EVALUATION OF REAL TIME PERFORMANCE OF EMBEDDED LINUX

Truong Quang Vinh, Vo Ky Chau

Department of Electronic Engineering, HCMC University of Technology

---

ABSTRACT

Nowadays, embedded systems with Linux OS are commonly applied in automotive industry. Some of applications require strict time response, and others need to be exactly scheduled to execute a period task. All of these are called time-sensitive applications. Measuring and evaluating time parameters of an embedded system for time-sensitive applications is very necessary for developers to guarantee that it works functionally. For this purpose, signal stimulating and signal analyzing hardware are required. In this paper, modular signal generating and signal analyzing means will be designed. The prototype of the evaluation system will be implemented to test and verify an embedded system.

Improving the performance of the system is the next part of this report. The research focuses deeper on the possibility of turning Linux into a real-time operating system. Particularly, it investigates available real-time solutions for Linux, but also looks into the soft variants for completeness. A number of performance tests are conducted during the improvement to make sure that the implementation meets the demands of a real-time operating system.

1 INTRODUCTION

1.1 Embedded Linux OS

Since Linux was initially developed by Linus Torvalds in 1991 as an operating system for IBM compatible personal computers based on the Intel 80386 microprocessor, Linux today is available for many embedded systems with others various microprocessors. Over the years, Linus and many coordinating developers around the world have worked to make Linux available on other architectures, including Intel Xscale, Alpha, SPARC, Motorola MC680x0, PowerPC, and IBM System/390 [1].

Linux is a versatile and cost effective operating system for embedded systems. It can be embedded in a surprisingly small system to handle simple tasks and scaled up to handle more complex tasks. Linux can run on most microprocessors with a wide range of peripherals and has a ready inventory of off the shelf applications. Development cycles are shortened through the use of mature tools, open source code, substantial documentation and available support services. Due to all those advantages, Linux is a good choice for embedded systems [2].

1.2 Real-time embedded Linux OS

Since the evaluation system needs to be process all analyzing signals at strict time, we need a real-time operating system for the processor. Linux, with the real time extension, provides this kind of precise control over interrupt handling. Essentially, there is an interrupt manager that handles all interrupts. It does a good job of making sure that critical interrupts get run when needed. The hardness of this approach depends mostly on the CPU interrupt structure and context switch hardware support. This approach is sufficient for a large range of real time requirements. Even without the real time extensions, Linux is pretty good at

keeping up with multiple streams of events [6] [7].

1.3 Evaluation methods

There are varying levels of real-time systems evaluation. The most prevalent ones are the use of analytical models, the simulation of scheduling algorithms, and hardware simulation. Analytical models are mathematical theorems and proofs that model the worst time performance of one or more of the aspects of real-time systems, and by changing certain inputs to these theorems, an optimum performance can be proven. Simulation takes the analytical models one step further in creating a simulation using scheduling theory to experiment with behavior of real-time systems. Finally, hardware tests take the theorems that were postulated by the analytical model and have been simulated through the use of scheduling algorithms, and run tests on the actual hardware to discover any behavior that was not determined through either of the other two methods.

This paper focuses on experimental test for actual hardware. The test is for the purpose of evaluating real-time scheduling capability of the embedded operating system. The target system is Triton board using PXA255 processor with embedded Linux. In experiments, some test programs are run on the target system. Then a testing module logs all real-time data from it. The data will be sent to the host PC to analyze. The detail of experiments will be presented in the Section 2.

1.4 Metrics of Characterization

Two of the most common metrics used to characterize real-time systems are jitter and response time. These metrics will be measured and analyzed carefully on the test.

Jitter represents the minimum and maximum time separating successive iterations of periodic tasks. If this inter-arrival time is greater than the period of the task, it means that the task is running late, and this will show up as a positive jitter value. If that inter-arrival time is less than the period of the task, it means that the task is running early, and that will show up as a negative jitter value. This variation is caused by interference with interrupt and other tasks. This parameter is very important for real-time applications such as mechanic control, medicine instrument, etc.

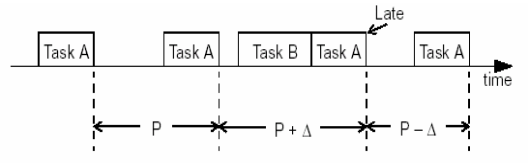


Figure 1: Real-Time Jitter.

The execution of Task B pushes back the 3rd execution of Task A, causing the task completion times to deviate from their ideally periodic nature.

Interrupt response time (interrupt latency) is the time that it takes for a real-time system to respond to an external interrupt and represents the reaction time of the system to an unscheduled event while under load. In other word, interrupt response time is the amount of time between when an interrupt is generated (internally or by an external device) and when an installed interrupt handler starts to execute. When the system is in an idle state this time is very short, but will be longer when for example other interrupts are processed.

Interrupt response time is a very important measurement in a real-time system. It affects many other performance aspects such as scheduling precision and interrupts task latency. The worst case interrupt latency yields, for example, a lower boundary for the worst case interrupt task latency. The most interesting tests are those with load as the worst latencies show up under different levels of load.

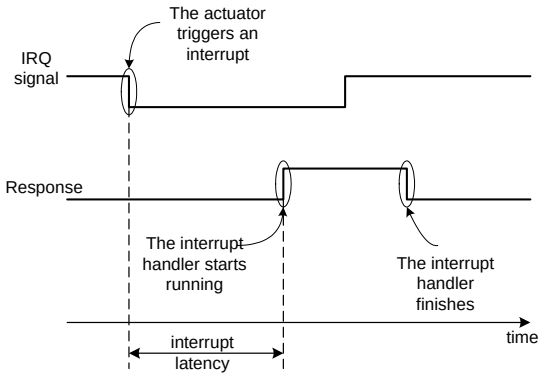


Figure 2: Interrupt Latency

2 EXPERIMENTS

In this section, two tests will be proposed. They are performed on Triton Starter Kit.

For the first one, a mark with an output bit of the GPIO port is generated, when a fixed period of time has elapsed, using as time generator the internal clock of the PC. The intention of this test is to measure the variations that come up when Linux is performing synchronic tasks.

For the second test, the time that Linux take to response to an external interrupt is measured. The intention of this test is to characterize the ability of Linux in handling asynchronic tasks.

All tests are performed under different kinds of load for the system: no load, background load, hard disk load, network load, calculation load, full load.

2.1 Test 1: Jitter

A 20 millisecond pulse-width square wave is generated in an output bit of the GPIO port. The time between the leading edge and the trailing edge of the square wave measured for single periods. This experiment will show the jitter of a periodic task. The period time is realized by using *usleep* function to sleep in a specified mount of time. The test result is the time that it actually slept.

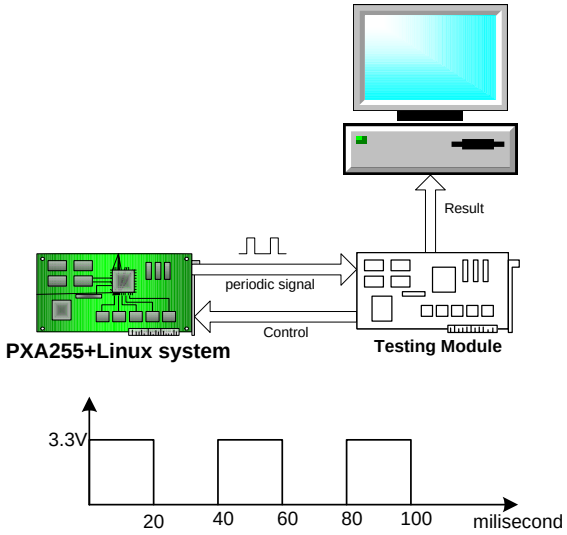
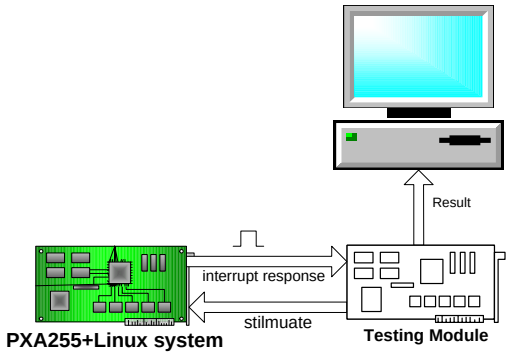


Figure 3: Hardware setup for test 1

2.2 Test 2: Interrupt Latency

The testing program responds by turning on a digital output when an interrupt occurs. When the system receives a stimulate signal for interrupt at GPIO10, a pulse is generated in an output bit of the GPIO11. Any interrupt occurring while the output is high should be ignored.

The elapsed time between the interrupt edge and the leading edge of the response pulse will be statistically measured, and shown in the histogram.



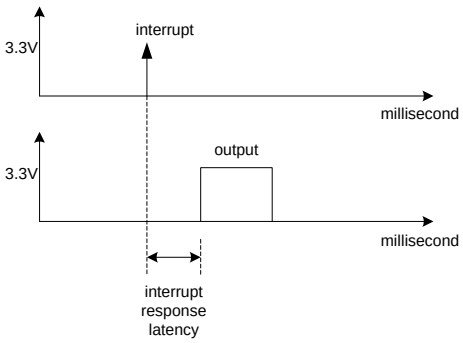


Figure 4: Hardware setup for test 2

3 IMPLEMENTATION OF TESTING SYSTEM

3.1 Jitter Test Module

Jitter testing module is designed based on FPGA. The objectives of this module are to receive pulse signal from target system, count the time interval of this signal and transfer this value to PC through serial port. The accuracy degree of this module is 1 microsecond.

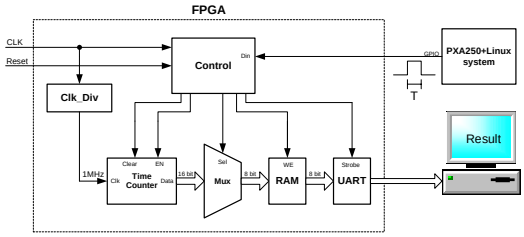


Figure 5: Block diagram of jitter test module.

3.2 Interrupt Latency Test Module

Interrupt test module is design based on FPGA. The objectives of this module are to generate an interrupt requests and receive response pulse from target system. The time interval between the rising edge of the interrupt pulse and the leading edge of the response pulse will be statistically measured, and be sent to PC to draw in the histogram.

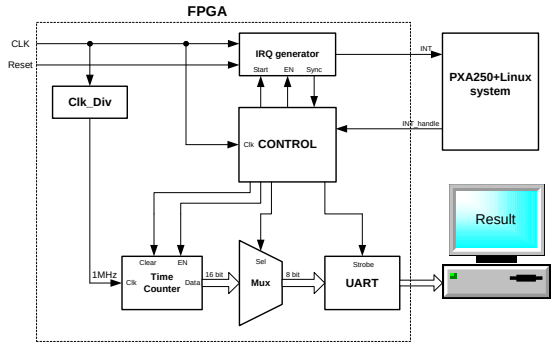


Figure 6: Block diagram of Interrupt Latency Test Module

4 RESULT AND ANALYSIS

4.1 Jitter

All of the graphs shown are probability density of the jitter values. The highest value of occur number presents the desired period of 20 milliseconds.

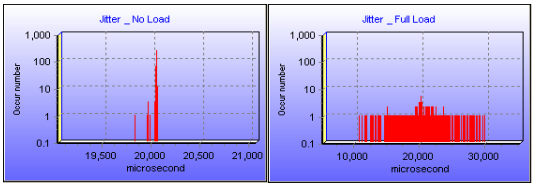


Figure 7: Histogram for Jitter of the target system

Table 1: Statistic data of jitter using IRQ

Type of Load	Mean (us)	Max (us)	Min (us)	Standard Deviation (us)
No load	38.56	184	22	2.33
Background	41.47	647	14	10.37
Calculation	43.13	729	14	11.87
Disk	229.06	6889	2	322.43
Network	109.26	765	0	95.9
Full	1696.15	9778	0	1515.91

When the target system has no load, the jitter is low. But it increases numerously when some other tasks are running at the same time. In the worst case, when the system runs with full load, maximum jitter is about 9.8 milliseconds. This value of jitter cannot be allowed in a time-sensitive application.

4.2 Interrupt

The histograms below describe probability density of interrupt latency.

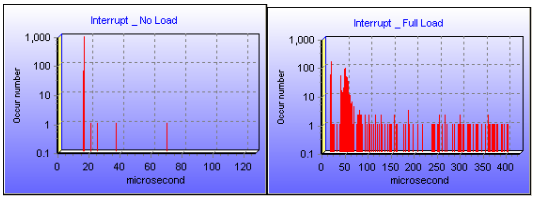


Figure 8: Histogram for Interrupt Latency of the target system

Table 2: Statistic data of interrupt latency using IRQ

Type of Load	Mean (us)	Max (us)	Min (us)	Standard Deviation (us)
No load	16.02	69	15	.17
Background	15.94	69	15	.25
Calculation	15.78	77	15	.73
Disk	22.93	380	14	10.88
Network	29.99	329	21	6.97
Full	58.02	400	14	33.43

According to the test result, it is realized that the level of system load affects interrupt latency considerably. This comes from the fact that the system disables all interrupts a short time when receiving new interrupts. If a new interrupt is generated when the other interrupts are disabled, it will not be handled by the system until the interrupts are enabled again.

5 IMPROVEMENT

Some improvements of Linux kernel on target system are proposed in order to gain a better real-time performance of the embedded system.

5.1 Timer interrupts on PXA255 processor

Standard Linux kernel has already the timers. They are triggered by a periodic tick interrupt which has a period of 10 milliseconds. This value can be reduced by using a higher resolution timer. This approach depends on the hardware that Linux is ported to. The objective to improve resolution of timers is to make an extensive driver for timers.

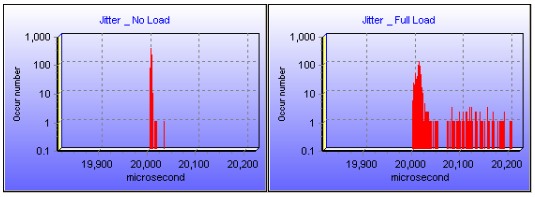


Figure 9: Histogram for Jitter using OS timer interrupt

Table 3: Statistic data of jitter of the periodic task using OS timer interrupt

Type of Load	Mean (us)	Max (us)	Min (us)	Standard Deviation (us)
No load	5.03	31	0	0.91
Background	4.11	32	0	0.98
Calculation	4.8	33	0	1.24
Disk	6.16	111	0	3.46
Network	15.6	195	3	6.35
Full	21.23	289	0	18.4

5.2 FIQ on Intel PXA255 processor

FIQ is an advanced feature of exception of ARM architecture. The FIQ exception is generated externally by asserting the FIQ input on the processor. FIQ is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such application, therefore minimizing the overhead of context switching. FIQs have higher priority than IRQ in two ways. First, FIQ are serviced first when multiple interrupts occur. Second, servicing a FIQ causes IRQs to be disabled, preventing them from being serviced until after the FIQ handler has re-enabled them. This is usually done by restoring the CPSR from the SPSR at the end of the handler. [17]

The jitter test with FIQ mechanism is done on the target system. The test result is shown in the histograms below.

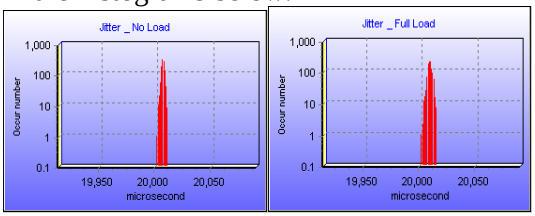


Figure 10: Histogram for Jitter of the target system

Table 4: Statistic data of jitter of the periodic task using FIQ

Type of Load	Mean (us)	Max (us)	Min (us)	Standard Deviation (us)
No load	6.29	10	1	1.1
Background	5.55	11	1	1.13
Calculation	7.52	12	2	1.21
Disk	7.09	13	2	1.27
Network	6.64	12	1	1.2
Full	8.55	14	1	1.59

To demonstrate an external interrupt using FIQ, GPIO FIQ is implemented. GPIO58 is chosen for interrupt source. When the processor detects a rising edge on GPIO58, the FIQ handler will be executed. The handle function generates a pulse on GPIO11, so that the Interrupt Test System can measure interrupt latency of the FIQ interrupt. The value of FIQ latency is the time interval between the rising edge of GPIO58 and the rising edge of GPIO11 output.

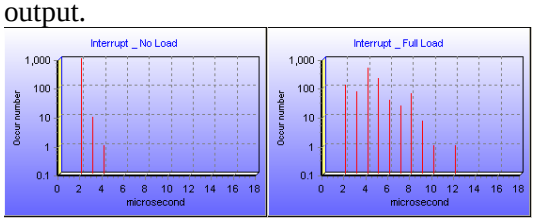


Figure 11: Histograms of Interrupt latency using FIQ

Table 5: Statistic data of Interrupt latency using FIQ

Type of Load	Mean (us)	Max (us)	Min (us)	Standard Deviation (us)
No load	2.01	4	2	0.02
Background	2.02	4	2	0.03
Calculation	2.01	4	2	0.02
Disk	2.62	10	2	0.93
Network	2.14	4	2	0.25
Full	4.31	12	2	1.04

6 CONCLUSION

This paper has presented a method of using FPGA-based emulation to evaluate the real-time performance of an embedded system with Linux OS. A number of tests were designed and implemented in order to measure and evaluate two parameters of real-time behavior, jitter and interrupt latency.

The result of jitter test shows that the load of system affects remarkably the precision of scheduler. The maximum value of jitter goes up from 184 microseconds to approximate 9.8 milliseconds when the load increases.

The maximum value of interrupt latency in full load case goes up 400 microseconds. The statistic data from the test show that the latency is affected much by other processes with interrupt request for peripheral devices.

Throughout this evaluation, users can estimate how well the time-sensitive applications run on the target system. The maximum values of latency are investigated and analyzed to ensure time constrains of applications.

Some solutions have been proposed to reduce the latency. By utilizing the advanced features of ARM architecture, real-time performance of the embedded system can be improved significantly. Two these features are OS timer and Fast Interrupt Request (FIQ).

Result of timer test shows that jitter can be reduced remarkably by OS timer. The maximum value of jitter in worst case is about 300 microseconds. It is good performance for applications that do not require very strict real-time.

The next feature that can improve the real-time performance is FIQ. The best performance of FIQ is proved by the test. The interrupt latency measures 12 microseconds. It is near the hardware limit, and is a very good result for hard real-time applications.

REFERENCES

[1] John Lombardo, 2001, “Embedded Linux”, New Riders.

[2] Blue Mug, Inc., December 2002, “Embedded Linux Survey”, www.bluemug.com

[3] Intel PXA255 and PXA210 application Processors, 2003, “Developer’s Manual”, http://www.intel.com
[4] Daniel P. Bovet and Marco Cesati, 2000, “Understanding the Linux Kernel”, O’Reilly.
[5] Tigran Aivazian, 2001, “Linux Kernel Internal”, http://www.moses.uklinux.net/patches/lki.sgml
[6] Bill Weinberg, MontaVista Software Inc., 2001, “Embedded Linux – Ready for real-time”, white paper, http://www.mvista.com
[7] Bill Weinberg, MontaVista Software Inc., 2001, “Moving from a Proprietary RTOS To Embedded Linux”, white paper, http://www.mvista.com
[8] Blue Mug Inc., 2002, “Embedded Linux Performance Study”, http://www.bluemug.com
[9] Marco A. Sanvido, Vaclav Cechticky Walter Schaufelberger, “Testing embedded control systems using hardware-in-the-loop simulation and temporal logic”, 15th Triennial World Congress, Barcelona, Spain
[10] Andrei V. Gurtov, 1999, “Technical Issues of Real-Time Network Simulation in Linux”, FDPW99 Volume 2, University of Helsinki.
[11] Steve Babin, March 2003, “Diagnostics for design validation”, article at http://www.embedded.com/story/OEG20030325S0033
[12] Luca Abeni, Ashvin Goel, Charles Krasic, Jim Snow, Jonathan Walpole, 2002, “A measurement-base analysis of real-time performance of Linux”, scientific report at Oregon Graduate Institute, Portland.
[13] Nicholas Mc Guire, 2001, “MiniRTL - Hard Real Time Linux for Embedded Systems”, FSMLab.
[14] Clark Williams, Red Hat, Inc., March 2002, “Linux Scheduler Latency”, www.linuxdevices.com/articles/AT8906594941.html
[15] Neil Matthew and Richard Stones, 1997, “Linux Programming”, Wrox Press.
[16] Alessandro Rubini and Jonatthan Corbet, “Linux Device Driver”, O’Reilly.
[17] ARM Limited, 1999 , “ARM Architecture Reference Manual”, www.arm.com
[18] Douglas L. Perry, 1999, “VHDL”, McGraw Hill.