Real Time on Moblin/Linux

Yannick MEYER[1] (main author), Abdelillah YMLAHI OUAZZANI[1]

1: ESG Automotive France, Centre Paris Pleyel – 153, Boulevard Anatole France - 93521 - Saint-Denis Cedex

Abstract: The automotive manufacturers are more than ever facing many challenges. With the tremendous involvement of electronics in the achievement of end-customers needs, automotive suppliers must accommodate themselves to the constantly changing features of carmakers. Infotainment is one of the most challenging fields, where the carmakers and suppliers realized that the use of a dedicated platform for each new car is no longer possible. This is why the GENIVI alliance was founded on March 2nd, 2009 with the goal of establishing an open platform for the automotive in-vehicle infotainment industry.

GENIVI's strategy is to use Moblin as the baseline code within its reference implementation. Moblin is a Linux-based distribution designed for mobile devices and can perfectly be used in embedded devices such as those targeted by the infotainment industry.

In this paper we experiment the real-time performances of Moblin as a standalone operating system and show how those performances can be improved by means of adding some customized real-time extensions. The main contribution of this work is to outline how those extensions are in fact necessary to comply with the real-time constraints required by automotive hosts on which infotainment applications are embedded.

Keywords: Infotainment, Genivi, Moblin, Linux real-time, latency measurements

1. Introduction

GENIVI is self-described as a non-profit industry alliance committed to create and promote an In-Vehicle Infotainment (IVI) reference platform. The platform consists of Linux-based core services, middleware, and open application layer interfaces. It establishes a base upon which carmakers and suppliers can add their variety of products and services.

At ESG, we anticipated at early stage the development of an automotive infotainment platform similar to what GENIVI alliance would expect. We invested ourselves in technical training and purchase of a "GENIVI-like" platform and started evaluating its software and hardware performances. After few weeks of evaluations, we explicitly expressed our motivation to actively participate in the development of automotive Infotainment architectures within the GENIVI alliance and we joined it as associate member on January 2010.

Moblin is an open source operating system, acting as an independent distribution for the first GENIVI open source implementation. The combination of the Moblin and GENIVI codes will be used to provide carmakers and suppliers with an automotive infotainment reference platform mixing the best from both consumer and automotive worlds.

Moblin is Intel's open source initiative project created to develop software for the next generation of mobile devices including Netbooks, Mobile Internet Devices, and In-vehicle infotainment systems based on Intel’s Atom target processor.

In automotive field, any embedded software system must deliver both speed and accuracy. Speed is needed to run large bodies of software ECUs, including real-time operating systems and network protocol stacks. Accuracy, reflected by the latency time, is needed to ensure correct and predictable behavior in the vehicle. In fact, deterministic latency time is one of the core requirements of “hard" real-time constraints.

In contrast to those "hard" constraints, Linux and mostly all distributions based on it satisfy only "soft" real-time requirements. For example, Linux can provide speed to manage tasks priorities but with unpredictable latency time, varying from less than one millisecond in most cases up to several milliseconds in some others.

Missing a deadline in “hard” real-time requirements implies a catastrophic behavior (like activating a security airbag too late in time). In “soft” real-time requirements such a missing causes system failure but doesn’t alter the general behavior (like losing a video frame or gps connection).

In this paper, we report our experience in evaluating the real-time capabilities of Moblin v.2.0b3 on a generic infotainment platform composed of the following peripherals:

- Processor: the Intel Atom Z530 (1.6 GHz)
- Memory : 1GB RAM DDR2, 2GB Flash
- Display : VGA touch screen
- Audio : HD-Audio
- Connectivity: Ethernet, GPS antenna, CAN
- Misc: 8 x USB 2.0, RS232/422/485, GPIO

In addition to that, we used an external board called "Measurement Board" to stimulate the "Moblin platform" and measure its reaction time to the input stimulus.

We tested first the real-time behavior of Moblin-Linux APIs to point out its limitations. Then we added the real-time extensions RTAI and XENOMAI (Cf. §.5 and §.6), once at a time, and evaluated again the new behavior.

The rest of the paper is organized as follows. Section 2 describes the testing methods. Section 3 is dedicated to the real-time capabilities of Moblin. Section 4 explains how the real-time extensions can be added to Linux. Section 5 and 6 discuss the real-time performances of Moblin with the real-time extensions RTAI and XENOMAI respectively, and Section 7 concludes.

2. Testing methods

2.1 Preparation

We reserved the first activity of this project to the building of the Moblin image from the kernel source package **kernel-2.6.29.4-6.1.moblin2.src.rpm** available online at Moblin's repository [3]. Then, with some few adjustments, we were able to drive the hardware platform.

The second activity consisted in setting up the test bench by connecting the "Moblin platform" to the "Measurement Board". Figure 1 illustrates how we connected the boards; the hardware connection was ensured by a serial RS232 cable for data exchange and a double wired cable to control a GPIO pin.

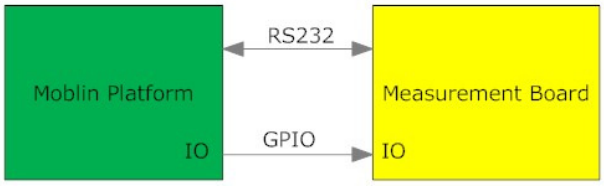

Figure 1: Hardware connections setup

The reason behind choosing the serial communication device rather than any other way (CAN, Timer…) is its hardware availability on the measurement board.

The evaluation of the real-time capabilities of the "Moblin platform" is composed of two sets of tests:

- Measuring the latency time the "Moblin platform" takes before responding to an external event.

- Measuring the scheduling time of a periodic task implemented in the "Moblin platform".

Those two tests cover mostly all cases in which embedded software is to be used. In the next sub-sections we explain the tests design and setup.

2.2 Measurement of the latency time in response to an external event

The main purpose of this test is to measure the minimal, average and maximal latency time consumed by the "Moblin Platform" before it reacts to an external event. Figure 2 shows how we designed this test.

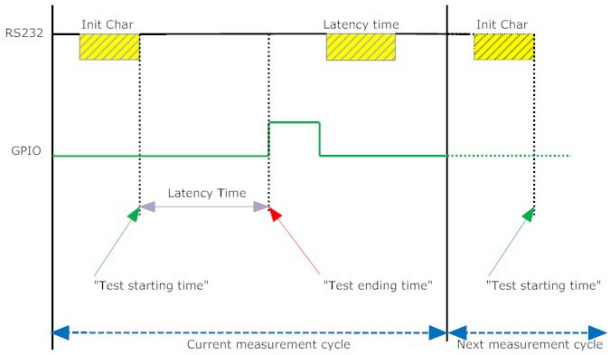

Figure 2: Latency time measurement test cycle

We stimulated the "Moblin platform" by an external event generated by the "Measurement board". The event consisted in sending a character through the serial communication device. We saved the time stamp when the character was released on the "Measurement board". We labeled it the "**Test starting time**". Upon the reception of that character, the "Moblin platform" toggled the GPIO pin, the "Measurement board" detected this change of state immediately and saved its occurrence time as the "**Test ending time**".

Based on the "Test starting time" and the "Test ending time", the "Measurement board" calculated the "**latency time**" and sent it back to the "Moblin platform" through the RS232.

Finally, we compared the obtained results to the reference value of 40µs which reflects a targeted value of the system at early characterization phases.

The way we found this reference value is outside the scope of this document.

2.3 Measurement of the scheduling time of a periodic task

The main objective of this test is to measure how accurate is the management of internal scheduling of tasks within the "Moblin platform". Figure 3 depicts the test design.

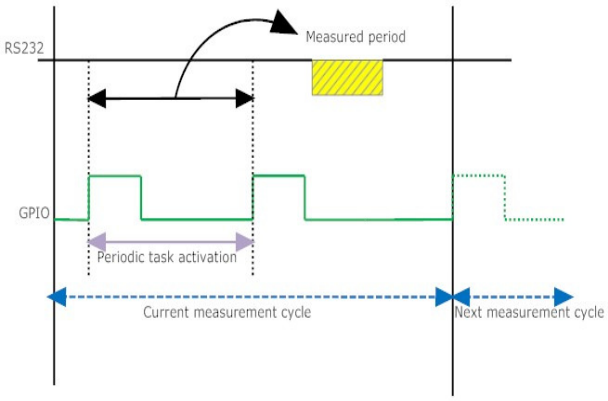

Figure 3: Periodic task scheduling test

We implemented a periodic task which consisted in changing the state of a GPIO pin in the "Moblin platform". We were interested in calculating the periodicity of this task when the "Moblin platform" was "lightly" and "fully" loaded. (Cf. §2.4)

We configured the "Measurement board" to calculate the time separating two consecutive GPIO rising edges and transmit this time back to the "Moblin platform" for measurement statistics accumulation.

Finally, we compared the cumulated statistics to the task reference period; the acceptance range has been set from -5% below to +5% above that reference period.

2.4 Tests conditions

We ran the set of tests described above in different configurations to simulate multiple environments where the infotainment applications can be incorporated. Intel's Atom processor CPU frequency can be adjusted on the fly from 800 MHz to 1.6 GHz depending on the system load. For the purpose of tests we have chosen to run at known frequency.

We evaluated the application test in both "light" and "full" loads configurations.

We define the "light" load configuration as the case when only the test application process is running along with Moblin on the "Moblin platform".

We define the "full" load configuration as the case when the test application process is sharing the 100% CPU load with two other hardware resources consuming processes: one is reading endlessly a USB memory stick and the other one is incrementing a counter. We set up the test application process with the highest priority level over all other processes.

Note: We have chosen the USB key reading process in order to put the system in interaction with an external environment, like that, the reading process preempted the kernel scheduling constantly.

3. Real-time capabilities of Moblin-Linux

This first experiment evaluates the real-time performances of Moblin-Linux APIs. The test application uses the Linux native APIs and the real-time parts (latency or task scheduling) use specific drivers to achieve the test objectives. In this case the drivers are also using the Linux-APIs. The detailed architecture is shown below:

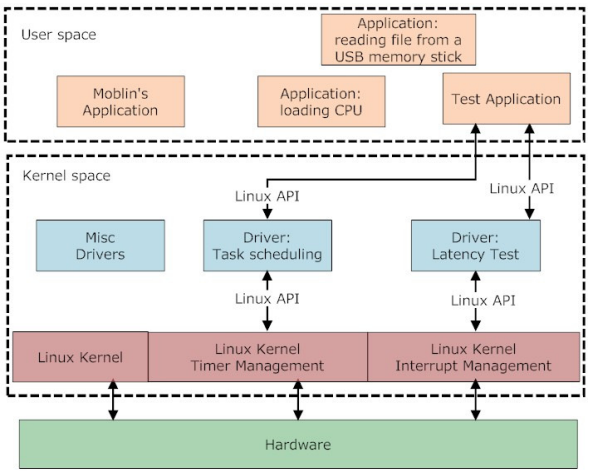

Figure 4: Test application with Linux APIs architecture

We set up the CPU frequency at 1.6 GHz and we ran the test application in "light" load configuration.

3.1 Latency time measurement test

We tested one million samples, for each sample we calculated the latency time of the "Moblin platform". The cumulated statistics are shown in Figure 5.

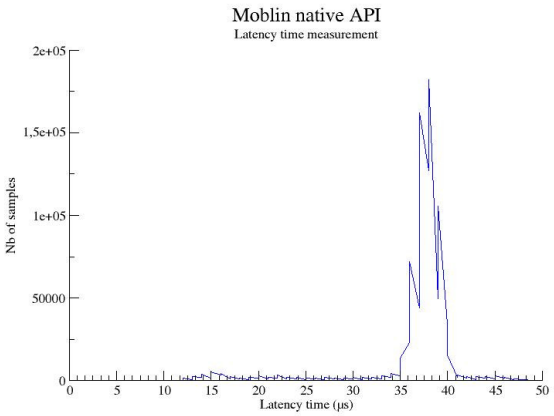

Figure 5: Moblin latency time test results

We found that almost 90% of samples had a latency time less than 40µs, whereas almost 1% of samples had that time higher than 5,5 ms. the latest 1% samples don't comply with the real-time deterministic requirement.

3.2 Periodic task scheduling measurement test

We ran the periodic task scheduling test one million cycles, each time we saved the task activation period in the "Moblin platform".

The smallest timer period that can be configured in "Moblin platform" is 1ms, so we set the task reference period to 2ms. The cumulated statistics are illustrated in Figure 6.

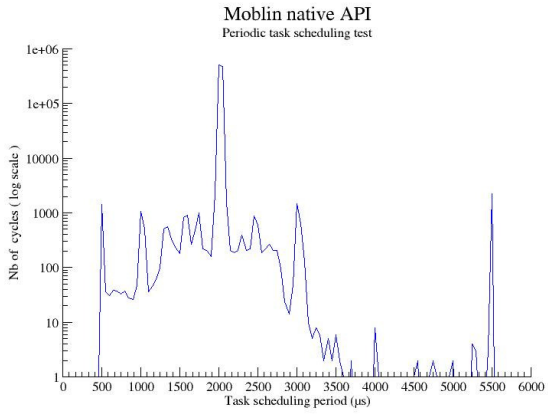

Figure 6: Moblin periodic task scheduling test results

The y-axis is given in a logarithmic scale. This graph shows that almost 98 % of calculated periods were from -3.5% to +2.6% around the expected value. However we noticed a high dispersion of all remaining values, which means that Moblin is not managing the task scheduling uniformly. In addition to that, more than 2000 samples had their periods 170% above the reference period (the peak on the right side) and this was out of the tolerated range.

Note: the maximum value that the "Measurement Board" can return is 5,5 ms. In other words, the real maximum latency time and task scheduling period values observed are in fact higher than the 5,5 ms limit reached in tests.

3.3 Assessment

Those two tests demonstrated the weaknesses of Moblin with respect to real-time constraints. Linux lacks determinism in response to interrupt events, and it can not manage periodic tasks scheduling adequately.

Moblin can not then be categorized in "hard" real-time systems class because it can not afford the desired deadlines of real-time tasks. However it can be considered to support the expected deadlines on average since it has an average time response lesser than 50µs; this is why it is said to belong to "soft" real-time systems class.

In the next section we will discuss how Moblin's real-time capabilities can be hardened by means of real-time extensions.

4. Adding a real-time extension to Linux

There are several options available to enhance the real-time features of Linux, some of these are based on Linux alone (changing the kernel configuration), some use Linux with an additional sub-kernel, and since kernel 2.6 software patches are used to improve the real-time behavior of Linux. Here we focus our study on the configuration of Linux with a sub-kernel; this technique consists in using a second kernel as an abstraction layer between the hardware and the Linux kernel. The non-real-time Linux kernel runs in the background as a lower-priority task of the sub kernel and manages all non-real-time tasks. The real-time tasks are processed by the sub kernel. (See Figure.7)

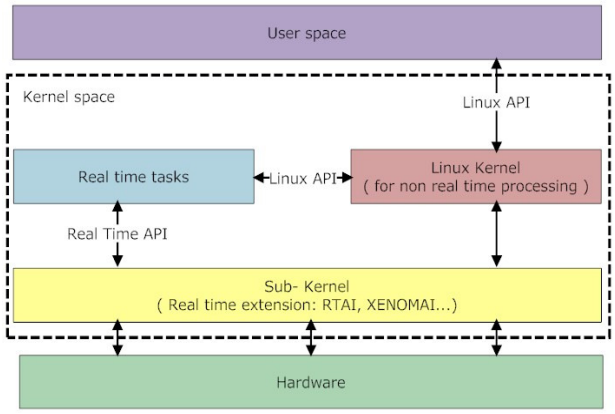

Figure 7: The real-time extension approach

Among all solutions we have chosen the RTAI and XENOMAI sub-kernels. We configured those extensions to run on the "Moblin platform".

The test application uses the Linux native APIs and the real-time parts (latency or task scheduling) use specific drivers to achieve the test objectives. In this case the drivers are using the real-time APIs instead of the Linux APIs. The detailed architecture is shown below:

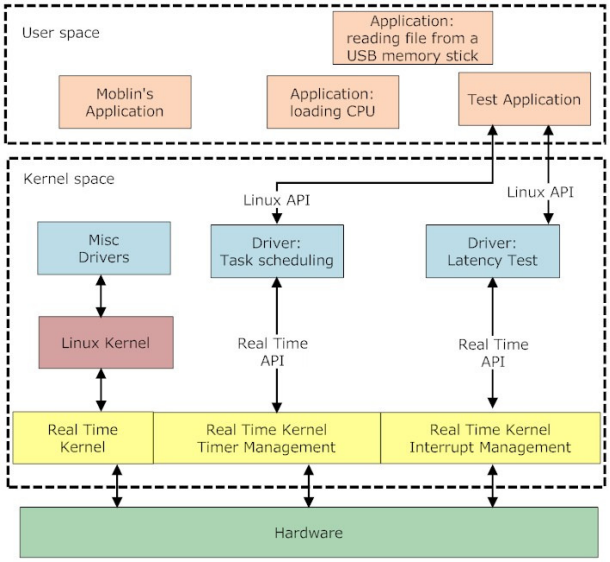

Figure 8: Test application with Linux and real-time APIs architecture

We executed the same set of tests already described in section 2 to evaluate their added-values in improving the real-time performances of Moblin.

5. Real-time capabilities of Moblin + RTAI

RTAI stands for Real-Time Application Interface. It is a real-time extension for the Linux kernel, we setup and configured Moblin + RTAI to get it running on the platform.

5.1 Latency time measurement test

We carried out the latency time measurement test on lightly and fully loaded systems at both 800 MHz and 1,6 GHz CPU frequencies. We tested one million samples, for each sample we recorded the latency time. The total statistics are represented by Figures 9 and 10.

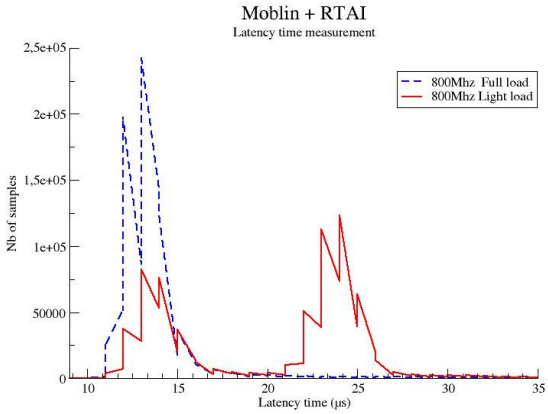

Figure 9: Moblin + RTAI latency time at 800 MHz

At 800 MHz, the dashed line graph represents the response of a fully loaded system; in this case the average latency time was 13,8 µs with 99% of samples below 28,8 µs.

The straight-line graph represents the response of a lightly loaded system; the average latency time was 20,3 µs with 99% of samples below 33,1 µs. Two peaks centered on 13 µs and 24 µs were detected, the reason behind is due to the internal architecture of the Intel's atom processor. Indeed, we noticed that when the processor is not fully loaded it goes into a sleep state and the response to an external event is then delayed by the wake-up time the processor takes before resuming normal operations.

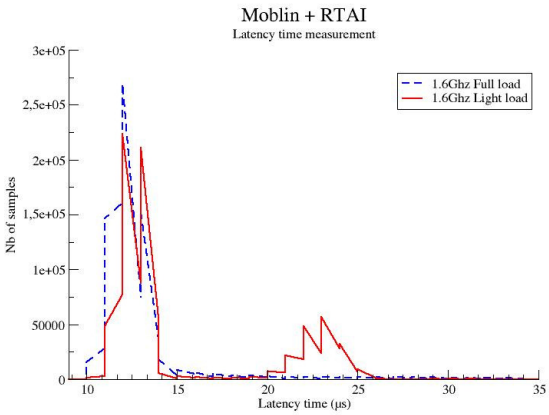

Figure 10: Moblin + RTAI latency time at 1,6 GHz

At 1,6 GHz, the dashed line graph represents the response of a fully loaded system; the average latency time was 13,3 µs with 99% of samples below 32,1 µs, the main difference compared to the 800 MHz case was that the values were more squeezed to the left and the average was slightly smaller.

The straight-line graph represents the case of a lightly loaded system; the average latency time was 15,6 µs with 99% of samples below 25,5 µs, which means that the performance has been improved by 25% compared to the 800 MHz case.

Most of the samples were between 10 and 15 µs and here also we observed another small peak around 23 µs, this was caused by the processor wakeup time.

Note: Only 0.1% of measured samples exceeds the reference latency time of 40 µs but remains below 100 µs.

5.2 Periodic task scheduling measurement test

We set the task reference period to 2.5 ms, we carried out the periodic task scheduling test one million cycles; at each cycle we recorded the task activation period in the “Moblin platform”.

The cumulated statistics are illustrated in Figures 11 and 12.

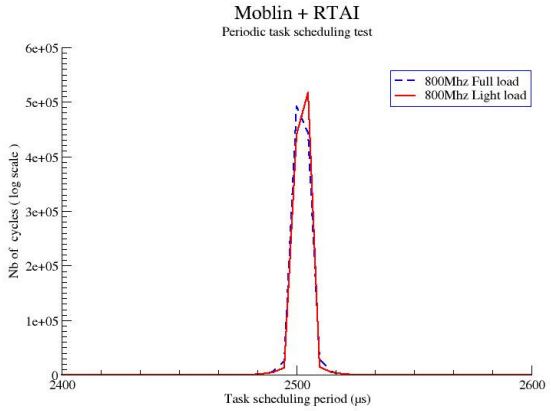

Figure 11: Moblin + RTAI periodic task at 800 MHz

At 800 MHz, the graphs almost matched, for both fully and lightly loaded systems 98 % of samples were from -0.6% to 0.54% around the reference period.

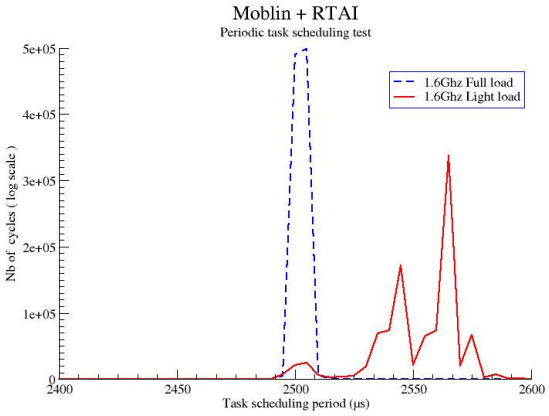

Figure 12: Moblin + RTAI periodic task at 1.6 GHz

At 1,6 GHz, for a fully loaded system, 98% of values were from -0.16% to 0.17% around the reference period.

For a lightly loaded system, 98 % of values were from -0.16% to 3.15% around the reference period. In this case we noticed a high dispersion of values above the expected period.

5.3 Assessment

We obtained the best real-time performances of Moblin plus RTAI extension with fully loaded systems. RTAI improved the Moblin management of external events but lacked determinism in scheduling periodic tasks.

6. Real-time capabilities of Moblin + XENOMAI

XENOMAI is a real-time extension cooperating with the Linux kernel, newer than RTAI and more “hard” real-time oriented. We setup and configured Moblin + XENOMAI to get it running on the platform.

6.1 Latency time measurement test

We carried out the latency time measurement test on lightly and fully loaded systems at both 800 MHz and 1,6 GHz CPU frequencies. We evaluated one million samples. The total statistics are depicted by Figures 13 and 14.

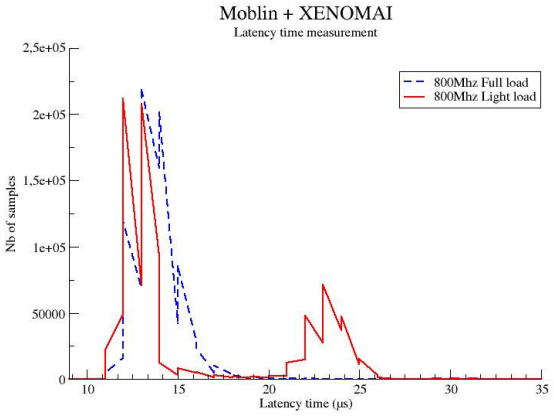

Figure 13: Moblin + Xenomai latency at 800 MHz

At 800 MHz, fully loaded system has an average latency time of 14 µs with 99% of samples below 19,1 µs while lightly loaded system has an average latency time of 16,2µs with 99% of samples below 25,8µs.

Like RTAI, we observed another peak due to the processor low power mode around 23 µs.

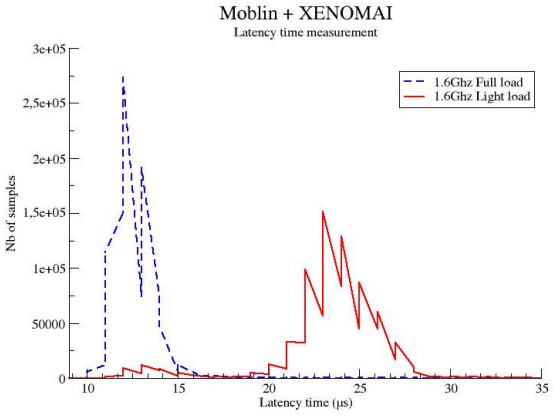

Figure 14: Moblin + Xenomai latency time at 1,6 GHz

At 1,6 GHz, fully loaded system has an average latency time of 13 µs with 99% of samples below 29,8 µs; while lightly loaded system has an average latency time of 23,6 µs with 99% of samples below 32,5 µs. Most of the samples were between 20 and 30µs. Here again we have seen a small peak around 13µs due to the processor internal architecture.

Note: Only 0.1% of measured samples exceeds the reference latency time of 40 µs but remains below 100 µs.

6.2 Periodic task scheduling measurement test

We performed the periodic task scheduling test one million cycles, at each cycle we recorded the task activation period in the “Moblin platform”. The cumulated statistics are illustrated in Figures 15 and 16.

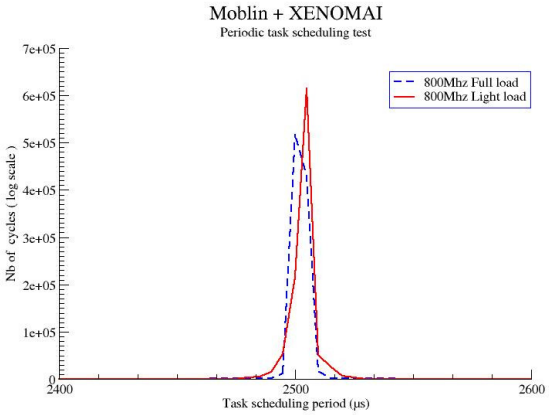

Figure 15: Moblin + Xenomai periodic task at 800 MHz

At 800 MHz, both fully and lightly loaded systems had 98% of samples situated from between -0.6% and 0.56% around the reference period.

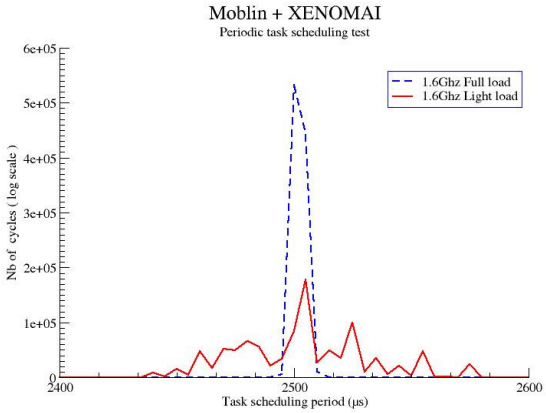

Figure 16: Moblin + Xenomai periodic at 1.6 GHz

At 1.6GHz, fully loaded system has 98 % of values located at +/- 0.2% around the reference period, while lightly loaded system has 98% of samples from -2.4% to 2.9% around the expected value.

In the second case, due to the processor low power mode, we noticed a dispersion of values but the average remained acceptable.

6.3 Assessment

The best real-time performances of Moblin + XENOMAI extension were obtained with both fully and lightly loaded systems; XENOMAI is deterministic with respect to “hard“ real-time constraints governing the handling of external events and the scheduling of periodic tasks.

7. Conclusion

Many Linux real-time extensions and distributions are becoming available, they are responding to the high technical use of embedded Linux in various domains. Since multiple approaches are available, it is obvious that no one solution will serve all applications best. In this paper, we have evaluated and tested the real-time extensions RTAI and XENOMAI with Moblin in different configurations. We outlined the limitations of Moblin and the RTAI approach and highlighted how XENOMAI is improving the real-time behavior of Moblin, enabling it to satisfy some “hard” real-time constraints. Other solutions based on kernel patches will be addressed in future work.

8. Acknowledgement

We would like to express our warm thanks and regards to our CTO Mr. Thierry SEYNAEVE and the embedded software department head Mr. Thierry BOUQUIER for their guidance and encouragement throughout this project.

We are also grateful to our mother company ESG-GERMANY for providing financial support to this project.

9. References

[1] C. Blaess: "*Programmation système en C sous Linux* ", 2nd edition, Eyrolles, 2005.

[2] P. Ficheux: "*Linux embarqué*", 2nd edition, Eyrolles, 2005.

[3] Moblin repository: *http://repo.moblin.org/moblin/releases/*

[4] Xenomai project’s web site: *http://www.xenomai.org*

[5] Rtai project’s web site: *http://www.rtai.org/*

10. Glossary

OS: Operating System

ECU: Electronic Control Unit

RAM: Random Access Memory

CAN: Controller Area Network

API: Application Programming Interface

CPU: Central Processing Unit

CTO: Chief Technical Officer

GPIO: General Purpose Input/Output

RS232: Computer serial line (Recommended Standard 232)

USB: Universal Serial Bus