Linux® As a Real-Time Operating System

Freescale Semiconductor
Author, David Beal

Document Number: SWVERIFICATIONWP
Rev. 0
11/2005

freescale™
semiconductor

CONTENTS

1. Executive Summary

From 1998 until 2002, Freescale's engineers developed and integrated the dual-kernel, real-time solutions for Linux® (RTLinux and RTAI) with our Linux board support packages (BSPs). These solutions provide sub-20 microsecond precision with hard, real-time scheduling guarantees but at the expense of increased programming effort and memory protection.

Now, with the evolution and maturation of the pure Linux approaches to real-time, Freescale includes these enhancements within our newly released Linux BSPs. These techniques offer less than 250 uSec latency performance for greater than 99.5 percent of all scheduled tasks under demanding system test loads while significantly easing application development and maintaining memory protection. (See included graphs for performance details.) Under real-world conditions, performance is near the 99.5 percent, 100 uSec mark.

Today's standard Linux kernel and downloadable patches/modifications include many features and performance enhancements that make it suitable for challenging real-time products and applications, including:

> Enhanced Schedulers

> Virtual Memory

> Shared Memory

> Portable Operating System Interface X s(POSIX) Timers

> Real-Time Signals

> POSIX Asynchronous I/O

> POSIX Threads

> Quality of Service Capabilities

> Low Latency/Preemptable Kernel Modifications

Of these features, a large number of developers focus on the low latency/preemptable kernel modifications and their resultant effect on kernel latency. On this subject, third-party independent reports included within this document and originally published online (at http://www.mnis.fr/opensource/ocera/rtos/x1422.html) indicate very positive and real-world useful results, showing that, for heavily loaded systems, both the preemptable and low-latency enhancements achieve greater than 99.5 percent task scheduling with less than 250 uSec interrupt latency (see included test results) on a heavily loaded test system. In the real world, it is reasonable to expect this performance to be closer to 100 uSec.

Although this data shows that the low latency modification yields a marginal performance benefit, we stress that the performance difference between these approaches is of such a small magnitude that the suitability of one technique for any specific product application is equal to the suitability of the other technique for that same application. Caution is advised because with such small performance variances, it is likely that a subjective test could be designed in such a way that the tester's "preferred" solution might show itself to be superior, or that key system stresses (e.g., fork) which might adversely effect the average or maximum latency values may not be tested at all.

2. Linux Overview

Linux is a full-featured UNIX® implementation. The main design criterion of the Linux kernel is the throughput, while real-time and predictability is not an issue. The main handicap to considering Linux as a real-time system is that the kernel is not preemptable; that is, while the processor executes kernel code, no other process or event can preempt kernel execution.

Although Linux is not a real-time system, it has some features, already included in the mainstream source code or distributed as patch files, designed to provide real-time to Linux. These are the features described in this section.

From the programmer point of view, there are two main programming paradigms to build a real-time application: weight-processes (normal UNIX processes) and lightweight-processes (known as threads or LWP). Linux provides support for both execution environments, mostly based on POSIX standards 1003.b and 1003.c, respectively.

Real-time system POSIX threads, memory locking, mutexes and condition variables (condvars) calls are implemented and distributed in the GNU "C" library (glibc). Asynchronous I/O and POSIX clocks and timers are located in a separate library called "librt". Therefore, if you want to make use of AIO and POSIX clocks and timers, the compiler must be invoked flag the *-lrt.*

2.1 Real-Time Features of Linux Scheduling

Even in early Linux kernel releases, the scheduler was real-time POSIX compatible. It supports two fixed priority scheduling policies, each with priorities from [0 to 99]:

> SCHED_FIFO—when faced with two equal priority tasks, SCHED_FIFO allows the first task to complete before scheduling the other task.

> SCHED_RR—when faced with two equal priority tasks, SCHED_RR time slices.

A lot of work has been done to improve the performance of the scheduler through a careful design that yields to a new scheduler code and structure. Linux kernel v2.4.19 and beyond, as well as the unstable kernel development tree (2.5.x) includes a new scheduler which replaces the old scheduler code with an improved "O (1) scheduler" developed by Ingo Molnar. This new scheduler is able to manage a large number of processes with no overhead degradation.

2.2 Virtual Memory

Real-time application tasks must be prevented from being swapped out of memory because the random and long delays introduced when RAM is exhausted and swapping is required are intolerable in a real-time system.

To support this, Linux provides the `mlock()` and `mlockall()` functions that disable paging for the specified range of memory, or for the whole process respectively. Therefore, all the "locked" memory will stay in RAM until the process exits or unlocks the memory.

`mlock()` and `mlockall()` are included in the POSIX real-time extensions.

2.3 Shared Memory

One of the main communications methods used by real-time applications is shared memory.

Linux processes can share memory with each other and with drivers by using the POSIX.1b call `mmap()`. This function can be used to map a regular file into main memory, and to map shared memory objects. When multiple processes map the same memory object (or file), they share access to the underlying data, which is an efficient way to communicate large amounts of data between processes.

From version 2.4.x and glibc 2.2 (GNU "C" library), Linux provides open shared memory objects, which are a part of the POSIX real-time extensions. This API has the following functions: `shm_open()` and `shm_unlink()`.

2.4 High Resolution POSIX Timers (HRT)

Current POSIX API defines two different timer facilities:

BSD timers: `setitimer()` and `getitimer()` functions.

IEEE® 1003.1b REAL-TIME timers: `timer_create()`, `timer_settime()`, `timer_getoverrun()`, etc.

Linux provides the BSD POSIX timers with a timing resolution around 10 ms. The High Resolution Timers (HRT) project provides microsecond resolution with lower overhead following the IEEE 1003.b POSIX API. It is distributed as a patch file that can be downloaded from Sourceforge. Currently, HRT works with Linux kernel 2.4.18 and above.

This patch can provide high-resolution timers with very low overhead because of two main design issues: the use of several timing and interrupt hardware sources (the old 8254, the Pentium internal instruction TSC, and the ACPI timers when available), and a clever data structure to maintain the timers.

The patch provides high-resolution clocks: CLOCK_REAL-TIME_HR and CLOCK_MONOTONIC_HR. The accompanying functions are: `clock_settime()`, `clock_gettime()`, `clock_getres()`. Also, the POSIX timers functions are implemented: `timer_create()`, `timer_delete()`, `timer_settime()`, `timer_gettime()` and `timer_getoverrun()`.

Note that current Freescale Linux BSPs do not, by default, include this high-resolution timer. Contact us about this if your application requires such timer functionality.

2.5 Real-Time Signals

POSIX extended the signals generation and delivery to improve the real-time capabilities. Signals take an important role in real-time as the way to inform the processes of the occurrence of asynchronous events like high-resolution timer expiration, fast inter-process message arrival, asynchronous I/O completion and explicit signal delivery. Linux fully supports the POSIX real-time signals standard with the following features:

> The range of real-time signals supported by Linux is from 32 (SIGRTMIN) to 63 (SIGRTMAX).

> Signals can deliver a small piece of data (an integer or a pointer) to the signal handler (signal-catching function).

> Signals that carry information are delivered in chronological FIFO order.

> It is possible to automatically create a thread in response to a signal.

2.6 POSIX Asynchronous I/O

Asynchronous I/O (AIO) is the POSIX interface to provide high efficiency asynchronous I/O access. The standard way to access I/O devices (files, drivers, sockets, FIFOs, etc.) defined by UNIX is the `read() write()` blocking sequence, where a next file access is performed only when the previous request has been completed. AIO mechanism provides the ability to overlap application processing and I/O operations initiated by the application.

A process can start one or more IO requests to a single file or multiples files and continue its execution. Also, a single system call can start a sequence of I/O operation on one or several files, which reduces the overhead due to context switches.

There are two implementations available in Linux: the one provided at the library level by using non-AIO system calls (included in the glibc/librt since version 2.1); and the kernel implementation first developed by SGI™ called KAIO (until Linux kernel 2.4.0), and now the Linux-AIO, which provides this functionality to newer kernels.

2.7 POSIX Threads

Current Linux implementation of POSIX threads (POSIX 1003.1c) is based on the work done by Xavier Leroy, known as LinuxThreads. LinuxThreads is now integrated in the glibc, and distributed as a part of it. It provides kernel-level threads: threads are created with the `clone()` system call, and all scheduling is done in the kernel. This kind of threading is defined as 1:1, i.e., each thread is mapped to a Linux process.

LinuxThreads implements most of the POSIX API: mutex, condition variables, cancellation, signals, timed calls, etc. The library also provides POSIX semaphores. Mutex do not implement any protocol to prevent priority inversion. Since threads are scheduled by the Linux scheduler, the scheduling policies are the same as in Linux: SCHED_FIFO, SCHED_RR and SCHED_OTHER.

2.8 Quality of Service

Today, Linux v2.4 offers a sophisticated component for bandwidth management called Traffic Control. This component supports methods for classifying, prioritizing and limiting both incoming and outgoing traffic. Therefore, Linux can perform the following tasks: limit bandwidth for certain computers, help to fairly share bandwidth, protect the Internet from abuses, restrict access and do routing based on user id, MAC address, source IP address and so on.

2.8.1 Network Subsystem Overview

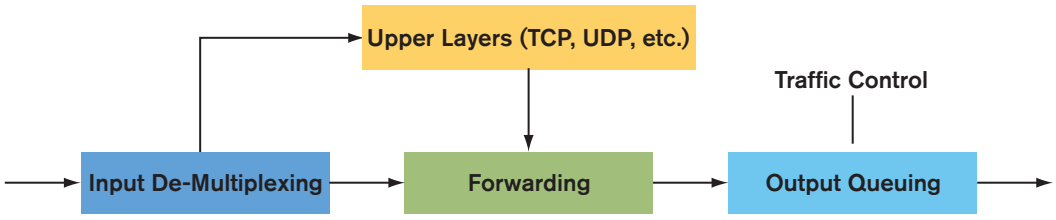

FIGURE 1: LINUX NETWORK SUBSYSTEM

There are four components:

Input De-Multiplexing: Decides if incoming packets are passed to higher layers or are directly forwarded to the network.

Upper Layers: Processes packets, and may also generate new traffic and pass it to the lower layers.

Forwarding: Performs the selection of the output interface, the selection of the next hop, encapsulation, etc.

Output Queueing or Traffic Control: Decides if packets are queued or dropped, decides in which order packets are sent, etc. This is the most important component.

Once the traffic control releases a packet for sending, the network device driver sends it to the network.

2.8.2 Traffic Control Overview

The traffic control component consist of the following elements: queueing disciplines (qdisc), classes (within a queueing discipline), filters and policing.

In this way, queueing discipline provides a method to enqueue a packet. A class is the place where packets are stored and processed in a specific way; afterwards, the qdisc selects the following packet for sending from classes. Filters are used by a qdisc to assign incoming packets to one of its classes. And finally, policing is used to ensure that incoming traffic does not exceed certain bounds.

The following picture illustrates an example of traffic control configuration:

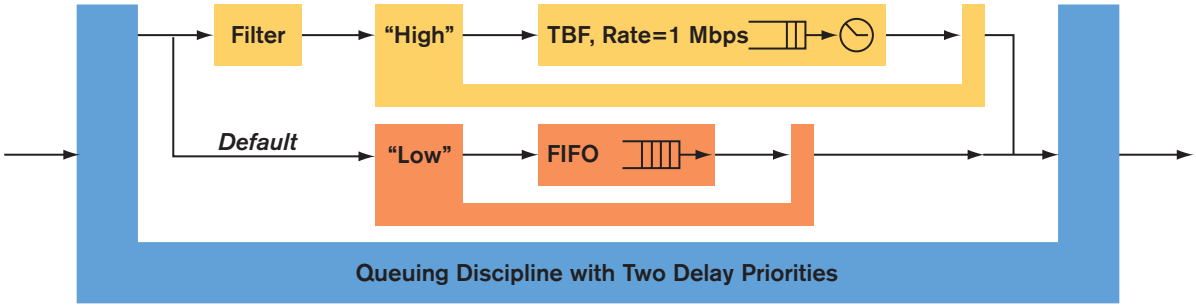

FIGURE 2: TRAFFIC CONTROL CONFIGURATION

This configuration consists of a queuing discipline with two delay priorities, as well as two classes: the higher class contains a token bucket filter discipline that limits the traffic, while the lower class contains a First Input First Output (FIFO) qdisc. Therefore, while the higher class has packets for sending (rate < 1 Mbps), the priority qdisc selects packets from this class. The filter decides which packets are sent to the higher class. Once a priority qdisc selects the following packet for sending, the network driver sends it on the network.

In conclusion, the traffic control layer decides whether the packets are queued or dropped, in which order the packets are sent, and it may delay packet transmission. Moreover, the traffic control elements can be combined in a modular way to support Differentiated Service (DS), Integrated Service (RSVP), ATM and so on.

The following four sections describe the traffic control elements.

Queueing Discipline

Each network interface has a queue discipline attached with it, which controls how packets are enqueued and treated.

A qdisc is a black box, which is able to enqueue packets and dequeue them using its own algorithm. For example, a CBQ qdisc uses a WRR (Weight Round Robin) scheduling to select the following packet for sending on the network.

Moreover, qdisc are divided into two categories:

> Classfull: Qdiscs that may have child qdiscs attached to them.

- PRIO–an n-band strict priority scheduler
- CBQ–Class Based Queue
- CSZ–Clak-Scott-Zhang
- ATM–Asynchronous Transfer Mode
- DSMARK-DSCP–a Diff-Serv Code Point marker
- INGRESS

> Leafs: Qdiscs that do not have child qdiscs attached to them.

- FIFO–a simple FIFO (it is the default qdisc)
- TBF–Token Bucket Filter
- RED–Random Early Detection
- GRED–Generalized Random Early Detection

- TEQL–Traffic Equalizer
- SFQ–Stochastic Fair Queue

Classes

A class is attached to a qdisc. However, queueing disciplines and classes are intimately tied together; the presence of classes and their semantics are fundamental properties of the queueing discipline. There is only one available class. This is the CBQ class. Note that a CBQ may work as queue discipline or class.

Filters

Filters are used to classify packets based on their certain properties (address IP). The supported filters are:

> *rsvp* (use RSVP protocol for classification)

> *u32* (anything in the header may be used for classification)

> *fw* (use firewall rules for classification)

> *route* (use routing table for classification decisions)

> *tcindex* (use the DS field for classification)

Note that the u32 filter is the most advanced filter available, and the tcindex filter is used in DiffServ (differentiation services).

Policing

The goal of policing is to ensure that traffic does not exceed certain bounds. There are four types of policing mechanisms: policing decisions by filter, refusal to enqueue a packet, dropping a packet from an inner queueing discipline and dropping a packet when enqueuing a new one.

Compatibility

Linux was developed around UNIX and POSIX standards. Therefore, its native API is mostly POSIX compatible.

3. Low-Latency Patches for Linux

3.1 Kernel Latency

While scheduling a real-time user-level Linux process, the real-time performance can be affected by bottom halves (BHs) execution, kernel non-preemptable sections and so on. The kernel latency is a quantity used to measure the difference between the theoretical schedule and the actual one.

The kernel latency is defined as follows: Let *T* be a real-time process that requires execution at time *t,* and let t' be the time at which T is actually scheduled; we define the kernel latency experienced by T as $L = t' - t$.

The biggest source of kernel latency is kernel non-preemptable sections (including BHs and Interrupt Service Routines-ISRs). In fact, non-preemptable sections in the kernel can prevent a high priority task from being scheduled for a very long time (up to 100 ms). For example, if interrupts are disabled at time t, task T can only enter the ready queue later when interrupts are re-enabled. In addition, even if T enters the ready queue at the correct time t and has the highest real-time priority in the system, it may still not be scheduled if another task is running in the kernel in a non-preemptable section. In this case, T will be scheduled when the kernel exits the non-preemptable section at time t'.

The length of a kernel non-preemptable section depends on the strategy that the kernel uses to guarantee the consistency of its internal structures, and on the internal organization of the kernel. The standard Linux kernel is based on the classical monolithic structure, in which kernel structures are allowed no more than one execution flow in the kernel at any given time. This is achieved by disabling preemption when an execution flow enters the kernel, i.e., when a system call is invoked or when an interrupt fires. When a system call is invoked, the thread that invokes it enters in the kernel and becomes non-preemptable, returning preemptable when the execution exits from the kernel. When an interrupt fires, a short ISR is invoked with interrupts disabled: the ISR acknowledges the hardware interrupt and schedules a BH for execution. The BH will be executed in a non-preemptive way immediately before returning to user mode; hence, if the ISR interrupts a system call, the BH will be executed only after the system call is completed (system calls can synchronize with ISRs by explicitly disabling interrupts). Summing up, in a standard Linux kernel, the maximum latency is equal to the maximum length of a system call plus the processing time of all the interrupts that fire before returning to user mode. Unfortunately, this value can be as large as 100 ms.

3.2 Open Source Solutions

Two different approaches can be used to reduce the size of kernel non-preemptable sections: the one used by the Low-Latency patches (Ingo Molnar and Andrew Morton) [LowLat], and the one used by the kernel preemptability patch (MontaVista®, TimeSys®) [kpreem, TimeSys].

3.2.1 Low-Latency Linux

This approach corrects the monolithic structure by inserting explicit rescheduling points (that effectively are preemption points) inside the kernel. In this approach, when a thread is executing inside the kernel it can explicitly decide to yield the CPU to some other thread. In this way, the size of non-preemptable sections is reduced, thus decreasing the latency. In a low-latency kernel, the consistency of kernel data is enforced by using cooperative scheduling (instead of non-preemptive scheduling) when the execution flow enters the kernel.

This approach is also used by some real-time versions of Linux, such as RED-Linux. In a low-latency kernel, the maximum latency decreases to the maximum time between two rescheduling points. Since the low-latency patch has been carefully hand-tuned for quite a long time, it performs surprisingly well.

3.2.2 Preemptable Linux

The preemptable approach, used in most real-time systems, removes the constraint of a single execution flow inside the kernel. Thus, it is not necessary to disable preemption when an execution flow enters the kernel. To support full kernel preemptability, kernel data must be explicitly protected using mutexes or spinlocks. The Linux preemptable kernel patch, maintained by Robert Love and sponsored by MontaVista®, uses this approach and makes the kernel fully preemptable. Kernel preemption is disabled only when a spinlock is held.

A similar approach is used by TimeSys® that uses mutexes instead of spinlocks, and provide priority inheritance. While the Robert Love patch disables preemption when a spinlock is acquired, the TimeSys® one is based on blocking synchronization.

In a preemptable kernel, the maximum latency is determined by the maximum amount of time for which a spinlock is held inside the kernel. Again, it is important to note that BHs are serialized using a spinlock, thus, they contribute to the latency.

An additional patch (lock-breaking) merges some of the low-latency rescheduling points into the preemptable kernel, for decreasing the amount of time for which spinlocks are held.

3.3 Evaluation

3.3.1 Performance Comparison

The OCERA team measured the latency for the standard, Low Latency; preemptable and preemptable with lock-breaking kernels while running different loads in background. All the experiments were performed using the Latency Benchmark tool downloadable from [FT]. The results are shown in the figures below when the following system loads were run at the indicated time:

- “memory” at 1000
- “caps on” at 7000
- “caps off” at 8000
- “chvt 3” at 9000
- “chvt 2” at 10000
- “i/o” at 11000
- “proc read” at 17000
- “fork” at 20000

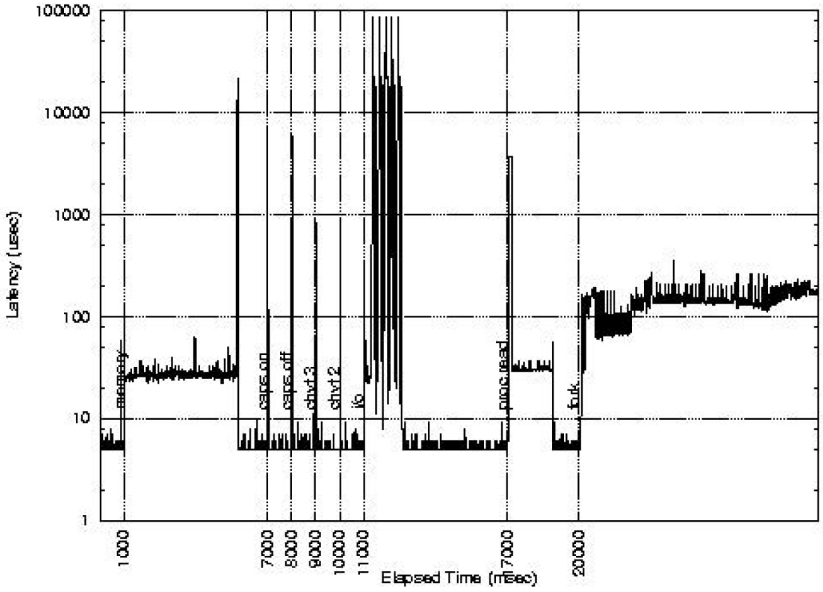

FIGURE 3: LATENCY IN THE STANDARD KERNEL

From the first figure, it is possible to see that standard Linux exhibits high latencies at the end of the memory stress (a program that reads and writes a large array in memory), during the I/O stress (a program that reads and writes large amount of data on a file), when accessing the /proc file system, and when switching the caps/lock LED. The large latency at the end of the memory stress is due to the `munmap()` system call. Comparing the figures, it is possible to see that the low latency kernel solves all the problems except the /proc file system access and the caps/lock switch. On the other hand, the preemptable kernel eliminates the large latency in the /proc fs access, but does not solve the problem with the memory stress, and is not as effective as the low latency kernel in reducing the latency during the I/O stress. Finally, the lock-breaking preemptable kernel seems to provide good real-time performance, but still has some problem during the I/O stress.

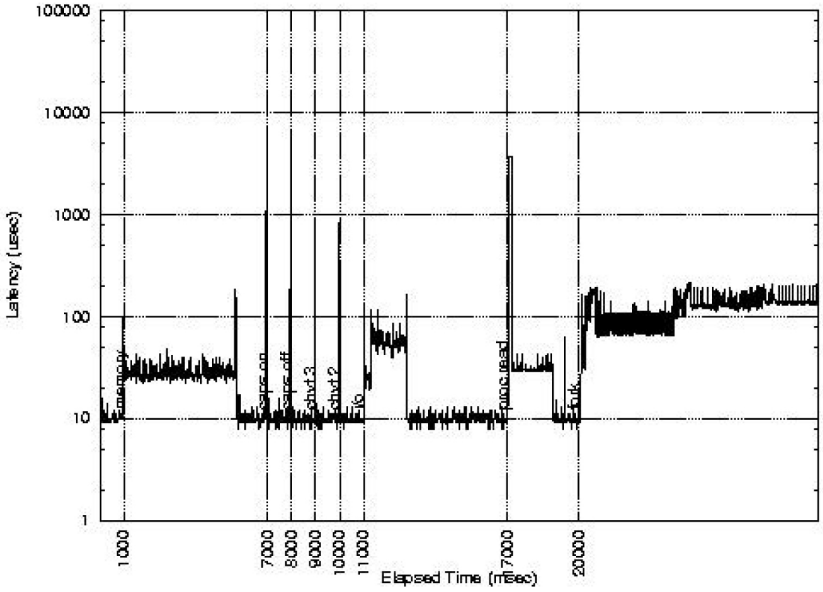

FIGURE 4: LATENCY IN THE LOW LATENCY KERNEL

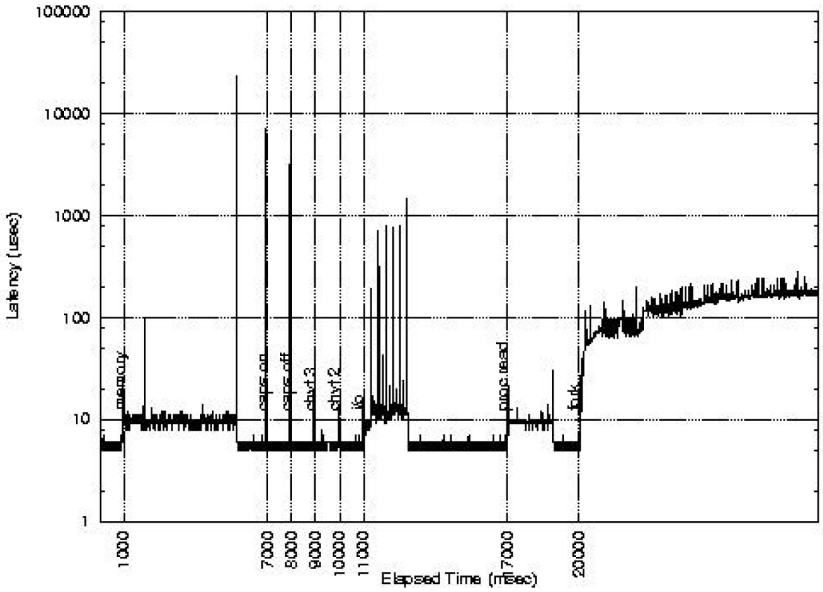

FIGURE 5: LATENCY IN THE PREEMPTABLE KERNEL

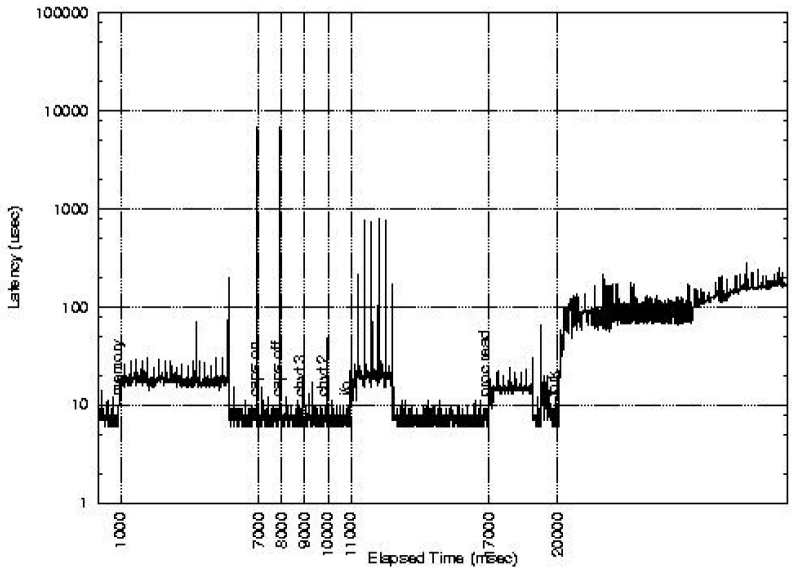

FIGURE 6: LATENCY IN THE PREEMPTABLE LOCK-BREAKING KERNEL

By repeating similar experiments for longer amounts of time, we verified that the low-latency kernel is characterized by larger average latencies respect to the preemptable and lock-breaking preemptable kernels, but reduces the worst case latencies. In other words, the tail of the probability distribution function is shorter for a low-latency kernel. As an example, the PDFs of the latency during the I/O stress are reported in the following figures (note that this new experiments were performed on a computer that is faster than the one used for the previous experiments).

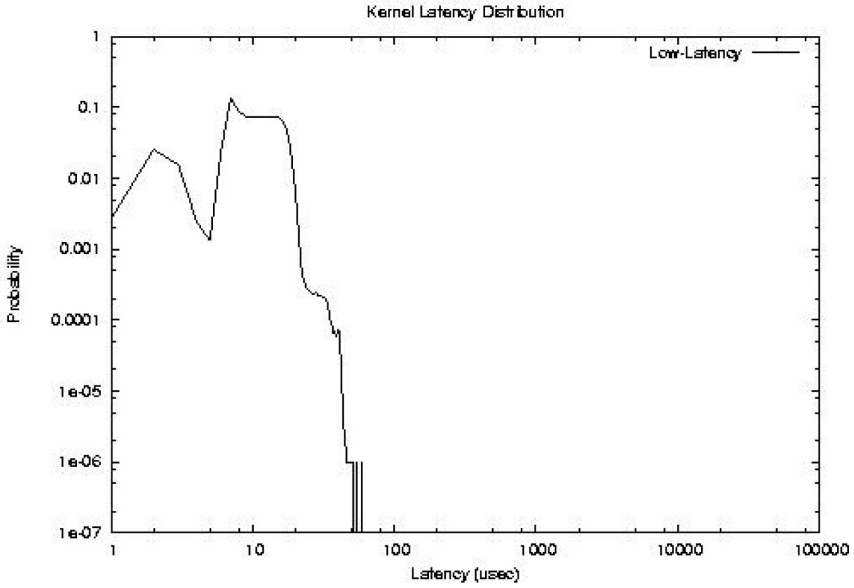

FIGURE 7: PDF OF THE LATENCY IN THE LOW-LATENCY KERNEL

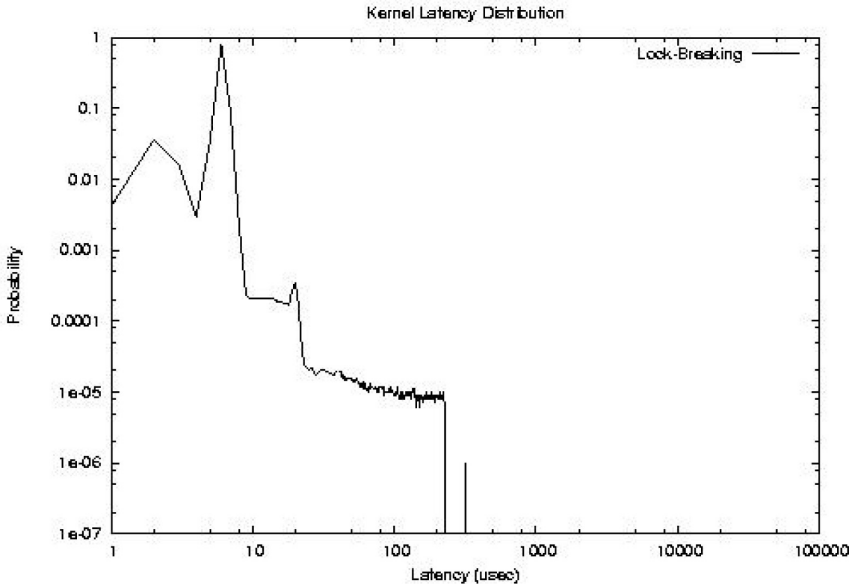

FIGURE 8: PDF OF THE LATENCY IN THE LOCK-BREAKING KERNEL

The previous figures indicate the low-latency patch provides better real-time performance. This is probably due to the fact that the Linux kernel still contains “big spinlocks’’ that are held for long amounts of time, and that the lock-breaking patch does not break properly. A classical example is the spinlock used to serialize BHs. However, in the future Linux developers will likely decrease the size of kernel critical sections (to improve SMP performance). Hence, it is reasonable to think that the latency of the preemptable and lock-breaking preemptable kernel will decrease.

3.3.2 Implementation Comparison

Although the low latency and preemptable kernel patches offer effectively identical performance, their specific architectures vary the implementation of device drivers and the migration of this patch across kernel versions.

Device Driver Implementation

The differing implementation technique of each kernel patch affects the manner in which device drivers should be coded.

Because the low latency patch effectively adds preemption points within the kernel source code, it requires no special treatment of device driver code. This can be especially important in cases where third party binary device drivers (for which you can not review source code) are planned.

However, because the preemptable kernel patch is based on the code for Symmetric Multiprocessing (SMP), it is necessary to ensure that all device drivers have been written to perform proper locking. This entails careful attention to driver spinlocks, semaphores and mutexes. Thus, developers should take more time investigating and validating the coding of each of the system's device drivers. (See: http://lwn.net/Articles/22912/.) In cases where binary device drivers are provided for inclusion within your product, the developer should test the impact on real-time performance carefully.

Patch Migration and Portability

As with device drivers, the differing implementation technique of each kernel patch affects the portability of the patch from one kernel version to the next (e.g., v2.4.19 to v2.4.20).

Because the low latency patch requires that explicit preemption points be placed within the kernel code, it requires a moderate amount of maintenance to forward port into new kernel versions. This is generally performed using a tool called "timepegs" (See: http://www.zip.com.au/~akpm/linux/index.html#timepegs), which is a tool for precisely measuring the time interval between the execution of arbitrary pieces of code in the kernel.

On the other hand, the maintenance/portability issues of the preemptable kernel are lower than the low-latency kernel because the preemptable technique relies strictly on the SMP mechanisms that remain relatively constant between kernel versions.

4. Summary

Linux is rapidly evolving to include more and more features that have explicit value to the embedded developer.

Key enhancements include two very high-performance real-time enhancement patches to the Linux kernel, which perform identically for all practical purposes. As a result of this performance, the developer's choice regarding which should be implemented can be based on independent criteria including the ability to update/modify existing device drivers for compatibility with the real-time implementation, and the availability of each real-time technique for the selected architecture and kernel version.

As of the date of this paper, all of Freescale new or in-process Linux BSPs include one of these real-time techniques, chosen for maturity and CPU support as appropriate to the individual development boards. Freescale will also be updating our existing Linux BSPs to include these enhancements with a priority based on market demand. Contact us for specific information related to your board of interest.

5. Bibliography

This paper uses the overview, test and analysis material as it relates to real-time Linux from [OCERA] Ismael Ripoll, Pavel Pisa, Luca Abeni, Paolo Gai, Agnes Lanusse, Sergio Saez, Bruno Privat , "WP1-RTOS State of the Art Analysis" Open Components for Embedded Real-Tme Applications.

[LowLat] Andrew Morton, *Linux Scheduling Latency (http://www.zip.com.au/~akpm/Linux/schedlat.html).*

[kpreem] Robert Love, *The Linux Kernel Preemption Project (http://kpreempt.sourceforge.net).*

[TimeSys] TimeSys, *TimeSys Linux.*

[FT] Systems Software Lab—Oregon Graduate Institute, The *Firm Timers Home Page (http://www.cse.ogi.edu/~luca/firm.html).*

6. Appendix A: README File from the FirmTimers Kernel Latency Test Suite

This directory contains three programs:

-measure_latency: This program tries to sleep for a specified amount of time, and measures the difference between the time when it is supposed to wake up and the time when it effectively gets scheduled (we define this value as “kernel latency”).

-generate_load: This program can be used to generate a lot of kernel activities, in order to stress the kernel during the latency test.

timetest: Implements the functionality of both the two programs above.

Note that these programs should not be used directly but should be invoked through the driver in the benchmark directory.

There are three main factors that can influence the kernel latency, as defined above and measured by measure_latency:

1) The scheduling algorithm. If a process is not assigned a real-time priority, it may not be scheduled as soon as it wakes up. In order to solve this problem, measure_latency can change its priority to real-time (see the -r switch).

2) The time resolution of the kernel. For example, the standard Linux kernel wakes up processes only at tick boundaries, and a tick is 10 ms. Hence, a process sleeping for a time that is not multiple of 10 ms will experience a big latency. This problem can be solved by using high resolution timers, such as the OGI firm timers or similar.

3) The non-preemptability of the kernel. Even if the kernel provides high resolution timers, a process can still experience high latencies (we measured up to 100 ms on Linux) when the kernel executes long non-preemptable sections (the generate_load program can be used to reproduce this situation). The Linux kernel preemptability patch (http://kpreempt.sourceforge.net) can solve this problem.

The measure_latency program writes to the standard output some of the time when the process begins to sleep, when it asks to be woken up, and when it really wakes up, plus the latency (fifth column).

In order to investigate the three problems highlighted above in all the possible situations, the programs provide numerous switches:

measure_latency:

-r: run the program with real-time priority (must be root).

-s <delay>: use select() for sleeping <delay> microseconds.

Note that if the kernel does not provide high resolution timers, only multiples of the system tick make sense for <delay>. For example, using the standard Linux kernel you should set <delay> to 10000, or 20000, or 30000 and so on.

-n <delay>: as -s, but use sleep() for sleeping.

-a <delay>: as -s, but use interval timers.

-k: use the RTC for sleeping (must be root).

-x: when -a is used, set the interval timer to periodic.

-C: use the timestamp counter instead of `gettimeofday()` for getting the time.

-l <loops>: iterate <loops> times, then exit.

-i <increment>: increase <delay> by <increment> at each loop.

-w <period>: print the latency only every <period> samples.

The maximum, average, and minimum latency in the period are printed, which is very useful to reduce the amount of points in a latency graph.

generate_load:

-r: run the program with real-time priority (must be root).

-c: generate CPU load (no system calls ---> no kernel latency).

-f <size>: generate file system load (`read()` and `write()` on a file of <size> MegaBytes).

-p <num>: forks <num> processes.

-m <size>: perform a memory stress test, reading and writing data on a buffer of <size> MegaBytes. This is generally critical, because it generates a lot of page faults.

-l <loops>: iterate <loops> times

-z <blocksize>: during the filesystem stress, write or read <blocksize> bytes per time.

7. Appendix B: GNU Free Documentation License

7.1 Preamble

The purpose of this License is to make a manual, textbook or other written document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

7.2 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

7.3 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

7.4 Copying In Quantity

If you publish printed copies of the Document numbering more than 100 and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

7.5 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

Use in the Title Page (and on the covers, if any) a title distinct from that of the Document and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

State on the Title page the name of the publisher of the Modified Version, as the publisher.

Preserve all the copyright notices of the Document.

Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

Include an unaltered copy of this License.

Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

Preserve the network location, if any, given in the Document for public access to a transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles.

Section numbers or the equivalent are not considered part of the section titles.

Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.

Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties-for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

7.6 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise, combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

7.7 Collections Of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7.8 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise, they must appear on covers around the whole aggregate.

7.9 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

7.10 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

7.11 Future Revisions Of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

7.12 How To Use This License For Your Documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

How to Reach Us:

Home Page:
www.freescale.com

e-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor

Technical Information Center, CH370

1300 N. Alma School Road

Chandler, Arizona 85224

1-800-521-6274

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 3 5437 9125

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

1-800-441-2447

303-675-2140

Fax: 303-675-2150

LDCForFreescaleSemiconductor

@hibbertgroup.com

Document Number: SWVERIFICATIONWP
Rev. 0
11/2005

freescale™
semiconductor