Porting Linux to the M32R processor

Hirokazu Takata
Renesas Technology Corp., System Core Technology Div.
4-1, Mizuhara, Itami, Hyogo, 664-0005, Japan
takata.hirokazu@renesas.com

Naoto Sugai, Hitoshi Yamamoto
Mitsubishi Electric Corp., Information Technology R&D Center
5-1-1, Ofuna Kamakura, Kanagawa 247-8501, Japan
{sugai,hitoshiy}@isl.melco.co.jp

Abstract

We have ported a Linux system to the Renesas[1] M32R processor, which is a 32-bit RISC microprocessor designed for embedded systems, and with an on-chip-multiprocessor feature.

So far, both of UP (Uni-Processor) and SMP (Symmetrical Multi-Processor) kernels (based on 2.4.19) have been ported and they are operating on the M32R processor. A Debian GNU/Linux based system has been also developed on a diskless NFS-root environment, and more than 300 unofficial `.deb` packages have already been prepared for the M32R target.

In this paper, we describe this new architecture port in detail and explain the current status of the Linux/M32R project.

[1]Renesas Technology Corp. is a new joint semiconductor company established by Hitachi Ltd. and Mitsubishi Electric Corp. on April 1, 2003. It would be the industry's largest microcontroller (MCU) supplier in the world. The M32R family microcontroller and its successor will be continuously supplied by Renesas.

1 Introduction

A Linux platform for Renesas M32R processor has been newly developed. The Renesas M32R processor is a 32-bit RISC microprocessor, which is designed for embedded systems. It is suitable for a System-on-a-Chip (SoC) LSI due to its compactness, high performance, and low power dissipation. So far, the M32R family microcomputers have widely used for the products in a variety of fields—for example, automobiles, digital still cameras, digital video camcorders, cellular phones, and so on.

Recently, the Linux system has begun to be used widely and employed even in the embedded systems. The embedded systems would be more software-oriented systems hereafter. The more complex and larger the embedded system is, the more complicated the software becomes and harder to develop. In order to build these kinds of embedded systems efficiently, it will be more important to utilize generic OSes such as Linux to develop software.

This is the first Linux architecture port to the M32R processor. This porting project, called a "Linux/M32R" project, has been active since

2000. Its goal is to prepare a Linux platform for the M32R processor. At first, this Linux porting was just a feasibility study for the new M32R processor development, and it was started by only a few members of the M32R development team. Then, this project has grown to a lateral project among Renesas Technology Corp., Renesas Solutions Corp., and Mitsubishi Electric Corp.

In this feasibility study, we have ported not only Linux kernel, but also whole GNU/Linux system including GNU tools, libraries, and other software packages, so called "userland." We also enhanced the M32R processor to add MMU (Memory Management Unit) facility in order to port Linux system. And we have also developed an SMP kernel to investigate multiprocessing by M32R's on-chip-multiprocessor feature[1]. At present, the Linux/M32R system can operate on the dual M32R cores in SMP mode.

In this paper, we describe this new architecture port in detail and explain about the current status of the Linux/M32R project.

2 Linux/M32R Platform for Embedded Systems

Recently, due to the continuous evolution of semiconductor technologies, it is possible to integrate a whole system into one LSI chip, so called "System-on-a-Chip (SoC)."

In an SoC, microprocessor core(s), peripheral I/O functions, internal memories, and user logics can be integrated into a single chip.

By making use of wide internal buses, an LSI can achieve high performance which can not be realized by combination of several general-purpose LSIs. In other words, we can optimize system performance and cost by using SoC, because we can employ optimum hardware architecture and circuit configuration.

In such an SoC, a microprocessor core is a key part; therefore, the more compact and higher performance microprocessor is significantly required. To make such a high performance embedded processor core, not only in circuit and process technology but also architectural breakthrough is necessary. Especially, multiprocessor technology is important even for the embedded processor, because it can improve processor performance and lower system power dissipation by increasing processor number scalably and without increasing operating clock frequency.

For an SoC with embedded microprocessor, software is also a key point. The more system is highly functional, the more software will be complex and there is an increasing demand for shortening development time of SoC. Under such circumstance, recently the Linux OS becomes to be adopted for embedded systems. Linux platform makes it easy to port application programs developed on a PC/EWS to the target system. We believe that a full-featured Linux system will come into wide use in embedded systems, because embedded systems will become more functional and higher performance system will be required.

In the development of embedded systems, it is important to tune system performance from both hardware and software points of view. Therefore, we used M32R softmacro and FPGA (Field Programmable Gate Array) devices to implement an evaluation board for rapid system prototyping. FPGA devices are slow, but make it possible to develop a system in short turn-around time.

In this feasibility study, to construct a Linux platform, we ported Linux to the M32R architecture, and validated the hardware system architecture through the porting, and developed the software development environment.

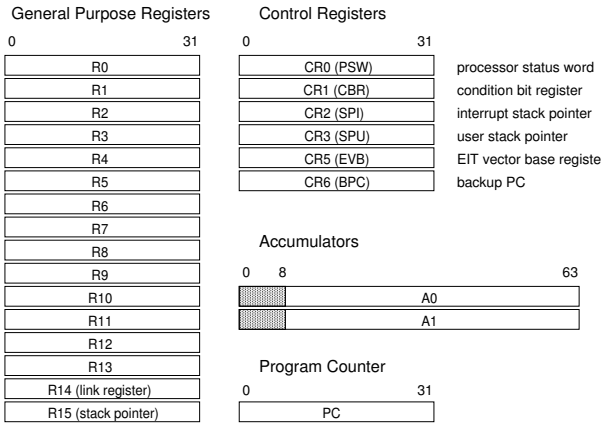


Figure 1: M32R register architecture

2.1 M32R architecture

The M32R is a 32-bit RISC microprocessor, and employs load-store architecture like other RISC processors. Therefore, memory access is executed by only load and store instructions and logical and arithmetic operation is executed among registers. Except for multiply and divide instructions, most of instructions can be executed in one clock, and instruction completion does not depend on the the instruction issuing order (out-of-order completion). The M32R supports DSP operation instructions such as multiply and accumulate instructions.

Figure 1 shows the M32R register architecture. The M32R has sixteen 32-bit general purpose registers (R0 $\sim$ R15) and 56-bit accumulators for the multiply and accumulate operations. R14 is also used as a link register (LR) which keeps return address for a subroutine call. There are two stack pointer registers, SPI (interrupt stack pointer) and SPU (user stack pointer). The CPU core selects one of them as a current stack pointer (R15; SP) by the SM (stack mode) bit of the PSW (processor status word).

2.2 M32R softmacro

The M32R softmacro is a compact microprocessor, developed to integrate into a SoC. It is a full synthesizable Verilog-HDL model and it has an excellent feature that the core does not depend on a specific process technology. Due to a synchronous edge-triggered design, it has good affinity to EDA tools.

This M32R softmacro is so compact that it can be mapped into one FPGA. Utilizing such an M32R softmacro, we developed software on a prototype hardware and co-designed hardware and software simultaneously.

To port Linux to the M32R, some enhancement of the M32R softmacro core was needed; processor mode (user mode and supervisor mode) was introduced and an MMU module was newly supported.

- TLB (Translation Lookaside Buffer): instruction/data TLBs are full-associative, 32-entries each, respectively.
- page size : 4kB/16kB/64kB (user page), 4MB (large page)

2.3 Integrated debugging function and SDI

The integrated debugging function is a significant characteristic of the M32R family microcomputer. The M32R common debugging interface, called as SDI (Scalable Debug Interface), is utilized via five JTAG pins; the internal debug functions are controlled through these debug pins.

Using the JTAG interface defined as IEEE 1149.1, internal debug function can be used. No on-chip or on-board memory to store monitor programs is necessary, because such monitor programs can be provided and executed via JTAG pins.

3 Porting Linux to the M32R

The Linux system consists of not only the Linux kernel, but also the GNU toolchain and libraries. Of course, a target hardware environment is also necessary to execute Linux.

Therefore we had to accomplish the following tasks:

- Porting the Linux kernel
- Development of Linux/M32R platforms (M32R FPGA board, etc.)
- Enhancement of the GNU toolchain
- Porting libraries (GNU C library, etc.)
- Userland; preparing software packages

Actually, in the Linux/M32R development, these tasks have developed concurrently.

3.1 Porting Kernel to the M32R

In the Linux kernel, the architecture-dependent portion is clearly distinguished. Therefore, in case of a new architecture port, all you need to do is prepare only architecture dependent code, which is far less than whole Linux code. In the M32R case, `include/asm-m32r/` and `arch/m32r/` are needed.

To be more precise, we can prepare the architecture-dependent portion in reference to the other architecture implementation. However, it has some difficulties in rewriting these portions:

asm function : It was very difficult to port some headers in which asm statement is extensively used, because an insufficient and inadequate rewriting easily cause bugs which are very hard to debug.

function inlining : In the Linux source code, function inlining is heavily used. We can not disable the compiler's inline optimization function to compile the kernel source, but a buggy toolchain sometimes causes a severe problem in optimization.

In this porting, we started with a minimum configuration kernel. We prepared stub routines, built and checked the kernel operation, and gradually added files of `kernel/`, `mm/`, `fs/`, and other directories. When we started kernel porting, there was no evaluation board. So, we made use of GNU M32R simulator to port a kernel at first.

The GNU simulator was very useful at the initial stage of kernel porting, though it does not support MMU. It had also good characteristics that downloading was quite fast comparing with evaluation board and C source level debug was possible.

Employing the simulator and initrd image of romfs root filesystem, it is possible to develop and debug kernel's basic operation, such as scheduling, initialization and memory management. Indeed, the demand loading is performed after `/sbin/init` is executed by a `execve()` system call at the end of kernel boot sequence of `init()`.

At first, we started to port the kernel 2.2.16. The current stable version of the M32R kernel is 2.4.19 and now we are developing 2.5 series kernel for the M32R.

3.1.1 System Call Interface

In the M32R kernel, like other processors, a system call is handled by a system call trap interface. In the system call interface (syscall I/F), all general purpose registers and accumulators are pushed onto the kernel stack to save

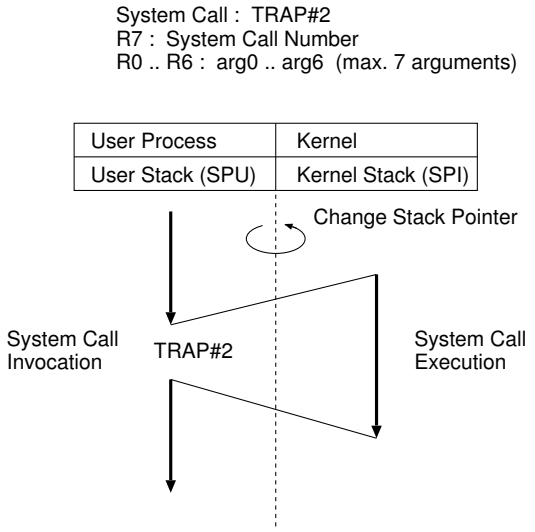


Figure 2: System call interface

the context.

In Fig. 2, the syscall ABI (Application Binary Interface) for the M32R is shown. Two stack pointers, kernel stack (SPI) and user stack (SPU), are switched over by software at the entry point of syscall I/F routine, because stack pointers do not change automatically by `TRAP` instruction. In order to switch stack pointers without working register and avoid multiple TLB miss exception, `CLRPSW` instruction is newly introduced.

The stack frame formed by the syscall I/F routine is shown in Fig. 3. It should be noted that there is a special system call in Linux, like sys_clone(), that has particular interface passing a stack top address as a first argument. Therefore, we employ a parameter passing method: The stack top address (∗pt_regs) is always put onto the stack as a implicit stack parameter like the SuperH implementation.

According to the ABI of the M32R gcc compiler, the first 4 arguments are passed by registers and the following arguments are passed by stack. Therefore, the ∗pt_regs parameter can be accessed as the eighth parameter on the kernel stack.

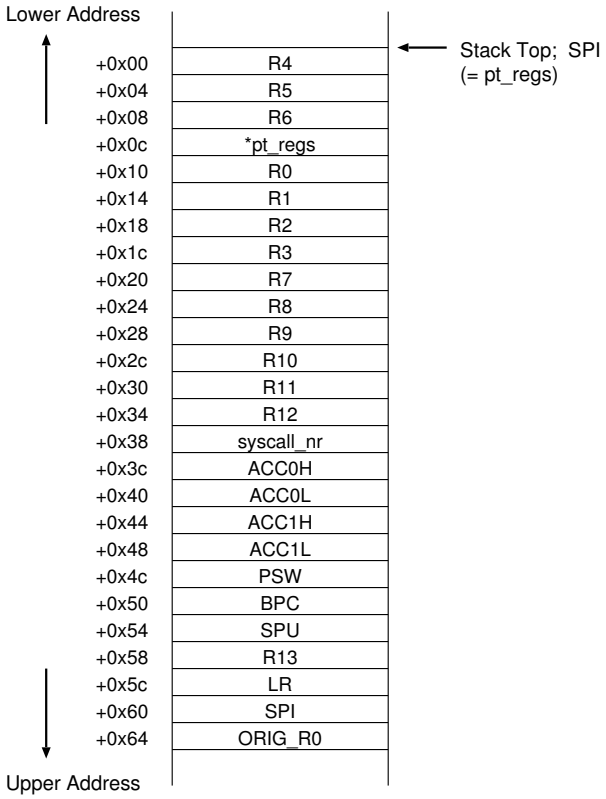


Figure 3: Stack frame formed by a system call

The `syscall_nr` and `ORIG_R0` field are used for the signal operations. When a system call is issued, its system call number is stored into `R7` and trap instruction is executed. `syscall_nr` also holds the system call number in order to determine if a signal handler is called from a system call routine or not. Because the `R0` field might be changed to the return value of a system call, `ORIG_R0` keeps the original value of `R0` in preparation to restart the system call.

3.1.2 Memory Management

Linux manages the system memory by *paging*. In the M32R kernel, the page size is 4kB like the other architecture.

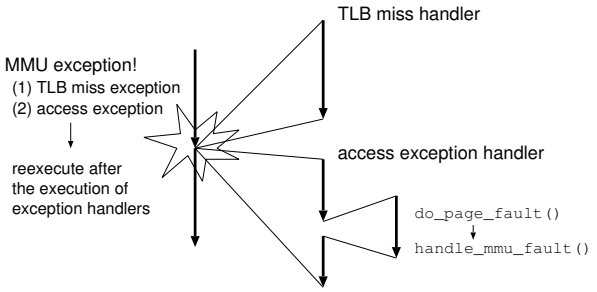


Figure 4: Exception handling for the demand-loading operation

In demand loading and copy-on-write operations, a physical memory page can be newly mapped when a *page fault* happens. Such a page fault is handled by both *TLB miss handler* and *access exception handler* (Fig. 4).

demand loading : If an instruction fetch or operand access to the address which is not registered in page table, an MMU exception happens. In case of a TLB miss exception, TLB miss handler is called. To lighten the TLB miss handling operation, TLB handler only sets TLB entries. Page mapping and page-table setting operations are to be handled by the access exception handler; For accessing a page which does not exist in the page table, the TLB miss handler sets the TLB entry's attribute to not-accessible at first. After that, since the memory access causes an access exception due to not-accessible, access exception handler deal with the page table operations.

copy-on-write : In Linux, copying a process by `fork()` and reading a page in read-only mode are handled as a copy-on-write operation to reduce vain copy operations. For such a copy-on-write operation, TLB miss handler and access exception handler are used like a demand loading operation.

The M32R's data cache (D-cache) is indexed and tagged physically. So, it does not have to take care the cache aliasing. Therefore the D-cache is flushed only for a signal handler generation and a trampoline code generation.

To simplify and speed up the cache flushing operations for trampoline code, a special cache flush trap handler (trap#12) is established in the M32R kernel.

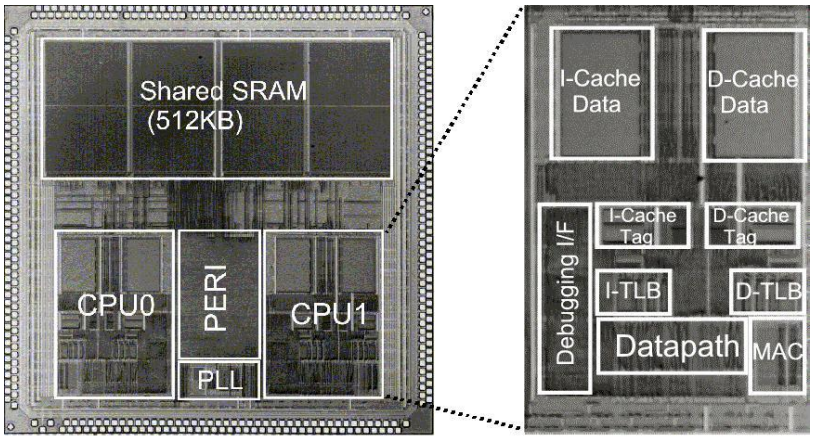
3.1.3 SMP support

In Linux 2.4, multiprocessing performance is significantly improved compared with Linux 2.2 or before, because the kernel locking for accessing resources is finer.

To implement such a kernel locking on SMP kernel, *spinlock* is generally used for mutual exclusion control. But the M32R has no atomic test-and-set instruction, the spinlock operations can be implemented with `LOCK` and `UNLOCK` instructions in the M32R kernel.

The `LOCK` and `UNLOCK` instructions are a load and store instructions for mutual exclusion operation, respectively. `LOCK` acquires and keeps a privilege to access the CPU bus until `UNLOCK` instruction is executed. Accessing a lock variable by `LOCK/UNLOCK` instruction pair under a disabled interruption condition, we implemented an atomic access.

Figure 5 shows an M32R on-chip-multiprocessor prototype chip. Linux SMP kernel can be executed on the on-chip multiprocessor system. On-chip multiprocessor might be a mainstream in near future even in embedded systems, because multiprocessor system can enhance the CPU performance without increasing operating clock frequency and power dissipation. In this chip, two M32R cores are integrated and each has its own cache for good scalability.



(a) Chip (b) CPU

Figure 5: A micrograph of an on-chip-multiprocessor M32R prototype chip

3.2 Development of Linux/M32R Platform

To execute full-featured Linux OS, an MMU is necessary; therefore, we developed a new M32R softmacro core with an MMU and made an evaluation board "Mappi," which used FPGAs to map the M32R softmacro core, as a Linux/M32R platform.

As shown in Fig. 6, the Mappi evaluation board consists of two stacked boards. The upper board is a CPU board and the lower board is an extension board. The CPU board has no real CPU chip, but it has two large FPGAs on it. We employ the M32R softmacro core and map it onto the FPGAs.

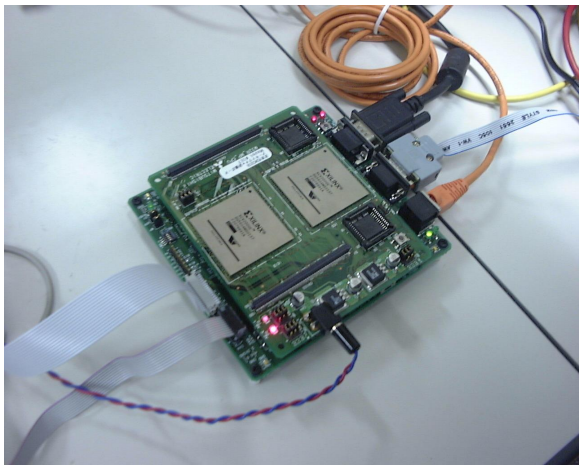
The Mappi board is a very flexible system for prototyping. If we have to modify a CPU or other integrated peripherals, we can immediately change and fix them by modifying their Verilog-HDL model.

At first, we could only use `initrd` and `busybox` on it, because the Mappi system had only a CPU board and it had only 4MB SRAMs. After the extension board was developed, more memory (SDRAM), Ethernet, and PC-card I/F became available. So, we introduced NFS and improved the porting environment. It was Dec. 2001 that we succeeded in booting via a network using the extension board.

Utilizing the M32R's SDI function and JTAG-ICE, mentioned before, we can download and debug a target program via JTAG port. It is much faster than a serial connection because the Debug DMA function is used for downloading and refering internal resources. Of cource, it is also possible to set hardware breakpoints for the PC break and the access break via SDI.

Generally speaking, it is too difficult to develop and debug software programs on an unsteady hardware which is under development. But, we could debug and continued to develop the system by using the SDI debugger, because the SDI debugger made it possible to access the hardware resources directly and it was very useful for the kernel debugging.

Finally, we constructed an SMP environment to execute the SMP kernel, mapping the M32R softmacro cores to two FPGAs on the Mappi



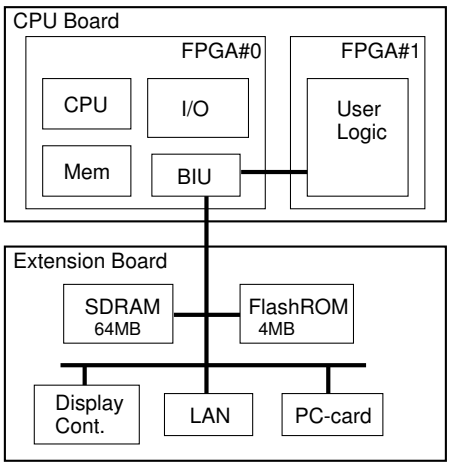


Figure 6: Mappi: the M32R FPGA evaluation board; it has the M32R softmacro on FPGA (CPU, MMU, Cache, SDI, SDRAMC, UART, Timer), FPGA Xilinx XCV2000E ×2, SDRAM(64MB), FlashROM, 10BaseT Ethernet, Serial 2ch, PC-card slot ×2, and Display I/F(VGA)

CPU board; concretely, we replaced the user logic portion in FPGA#1 shown in Fig. 6 with an another M32R core with a bus arbitor, and modified the ICU (Interrupt Control Unit) to support inter-processor interruption for the multiprocessor.

After the M32R prototype on-chip-multiprocessor chip was developed, the Linux/M32R system including userland applications has been mainly developed by using the real chip, because the operating clock frequency of the M32R FPGA is 25MHz but the M32R chip can run more than 10 times faster.

3.3 M32R GNU toolchain enhancement

The GNU toolchain is necessary to develop the Linux kernel and a variety of Linux application programs. When we started the Linux porting, we had only Cygnus's (now it's Red Hat, Inc.) GNUPro™m32r-elf toolchain. It was sufficient for the kernel development; however, it could not be applicable to user application development on Linux, because in a modern UNIX system a dynamic linking method is strongly required to build a compact system and achieve higher runtime performance. (Although a static linked program is much faster than a dynamic linked program if the program size is small. The bigger a program becomes, the larger cache miss penalty would be.)

We enhanced the M32R GNU toolchain to support shared libraries:

- Change BFD library to support dynamic linking; some relocations were added for dynamic linking.
- Change GCC and Binutils to support PIC (Position Independent Code).

Because the version of the GNUpro m32r-elf gcc was 2.8 and too old, we had to upgrade and develop a new m32r-linux toolchain. We applied GNUpro patch to the gcc of the FSF version and developed GCC (v2.95.4, v3.0, v3.2.2) and Binutils (v2.11.92, v2.13.90).

In a prologue portion of a C function, the following code is generated when the `-fPIC` option is specified.

```
; PROLOGUE
push r12
push lr
bl .+4  ; get the next instruction's
        ; PC address to lr
ld24 r12,#_GLOBAL_OFFSET_TABLE_
add r12,lr
```

We also modified BFD libraries to support dynamic linking. We referenced the i386 implementation and supported the ELF dynamic linking. In the ELF object format [3], GOT (Global Offset Table) and PLT (Procedure Linkage Table) are used for the dynamic linking. In the M32R implementation, the GOT is refered by R12 relative addressing and the RELA type relocation is emoployed. Like a IA-32 implementation, the code fragment of PLT refers the GOT to determine the symbol address, because it is suitable and efficient for the M32R's cache which can be simply flushed whole caching data.

As for GDB, we enhanced it to support a new remote target `m32rsdi` to use the SDI remote connection. By using the gdb, we can do a remote debugging of the kernel in C source-level. In the latest version of `m32r-linux-gdb/ m32r-linux-insight` (v5.3), we have employed a `SDI server` that engages in accessing the JTAG port of the ICE/emmulator connected with the parallel port of the host PC. This gdb makes it possible to debug using SDI, communicating with the SDI server in background. Though the SDI server requires privileged access to use parallel port, we can use gdb in user mode.

3.4 Porting GNU C library

The GNU C library is the most fundamental library, which is necessary to execute a variety kind of application programs. So we decided to port it to implement full-featured Linux system for a study, though its footprint is too large for a tiny embedded system.

We started to port glibc-2.2.3 (v2.2.3) in eary stage of the Linux/M32R porting, it was about the same time that the kernel's scheduler began to work.

Then, the glibc for the M32R have been developed step by step;

- Check by a statically linked program, for example, `hello.c` (newlib version → glibc version).
- Build a shared library version of glibc and check by dynamically linked programs, `hello.c, busybox, etc.`
- Port the LinuxThreads library to support Pthreads (POSIX thread).

The latest version of the glibc for the M32R is glibc-2.2.5 (v2.2.5). It also supports a LinuxThreads library, that implements POSIX 1003.1c kernel threads for Linux. In this LinuxThreads library, we implemented fast user-level mutual exclusion using the Lamport's algorithm [2], because the system call implementation was quite slow due to context switching.

After the glibc porting was finished, we started to build various kind of software. But it has taken several months to implement and debug the following:

- Fixup operations of the user_copy routines in the kernel
- Resolve the relocation by a dynamic linker `ld-linux.so`
- Signal handling

Especially, the dynamic linking operation was the one of the most difficult portions in this

GNU/Linux system porting, because the dynamic linker/loader resolved global symbols and subroutine function addresses in runtime. Furthermore, the dynamic linker itself is also a shared library, so we can not debug it in C source-level. However, we debugged the linker, making use of a simulator, a SDI debugger, and all kinds of things.

3.5 Userland

For the sake of preparing software packages and making the Linux/M32R distributable, we built major software packages.

We chose the Debian GNU/Linux as a base distribution, because it is well-managed and all of the package sources are open and published. In Debian, using command programs such as `dpkg` and `apt`, it is possible to manage abundant software packages easily.

To build a binary package for the M32R, we did as the following:

1. Expand the source tree from the Debian source package (*.dsc and *.orig.tar.gz)
2. Rebuild a binary package by using a `dpkg- buildpackage` command, specifying the target architecture to `m32r` (`dpkg-buildpackage -a m32r -t m32r-linux`).

So far, more than 300 unofficial `.deb` packages have been prepared for the M32R target, including the basic commands, such as self-tools and shells, utilities, package management tools (`dpkg`, `apt`), and application programs as follows:

> adduser, anacron, apt, base-files, bash, bc, binutils, bison, boa, bsdgames, bsdutils, busybox, coreutils, cpp-3.2, cvs, debianutils, devfsd, diff, dpkg, e2fsprogs, elvis-tiny, expect, file, fileutils, findutils, flex, ftp, g++-3.2, gcc-3,2, grep, gzip, hostname, klogd, less, libc6, locales, login, lynx, m4, make, mawk, modutils, mount, nbd-client, net-tools, netbase, netkit-inetd, netkit-ping, passwd, perlbase, perl-modules, portmap, procps, rsh-client, rsh-server, samba, sash, sed, strace, sysklogd, tar, tcl8.3, tcpd, tcsh, telnet, textutils, util-linux, wu-ftpd, ...

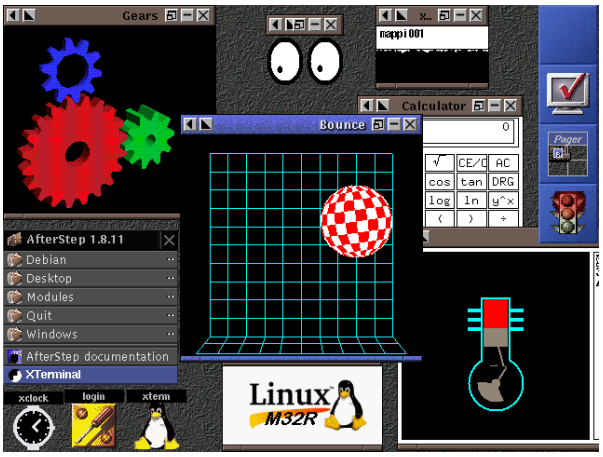


Figure 7: A snapshot of the desktop image of X; the window manager is `AfterStep`

Most of these packages were developed under the cross environment, except some software packages, such as Perl, Xserver, gcc, etc. Because they had to be configured in the target environment. Therefore, self tools were necessary in order to build packages under the self environment. Fortunately, by using `.deb` packages and `dpkg-cross` commands, the same package files can be completely shared between the self and the cross environment.

Regarding the GUI environment, we have ported some window systems (X, Microwindows, Qt-Embedded). We ported them easily using a framebuffer device. Figure 7 is a sample screen snapshot of X desktop image. `gears` and `bounce` are demonstration pro-

grams of the Mesa-3.2 3D-graphics library.

4 Evaluation

The Linux/M32R system's conformance have been checked and validated by using the LSB (Linux Standard Base) testsuites, which are open testsuites and based on the LSB Specification 1.2 [5]. In this validation, we compared the following two environments.

Linux/M32R
based on the Debian GNU/Linux (sid)
linux-2.4.19 (m32r_2_4_19_20030109)
glibc-2.2.5 (libc6_2.2.5-6.4_m32r.deb)
gcc-3.0 (self gcc; m32r-20021112)

RedHat7.3 [2]
linux-2.4.18-10 (kernel-2.4.18-10.i686.rpm)
glibc-2.2.5 (glibc-2.2.5-42.i686.rpm)
gcc-2.96

The result of validation is shown in Table 1. Judging from the result, the LSB conformance of the Linux/M32R is no less good than the RedHat Linux 7.3, because the original Debian distribution has basically good LSB conformance and quality.

5 Future work

To apply the Linux/M32R to embedded systems, it is indispensable to tune and shrink the whole system more and more. As for the kernel, particulary, tuning and improving realtime performance will be strongly required.

At present, we are porting the Linux 2.5 series kernel for the M32R in order to support the state of the art kernel features, such as O(1) scheduler, the preemptible kernel, the no-MMU support, the NUMA support, and so on.

We are also planning to utilize DMA function and internal SRAM to increase Linux system performance. And for the high-end embedded systems, we intend to continuously focus on the SMP kernel for the on-chip-multiprocessor.

6 Summary

To build a Linux platform, we have ported a GNU/Linux system to the M32R processor. In this work, a hardware/software co-design methodology was employed, using a full synthesizable M32R softmacro core to accelerate Linux/M32R development. To develop SoC in a short time, such a hardware and software co-design and co-debugging methodology will become more important hereafter.

Linux will play a great role in the field of not only PC servers but also embedded systems in the near future. Through the feasibility study, we believe that the Open Source will provide a quite large impact on developing embedded system design and development. If we have opportunity, we hope to publish the Linux/M32R and M32R GNU toolchain.

7 Acknowledgements

The authors greatly acknowledge the collaboration and valuable discussion with the M32R development team [1] and thank Takeo Takahashi, Kazuhiro Inaoka, and Takeshi Aoki for their special contributions, and we would also like to thank Dr. Toru Shimizu and Hiroyuki Kondo for their promotion of the M32R processor development project.

[2] The kernel and glibc are upgraded and different from the original Red Hat 7.3 distribution.

Section		ANSI.hdr	ANSI.os		POSIX.hdr	POSIX.os		LSB.os		Total	RedHat7.3
			F	M		F	M	F	M		Total
Total	Expect	386	1244	1244	394	1600	1600	908	908	8284	8284
	Actual	386	1244	1244	394	1600	1600	908	908	8284	8284
Succeeded		176	1112	86	207	1333	0	695	0	3609	3583
Failed		4	0	0	5	2	0	49	0	60	45
Warnings		0	12	0	0	5	0	2	0	19	18
FIP		2	0	0	2	2	0	1	0	7	7
Unresolved		0	0	0	0	0	0	5	0	5	4
Uninitiated		0	0	0	0	0	0	0	0	0	0
Unsupported		203	0	0	179	72	0	59	0	513	513
Untested		0	4	0	0	7	0	39	0	50	43
NotInUse		1	116	1158	1	179	1600	58	908	4021	4021

Key: F:function, M:macro; FIP: Further Information Provided

Table 1: LSB 1.2 testsuites result

References

[1] Satoshi Kaneko, Katsunori Sawai, Norio Masui, Koichi Ishimi, Teruyuki Itou, Masayuki Satou, Hiroyuki Kondo, Naoto Okumura, Yukari Takata, Hirokazu Takata, Mamoru Sakugawa, Takashi Higuchi, Sugako Ohtani, Kei Sakamoto, Naoshi Ishikawa, Masami Nakajima, Shunichi Iwata, Kiyoshi Hayase, Satoshi Nakano, Sachiko Nakazawa, Osamu Tomisawa, Toru Shimizu, *A 600MHz Single-Chip Multiprocessor with 4.8GB/s Internal Shared Pipelined Bus and 512kB Internal Memory*, Proceedings of 2003 International Solid-State Circuits Conference, 14.5.

[2] Laslie Lamport, *A Fast Mutual Exclusion Algorithm*, ACM Trans. on Computer System, Vol. 5, No. 1, Feb. 1987, pp. 1-11.

[3] *Executable and Linkable Format (ELF)*, `http://www.cs.northwestern.edu/~pdinda/ics-f01/doc/elf.pdf`

[4] *Debian GNU/Linux*, `http://www.debian.org/`

[5] *LSB testsuites*, `http://www.linuxbase.org/test/`, `ftp://ftp.freestandards.org/pub/lsb/test_suites/released-1.2.0/runtime/`

Proceedings of the Linux Symposium

July 23th–26th, 2003
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Alan Cox, *Red Hat, Inc.*
Andi Kleen, *SuSE, GmbH*
Matthew Wilcox, *Hewlett-Packard*
Gerrit Huizenga, *IBM*
Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Martin K. Petersen, *Wild Open Source, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*