Linux had a number of features which were normally only seen on operating systems costing hundreds of dollars per user. Features such as multiple simultaneous users, a demand paged virtual memory system and the ability of multi-tasking, quickly generated interest from prospective developers.

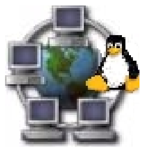
In 1997, three years after the release of Linux V1.0, an estimated 3,000,000 people were using Linux. The exact figures are difficult to determine since there are no licenses issued for individual users. By the end of the following year, 1998, this figured had doubled to well over 7,000,000. Today Linux is growing faster than any other operating system and accounts for 25% of the worldwide server market and over 50% of all web servers. See Section 1.7, “Data Sources and References” for more details.

1.3 Linux—A Versatile Operating System

The varied features of Linux allow it to be used in a variety of applications. In particular, Linux is ideally suited to networking environments, for the following reasons:

Since Linux is a multi-tasking operating system, it can handle the needs of more than one user at any one time and run many different programs at the same time. Linux can also handle programs and files of immense size. Features such as these make Linux perfectly adapted to the requirements of personal or networked workstations.

The virtual memory properties of Linux coupled with its multitasking capability and powerful file management system make it particularly suitable for file and print server applications. Since the operating system and layered software that controls the communication between the file system and printers attached to the Linux system is free, the solution is much cheaper. All versions of Linux support IP-masquerading. This allows a single Linux machine, or router, to control the routing of all network traffic from an entire network onto the Internet. Using this technique, an entire LAN can be securely hidden behind a single IP address, the configuration used most commonly by today’s business!



Linux is a popular choice with many Internet service providers because it is free and also because the source-code for the entire operating system is also freely available. Since this is the case, the Internet service provider (ISP) can quickly rectify problems themselves rather than wait for a commercial vendor to correct the problem. This gives the user the benefit of less network downtime. Indeed, Linux can be considered the backbone of the Internet as well over half of all web servers in the world use Linux because of its' stability and networking capabilities.

The kernel structure of Linux includes an integrated packet filter that allows the user to restrict incoming and outgoing IP traffic. This feature gives the kernel a high degree of security and also a high degree of immunity from harmful viruses. Thus, the kernel structure of Linux lends itself well to Internet firewall security applications.

The Linux operating system is a "friendly" operating system which means Linux can be easily installed alongside other operating systems on the same machine without causing any problems. It can be installed on a hard drive partition and using the Linux Boot Manager, LILO, the user can then select the appropriate operating system when booting up the system.

As systems grow in size and complexity, an operating system becomes essential to simplify the system software. An embedded Linux operating system has a number of key advantages over other available operating systems. Linux is a general-purpose operating system that provides a stable platform that is portable to many processors. The fact that the kernel can be considered completely configurable, or scalable, means that it can lend itself to a wide range of capabilities.

Linux supports the POSIX API (Portable Operating System Interface for Unix – Application Program Interface). This is a standard (IEEE 1003.1) that defines the language interface between application programs and the Unix operating system. Since Linux adheres to this standard it ensures compatibility when programs are moved from one Unix platform to another. The ability of uClinux to support the POSIX API means that many Unix /Linux applications can now be ported easily to embedded environments.

1.4 Linux—The Embedded System

Linux is considered a reliable operating system that is gaining considerable ground in the embedded field because of its small kernel. A key point to remember here is that the Linux operating system is just a core kernel, nothing more. In order to do something useful with the kernel and perform basic functions, additional elements, or applications, need to be added to the embedded Linux system. These applications can be anything from simple command line utilities to commercial applications. The Linux kernel, together with these applications is collectively called a distribution.

Since memory is usually limited in an embedded system, the size of the Linux kernel should be as small as possible. The two main methods of achieving this are to compress the kernel image and to customise the kernel. Customising the kernel allows the user to only configure the modules and device drivers that are needed for their particular application.

A basic embedded Linux system requires the following elements:

- A boot utility – Performing hardware initialisation and software start-up.
- Linux micro-kernel – Composed of timing services, process and memory management.
- Application set - Interrupt processing and environment setup.

Enhancing the functionality of the Linux system requires additional resources:

- Hardware drivers – serial console port driver, LAN drivers, customised drivers etc.
- One or more application processes – to provide the required system functionality.

As the number of applications are added, the following features might then be required:

- TCP/IP network stack – to handle network accesses.
- A file system – RAM or ROM (FLASH) based.
- Storage disk – for storing semi transient data and during data swap conditions.

The target device is often limited in terms of available memory in which the kernel and applications can be stored. Downsizing the Linux kernel to create a micro-kernel is often the most difficult part in porting the Linux core onto a processor core. One such port for the ColdFire Microprocessor, uCLinux, is a subset of the Linux core and not the entire operating system. (Removing unnecessary parts of the Linux core in an effort to minimise kernel size is not for the faint hearted and care must be exercised not to induce fatal bugs in the operating system).

Although target memory is of prime concern, Linux has a number of features that an embedded system can exploit to save RAM. Consider a typical embedded Linux application that consists of the kernel and some form of application program. In this simple example, both the kernel and application tasks should be in memory when the system first boots up. Both tasks and kernel can be compiled and then downloaded as one image to the target at boot time. This could be stored as a file system image in ROM or RAM.

Linux has the ability to save memory by being able to move programs to and from memory. Consider the situation when initialisation code is downloaded to the target. More often than not, this initialisation code is executed once to setup the target and is never run again until the target is reset. Hence, after the system has successfully booted, the initialisation code can be discarded or overwritten. Similarly, utility programs that run outside the kernel can be loaded when needed and then discarded (or unloaded) to save memory. In this way, the same area of memory could theoretically be used to load in utility programs, as and when the kernel requires them.

Memory can also be temporarily made available using a term called memory “swapping”. The term swapping is generally defined as when all or part of a running application is temporarily saved to a mass storage device such as a hard disk. The uClinux kernel does not support memory swapping.

The Linux kernel relies on the host processor having some form of memory management system to support virtual memory. This feature allows developers to write code without too much concern for the size of the program. Using Linux, the program can simply flow into the memory swap area of the disk or storage medium. The main hurdle to overcome during the development of an embedded kernel such as uCLinux, is the lack of an on-board memory management system. Non-embedded kernel operation would assume that virtual memory is available for the taking. The easiest way to workaround this problem for embedded kernels is to ensure that it only works with physical memory addresses rather than virtual ones. (Since virtual memory uses the swap area of a disk, setting the swap size of this area to zero enforces physical memory addressing).

The choice of storage medium limits what code should be located where. Using Flash memory (for memory swapping), which has a limited number of write cycles (anything from 1,000 to 400,000 cycles), should be avoided as it can quickly cause the Flash to wear-out / fail. Similarly, if power interrupts are configured into the code, memory could easily be lost if an interrupt occurs during a write cycle. Typically, Flash memory is nearly always used as a file system to store the kernel programs as files, which are loaded into RAM as required. The RAM itself is normally used as a medium for holding transient files.

Existing users of Linux will be familiar with the LILO boot manager that configures the kernel to take into account the available external hardware and perform the necessary initialisation sequence. When running the Linux micro kernel in an embedded ColdFire Microprocessor application, the initialisation routine has to be generated specifically for the target environment.

The next section of this document will discuss how to setup an environment to allow Linux to be downloaded onto an embedded Freescale ColdFire Microprocessor. For this example, we will use the M5206e LITE evaluation board and code that has been developed through the uCLinux project. At present, the uClinux kernel supports the Arnewsh M5206AN and M5307AN evaluation boards, the Freescale M5206eLITE, the M5206eC3 and the M5307C3 evaluation boards, as well as the Moreton Bay NETtel router board which is populated with a MCF5307 Microprocessor.

1.5 Linux Kernel Download onto the MCF5206e LITE Board

In order to set-up, run and debug a Linux kernel (uClinux) on the MCF5206e LITE board, some initial work must first be carried out.

1.5.1 Downloading the Linux Kernel Source, uClinux

As discussed earlier, the versatility of Linux is derived from the fact that the user can configure the kernel. The kernel is configured by enabling / disabling certain options which build up a makefile from which the kernel is compiled. The task of producing a source package that will run on the target M5206e LITE board has already been completed by the Australian Company, Moreton Bay. The term given to this source development is “Porting”. uClinux is a port of the Linux kernel onto the Freescale ColdFire Microprocessor family.

The uClinux source packages can be retrieved, free-of-charge, (Remember that Linux is open source!), from the Moreton Bay web site at http://www.moretonbay.com/coldfire/linux-coldfire.htm. Development tools for use with uClinux are also included in this directory. Pre compiled binary images of the uClinux kernel can also be downloaded from their site at the following URL: http://www.moretonbay.com/coldfire/linux-coldfire.htm.

1.5.1.1 Un-Archiving, Compiling, Con guring and Making the uClinux Kernel

A typical uClinux kernel can be built following the steps below:

1. Retrieve the sources from Moreton Bay’s web site onto your Linux machine / partition.

As ROOT user, extract the source code from the gzip’ed tar image into a “uClinux-coldfire” directory using the command line below. (The pipe command,” |”, allows the tar extraction command to be used and executed on the same command line as the zcat command).

```
zcat uClinux-coldfire-XXXXXXXX.tar.gz | tar xvf –
```

2. Now, extract the tool binaries into a new uClinux-coldfire/tools directory using the commands:

   “zcat uClinux-coldfire-tools-XXXXXXXX.tar.gz | tar xvf –“

3. The new kernel is now ready to be built. Firstly go down into the uClinux-coldfire directory by typing:

   “cd uClinux-coldfire”

4. To configure a new Linux kernel you need to make a configuration script that will build the kernel at compile time. To do this, type:

   “make config”

The developer can then configure the Linux kernel to be as large or small as required depending on the number of drivers and support functions that are needed on the final application. (Remember that the MCF5206e LITE board is factory populated with 1Mbyte of FSRAM on board. 1Mbyte of FSRAM is considered to be the minimum system requirement for running uClinux.)

At this stage, the true versatility of creating a Linux kernel becomes obvious as you can actively select CPU type, IP firewalls, IP masquerading, drivers, amongst many other options.

When making the kernel configuration file prior to compilation, consideration should be given to the target development board when selecting the kernel options. For example, building Ethernet support onto the kernel when setting up the target for a MCF5206eLITE board would cause a compilation error since the LITE board has no Ethernet on it. [Note that the console switch, on uClinux, has not been supported for the ColdFire boards by the Moreton Bay developers.]

5. The next stage in the build is to check and build the dependencies. This checks each C file in the source and figures out which “*.h” header library file it depends on. When a “*.h” file is modified in any way, the compiler then knows which C files it needs to re-compile. To build the dependencies, type:

“make dep”

6. The user-defined kernel is now ready to compile. The chosen options that were made during the “Make config” stage will determine the applications and features that the kernel will support. The application set generated during the configuration compile stage is dependent on the "common.mk" file. The “common.mk”, file, can be configured to give an application set based on the chosen CPU/platform by manually editing the entries. To select an option, remove the leading “#” from the first character on the line. The “common.mk” file currently includes five standard options:

BUILD_BIG	-- includes every app possible in ROM file-system
BUILD_eLITE	-- builds an app set tailored for the MCF5206eLITE board
BUILD_CADR3	-- builds an app set tailored for the MCF5206 CadreIII board
BUILD_NETtel	-- builds an app set tailored for the Moreton Bay NETtel box
BUILD_eLIA	-- builds an app set tailored for the Moreton Bay eLIA box

If none of these are defined then a small, minimal ROM file-system is generated. This is suitable for boards with 1MB of RAM. On a small build, the application set is minimal and is limited to “init” and “sh” processes.

7. To make the binary image of the Linux kernel, simply type:

“make”

As the kernel is being made, the display should show all of the necessary (or dependant) files being compiled and linked together into one large binary file which is saved to the /tftpboot directory of the uClinux tree by the make file. This file can then be downloaded onto the target board.

The default build generates a raw binary image (image.bin) containing both the uClinux kernel and the ROM file system. Some debugging suites, however may require ELF format binaries or standard S-Record format images. Provision has been made to generate these images. After building the binary, an ELF image that contains the kernel and ROM file system can be created using the “make” command. It is worth noting that the combined kernel and file system ELF images are not suitable for loading onto the target via DBUG. To generate an elf image for use with gdb and the BDM interface, the following command line should be typed:

“make image.elf”

Similarly, S-Record images can be created in the exact same way. After building the binary image type:

“make image.srec”

(This command uses objcopy to generate an S-RECORD image file that contains both the uClinux kernel and the ROM file system). Errors during the make process will cause failure and are often contributed to selecting invalid options for the chosen board during the “make config” stage. If this occurs, simply re-run the kernel configuration by typing “make config” and repeat the process, selecting the correct options for your board.

The next two sections will discuss how to set up the debugging environment for downloading the code through the serial port of the target board and also how to set-up the popular GDB software for communicating to the target boards BDM (Background Debug Mode) connector.

1.5.2 Communicating With The MCF5206e LITE Board—Serial Port

To download the uClinux kernel to the M5206e LITE board through the serial COM port and the serial port of the Linux PC, a number of different options are available, again all free-ware. Basic Linux commands such as “cu” and “tip” can communicate with the serial port / DBUG as well as more advanced software packages such as “Minicom”. When using the serial port to download data to the development board only images in s-record format should be used as - DBUG corrupts downloads of non s-record formatted data containing kernel + ROM file system images.

1.5.2.1 Linux “cu” Command

With the serial port of the LITE board connected to the COM port of the Linux PC, the most convenient way, under Linux, of talking to the serial ports is using the “cu” command. In Linux, the serial COM1 port device id is “cua0”. Hence to setup a serial port connection to the M5206e LITE board with a baud rate of 19200 BPS, the command entry would be:

“cu -l cua0 -s 19200”

After typing this command, the serial communications link should be established and the normal DBUG prompt will appear. DBUG should be setup to receive the kernel data over the serial communications link via the “DL” command – (see section 5.2.3 for further details). Code can then be downloaded to the development board by entering the following command entry:

“~”

then, “$cat <filename.srec>”

This will download the uClinux kernel (in s-record format) to the host development board. The tilde (“~”) command instructs the host cu software to escape. The “$cat <filename.srec>” simply uses the standard Unix “CAT” command to send the s-record to the board via the established serial link. (A full on-line Linux manual exists for the “cu” command that gives full details of all available options – type “cu - -help”).

1.5.2.2 Freeware Source Software—“Minicom”

Minicom is a freely available, Linux text-based modem control and emulation program that can be used to communicate through the serial port to the target development board. It has excellent vt102 screen and keyboard emulation that includes full use of the entire cursor keypad and script capability. The script feature is particularly useful when automated login is required.

In order to use Minicom, it must be first configured. This is easily achieved using the different menus that should be configured for the target system / environment. Most problems with Minicom are caused by the software being mis-configured or due to read/write permission errors whilst trying to download to a target directory. Minicom can be downloaded from one of the many GNU freeware sites such as http://www.pp.clinet.fi/~walker/minicom.html.

With the correct configurations set-up for the target system / environment, the binary image of the uClinux kernel can be downloaded to the board using the “Download file” menu. Again, ensure that the correct baud rate has been selected for the target board using the setup menus prior to downloading the image. The image can then be downloaded, usually in ASCII format, onto the target using ColdFire DBUG (other modem download formats are also available).

1.5.2.3 Physical Kernel Download onto the Target ColdFire Development Board.

The easiest way to download the code onto the ColdFire M5206e LITE board is to use the DBUG monitor stored in the Flash firmware. Communication should first be established between the Linux PC and the LITE development board using some form of serial communication programs. (Refer to sections 5.2.1 and 5.2.2 for examples). Pressing return whilst a physical link is set-up by one of these programs should display the DBUG prompt.

To download the S-record file of the uClinux kernel to the LITE evaluation board you have to use the serial communications port since the board does not have Ethernet capability. Serially downloading the image onto the LITE board can be achieved using the command:

“dl 30020000”

After entering this command, the s-record can then be downloaded to the board by a simple ASCII download. Again, the “cu” command, Minicom, HyperTerminal or some other terminal program can achieve this. Once the image has been successfully downloaded into the user RAM area (location 0x30020000) of the M5206e LITE board, it can be executed using the “go” command. To start up the image on the M5206e LITE board type:

“go 0x30020000”

Each different processor built during the “make config” stage will require a different start address for the images based on the memory map of the board. For example, the M5206AN board would use a start address of 0x10000, the M5307AN board would use an address of 0x20000 and the M5206EC3 board would use an address of 0x20000. Ensure the kernel data is downloaded to the correct user RAM location in memory according to the board that is used.

If everything is correctly configured and the image has been downloaded without errors, the Linux kernel startup messages should be displayed onto the console when the DBUG monitor is given the “go” instruction.

[The M5206AN development board uses the auxiliary terminal port to output kernel I/O at a baud rate of 9600. In the case of the other ColdFire development boards, the uClinux kernel internal baud rate can be changed from the default 9600-baud rate by editing the “common.mk” file located in the top-level uClinux directory. Only a limited set of baud rates are currently supported - 9600, 19200, 38400, 57600 and 115200. This file can also be edited to include the different standard builds mentioned earlier e.g. BUILD_BIG, BUILD_eLITE etc.]

The developer should consult the uClinux NOTES page on Moreton Bay’s website for a detailed overview of the features and limitations of uClinux: http://www.moretonbay.com/coldfire/linux-coldfire.htm.

1.5.3 Communicating With The MCF5206e LITE Board—BDM Port

The GNU DeBugger, or GDB for short, allows the user to step through a C or C++ program, (or several other languages for that matter) to find the point at which they fail. Like most software for Linux, GDB is freeware, stable, highly portable and, as such, is widely considered as one of the main debuggers within the

Linux community. It has a powerful range of features such as code stepping, memory examination / modification, code disassembly, code tracing, scripting capability and software implemented breakpoint setting.

The list of commands for GDB is quite extensive and beyond the scope of this document. The source code of GDB (the most recent version is 4.18 at the time of writing) can be downloaded (freely) at http://sourceware.cygnus.com/gdb/. If GDB is supported on your Linux system kernel, on line manual help is available for this version of GDB directly through the command line of Linux, and also at http://sourceware.cygnus.com/gdb/onlinedoc.

A number of software patches have been developed for GDB to allow a seamless interface with the uClinux kernel and ColdFire family of microprocessors. Connection from the host Linux PC to the target board can be made using either the P&E or Macraigor BDM Interface cables shipped with most ColdFire development kits. These cables connect the 26-way BDM port on the target development board to the parallel port on the host Linux PC via in-line programmable logic. The ColdFire BDM device driver for use with the GDB debugger is located at http://sourceware.cygnus.com/gdb/onlinedoc.

Using the BDM port is more desirable to the developer than simply using the serial terminal as it allows access to monitor the PST and DDATA pins that can be used for real-time trace applications, improving debugging capability when compared to normal serial debugging methods. Also Ethernet downloads are possible on most ColdFire development boards via the BDM connector. Downloading over the Ethernet connection is much faster than serially downloading the data and is easily be done via the DBUG firmware. Using the SET and SHOW DBUG commands, binary images can be downloaded to the target board using the following commands:

“dn –s –o 0x10000 image.srec” or “dn –I image.bin”

Full details of downloading data over the Ethernet and all other DBUG commands are given in the documentation supplied with all Freescale ColdFire development kits.

1.5.3.1 Installing the GDB Debugging Suite

Most commercial kernels available today such as Red Hat, Suse or Debian, already have the GDB debugger built into the kernel. For those systems that don’t have this command utility on their kernel, the installation procedure is very similar to that of installing the uClinux kernel and support tools that was detailed in section 5.1.

Firstly the source code should be downloaded from the Cygnus GDB web site (URL given in section 5.3). This source code file must first be decompressed and un-tarred. To do this, enter

“tar xzvf gdb-4.18.tar.gz”

After typing this command, a new gdb-4.18 source directory should have been created which includes full documentation and help files. These should be consulted prior to further installation as they contain important information on system defaults and installation. GDB then needs to be configured for your host machine which can be automatically by executing the configure script via “bash” (Bourne Again SHell):

“./configure”

Typing the above line executes the configure script which checks the local environment and automatically configures and builds all of the tools contained in the GDB package. To make the configuration file, simply type:

“make”

Once the configuration file has been created, the GDB software can be installed using the command below. It should be noted that at this stage, that a default compiler can be specified – full information of this is given in the GDB README file.

"make install"

Once GDB is up and running, commands are entered at the command line to perform debug functions. This method of controlling the debugger may be slightly daunting for many "windows" users. Fortunately, a number of graphical front-end packages exist to add "meat to the bones" of GDB. The majority of these packages run GDB as a sub-process and are thus slightly slower than using the command line or packages that are compiled into GDB.

Cygnus Solution's debugger, Insight, (available from http://www.cygnus.com) is a debugger compiled into GDB that has a slight performance advantage over other front-end packages such as Code Medic and DDD (Data Display Debugger) as it runs as a main process of GDB. DDD is an X-windows based debugger front-end that has easy to navigate menu options and is available from the following URL:: www.cs.tu-bs.de/softech/ddd/. Code Medic is again X-windows based and incorporates a more sophisticated drag and drop display. Code Medic is available from http://www.newplanetsoftware.com/medic/.

1.5.3.2 GDB Software Patch—BDM Device Driver

Once GDB is set-up and running, the device driver for the BDM port must be configured for the ColdFire microprocessor. This is a similar exercise to building the uClinux kernel and the GDB debugger already covered.

The GDB patch at URL http://www.calm.hw.ac.uk/davidf/coldfire/gdb-4.17-bdm-990115.tar.gz. contains everything that is needed for running GDB on Linux to control a ColdFire Microprocessor through a standard Linux PC parallel port. The README text in this newly created directory should be considered mandatory reading. As mentioned before, the GDB debugger can support event tracing through the PST and DDATA pins on the BDM port. To enable access to the PST bits, the makefile in the driver/linux directory should be edited so that "USE_PST" is uncommented and enabled.

The BDM device driver must first be compiled and installed onto the system using the "make install" command line i.e

"cd driver/linux"

"make install"

This will create the BDM device driver. The next stage is to create the physical BDM port device using the "mknod" command:

"mknod /dev/bdmcf0 c 34 4"

The minor device number (the second number in the above mknod command) specifies the parallel port to which the BDM interface is connected and the type of target CPU. The least significant two bits of the minor device number specify the parallel port and the remaining bits specify the target CPU type. The example above illustrates the command line that would be entered for setting up a ColdFire BDM debug device running from parallel port LPT1. This driver can be automatically loaded into the kernel on reboot by adding the following line into the /etc/rc.d/rc.local startup script:

"/sbin/insmod bdm"

Finally, before the BDM driver is fully functional, you must compile and install the BDM library. To do this, type in the command lines below:

“cd lib”

“make install”

In order to determine if the debugging suite and BDM driver are correctly set-up, the software patch contains a test program called “chk”. Assuming that the ColdFire development board is powered up and connected from the BDM to the Linux host machine’s printer port via a suitable cable, the driver can be easily tested using the following commands:

“cd test”

“make”

“./chk /dev/bdmcf0”

If everything is correctly configured, the output display window should show a screen dump of all the registers of the target device after the memory test routine has been run. The software patch should then be applied to the GDB distribution. To do this type:

“cd …/…./gdb-4.18”

“patch -p1 </…/….gdb-4.18-bdm/gdbPatches/gdb-4.18-patch”

The final stage of setting up the BDM software patch is to compile and install the cross-GDB support. An example script is given in the “gdb-4.18-bdm/local_scripts” for reference. Full details of the installation procedure is given in the README section of the GDB, BDM software patch.

A final test should be carried out to ensure that the software with the path can connect to the target. To do this start GDB with :

“m68k-bdm-coff-gdb”

then at the GDB prompt, connect to the target by typing:

“target bdm /dev/bdmcf0”

If all is well, the GDB software should return text indicating that the remote BDM has been connected to your patched device. All of the GDB commands can now be used on the target processor to manipulate code and data.

1.6 Testing the Operation of the uClinux Kernel

The kernel was tested on a 366MHz machine with Linux / PC partitions and the following (Linux) system parameters:

Distribution:	Redhat Linux V6.1
Kernel:	2.2.12.20
Gcc:	egcs-2.91.66
Make:	GNU make 3.77
Libc:	glibc-2.1.1
Gdb:	4.18

The serial port (com1) was configured to operate at 19200 baud rate with no parity or stop bits and 8 data bits using the Linux “cu” command. The serial port of the host Linux PC was connected to the terminal port of the ColdFire M5206eLITE development board via a standard 9 way connector. (HyperTerminal, or some other terminal emulation software can be used to establish a connection between a PC and the terminal port. The selected baud rate matches the default kernel baud rate of 19200 defined during the “make config” stage of the build).

After downloading the kernel onto the target ColdFire M5206eLITE development board via the COM1 serial port the kernel was booted up by typing “go” at start location 0x30020000. Executing the code showed the kernel boot up sequence coming out from the ColdFire microprocessors auxiliary debug port.

The exact details of the boot-up sequence will vary depending on the target processor (and user memory constraints associated with each ColdFire development board) and, of course, the applications that were selected during the kernel configuration stage. The kernel boot-up console displayed on the “cu” terminal port is displayed below. Outputs for other ColdFire development boards will take a similar form to the following: (Shown below for the ColdFire M5206eLITE board)

```
Hard Reset
FSRAM Size: 1M

Copyright 1997-1999 Motorola, Inc.  All Rights Reserved.
ColdFire MCF5206e EVS Debugger v1.4.7 (Mar  2 1999 13:04:24)
Enter 'help' for help.

dBUG> dl 30020000
Offset:  0x30020000
Escape to local host and send S-records now...

S-record download successful!
dBUG> g 30020000

uClinux/COLDFIRE(m5206e)
COLDFIRE port done by Greg Ungerer, gerg@moreton.com.au
Modified for M5206eLITE by Rob Scott, rscott@mtrob.fdns.net
Flat model support (C) 1998,1999 Kenneth Albanowski, D. Jeff Dionne
KERNEL -> TEXT=0x30020000-0x3004f12c DATA=0x3004f12c-0x30059408 BSS=0x30059408-0
x30068ff0
KERNEL -> ROMFS=0x30068ff0-0x3007b668 MEM=0x3007b668-0x300ff000 STACK=0x300ff000
-0x30100000
Calibrating delay loop.. ok - 35.73 BogoMIPS
Memory available: 500k/703k RAM, 0k/0k ROM (392k kernel data, 188k code)
Swansea University Computer Society NET3.035 for Linux 2.0
NET3: Unix domain sockets 0.13 for Linux NET3.035.
uClinux version 2.0.38.0 (root@ndick) (gcc version egcs-2.91.66 19990314 (egcs-1
.1.2 release)) #9 Wed Jan 26 11:54:54 GMT 2000
ColdFire internal UART serial driver version 1.00
```

```
ttyS0 at 0x10000140 (irq = 224) is a builtin ColdFire UART
ttyS1 at 0x10000180 (irq = 225) is a builtin ColdFire UART
Blkmem copyright 1998,1999 D. Jeff Dionne
Blkmem copyright 1998 Kenneth Albanowski
Blkmem 1 disk images:
0: 30068FF0-3007B7EF (RO)
VFS: Mounted root (romfs filesystem) readonly.
Shell invoked to run file: /etc/rc
Command: hostname uClinux-coldfire
Command: mount -t proc proc /proc
Execution Finished, Exiting
Sash command shell (version 1.1.1)
/> ls
bin
dev
etc
home
lib
mnt
proc
tmp
usr
var
/>
```

A similar setup is required for the ColdFire M5206AN development board. The serial port (com1) was configured to operate at 19200 baud rate with no parity or stop bits and 8 data bits using the Linux "cu" command. The serial port of the host PC was connected to the terminal port of the ColdFire M5206AN development board via a standard 9 way connector. The auxiliary terminal port on the development board was connected to the serial port on a secondary PC to display the output from the kernel. HyperTerminal was used to establish the connection between this secondary PC and the auxiliary terminal port at 9600 baud. (This baud rate matches the default kernel baudrate of 9600 defined during the "make config" stage of the build).

After downloading the kernel onto the target ColdFire 5206 Microprocessor via the COM1 serial port the kernel was booted up by typing "go" at start location 0x10000. Executing the code showed the kernel boot up sequence coming out from the ColdFire microprocessors auxiliary debug port.

```
uClinux/COLDFIRE(m5206)
COLDFIRE port done by Greg Ungerer, gerg@moreton.com.au
Flat model support (C) 1998,1999 Kenneth Albanowski, D. Jeff Dionne
```

```
KERNEL -> TEXT=0x010000-0x03f220 DATA=0x03f220-0x0495a8 BSS=0x0495a8-0x0591a0
KERNEL -> ROMFS=0x0591a0-0x06b7f8 MEM=0x06b7f8-0x0ff000 STACK=0x0ff000-0x100000
Calibrating delay loop.. ok - 16.43 BogoMIPS
Memory available: 564k/767k RAM, 0k/0k ROM (456k kernel data, 188k code)
Swansea University Computer Society NET3.035 for Linux 2.0
NET3: Unix domain sockets 0.13 for Linux NET3.035.
uClinux version 2.0.38.0 (root@ndick) (gcc version egcs-2.91.66 19990314 (egcs-1
.1.2 release)) #10 Wed Jan 26 12:00:40 GMT 2000
ColdFire internal UART serial driver version 1.00
ttyS0 at 0x10000140 (irq = 224) is a builtin ColdFire UART
ttyS1 at 0x10000180 (irq = 225) is a builtin ColdFire UART
Blkmem copyright 1998,1999 D. Jeff Dionne
Blkmem copyright 1998 Kenneth Albanowski
Blkmem 2 disk images:
0: 591A0-6B99F (RO)
1: FFE20000-FFE3FFFF (RW)
VFS: Mounted root (romfs filesystem) readonly.
Shell invoked to run file: /etc/rc
Command: hostname uClinux-coldfire
Command: mount -t proc proc /proc
Execution Finished, Exiting
Sash command shell (version 1.1.1)
/> ls
bin
dev
etc
home
lib
mnt
proc
tmp
usr
var
/>
```

The size of available on board memory does of course limit the size of kernel and features that can be supported on the Linux kernel. A basic system should contain screen and RAM disk device drivers so that physical applications can be loaded and their outputs displayed.

1.6.0.1 Porting Applications to ColdFire Development Boards Using uClinux

All applications that run on the uClinux kernel are compiled from C source code located in the USER directory of uClinux. Each application has it's own unique directory within the USER directory. When the kernel is being built, the top level makefile has a DIRS definition which lists all of the application directories that should be included at compile time.

Within each application directory, a makefile is present to set compile options and detail the naming conventions for the final application objects that will be created. The example below shows the makefile for the applications grouped under the "fileutils" directory. (This application group creates the basic Linux commands on the kernel)

```
EXECS = cat chgrp chmod chown cmp cp dd grep l ln ls mkdir mkfifo mknod \
        more mv rm rmdir sync touch
OBJS = cat.o chgrp.o chmod.o chown.o cmp.o cp.o dd.o grep.o l.o ln.o ls.o \
        mkdir.o mkfifo.o mknod.o more.o mv.o rm.o rmdir.o sync.o touch.o
all: $(EXECS)
$(EXECS): $(OBJS)
        $(LD) $(LDFLAGS) -o $@.elf $@.o $(LDLIBS)
        $(CONVERT)
clean:
        rm -f $(EXECS) *.elf *.o
```

The "fileutils" makefile clearly shows the naming conventions to be used for the executable commands and their associated object files. The makefile also gives the compiler important information for "make" and "make clean" instructions.

As discussed previously, this "fileutils" directory must be passed into the main USER directory makefile at the DIRS definition so that it will be compiled during the kernel build. The USER (application) makefile can be customised and may take a similar form as the following:

```
#
#       Makefile -- Build instructions for user level apps
#
.EXPORT_ALL_VARIABLES:
#
# Include architecture specific build rules.
#
include arch/coldfire/build.mk
DIRS = agetty boa bpalogin chat clock dhcpcd dhcpd diald discard \
             ethattach fileutils flashw flatfsd gettyd httpd \
        ifattach inetd init init.org ipfwadm lcd levee loattach login \
        mount netflash ping pppd pptpd play rootloader route \
        sash sh shutils smbmount stty sysutils telnet telnetd tftpd \
        thttpd tip traceroute version
all:
        for i in $(DIRS) ; do make -C $$i || exit $? ; done
clean:
        for i in $(DIRS) ; do make -C $$i clean; done
```

From the above USER makefile, it is easy to see how particular applications can be added or removed from the final kernel build. Including a newly ported application into the kernel would involve the following stages: (For examples sake, we will use the fictitious application “myHDdriver”)

1. Create a new directory for the application suite in …/uClinux-coldfire/USER/myHDdriver.
2. Copy all of the relevant source and library files associated with “myHDdriver” into this newly created USER directory.
3. Create a makefile to inform the compiler of object and executable naming conventions.
4. Include “myHDdriver” directory name into the main USER makefiles DIRS definition.
5. Compile the kernel.

Porting applications is fairly straightforward although a number of issues are present that must be highlighted to the potential kernel developer.

The first issue to be aware of is the compilation of the ported application. The uClinux kernel has a limited libc, C library, hence it may be necessary to add or remove library include files to or from the library for successful compilations. Unlike normal Linux kernels, the uClinux kernel does not support virtual memory since memory is limited in an embedded kernel.

Since there is no virtual memory to grow user memory stacks on the uClinux kernel, they must be fixed in size. The default setting of the kernel fixes them in size to 4K although this can be increased using the “-s” option of “elf2flt”. (The “elf2flt” utility converts’ elf records to flat format binaries for subsequent download to the development board)

Due to Linux’s allocation algorithm and the limited memory available on a typical embedded system, the maximum allocation size is limited to 256K on the uClinux kernel. The exec() loader will not be able to load any binary image that is bigger than 256K. Again, the lack of virtual memory makes process ID system calls, or forks impossible, and uses a virtual fork instead (vfork).

The above paragraphs detail the limits of the uClinux port. Porting applications from another microprocessor onto a ColdFire microprocessor running a uClinux kernel may require modifications to take into account the different register setups, interrupt / trap sequences, timers, addressing etc.

As with all development work on Linux, ported applications or code should be verified and stored in the central archive for the use and enjoyment of other developers. This is the essence of all Linux work.

[Freescale does not in any way endorse Moreton Bay Enterprises or any other software supplier or developer mentioned in this document. Freescale also does not accept any responsibility for the reliability or support of the code mentioned in this text. The uClinux port discussed in this document is to illustrate purely how Linux can be embedded onto the ColdFire MCF5206e LITE board. The code discussed is GNU open source and as such all software restrictions and rules apply.]

1.7 Data Sources and References

- *Linux for Dummies* by John "maddog" Hall, IDG Books
- *Linux Server Sales Surge As NT Treads Water* by John Lettice, http://www.theregister.co.uk / http://www.theregister.co.uk

The following is a list of useful Linux links:

- http://linux-embedded.com/
- http://www.linux-m68k.org/
- http://www.linux.org/
- http://www.uclinux.org/
- http://www.moretonbay.com/
- http://www.cygnus.com/
-

Home Page:
www.freescale.com
email:
support@freescale.com
USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com
Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com
Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 2666 8080
support.japan@freescale.com
Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com
For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com