Evaluation of Linux rt-preempt for embedded industrial devices for Automation and Power Technologies - A Case Study

Morten Mossige
ABB AS Robotics
Bryne, Norway
morten.mossige@no.abb.com

Pradyumna Sampath
ABB Corporate Research
Bangalore, India
pradyumna.sampath@in.abb.com

Rachana G Rao
ABB Corporate Research
Bangalore, India
rachana.rao@in.abb.com

Abstract

ABB is one of the leading solution providers for Industrial automation, Power technologies and Robotics. This makes ABB a vendor of several mission critical embedded devices. Most of these devices are characterized by stringent real time requirements. Real-time patches/extensions to Linux open up new possibilities to applying open source software in industrial real-time embedded applications. This paper presents a case study, where rt-preempt is evaluated in the context of an industrial controller.
The paper will reveal the methodologies used in this evaluation in terms of the test setup in detail and the parameters measured from an industrial control perspective. This evaluation has had focus on Linux in a distributed environment over Ethernet connectivity with corresponding time synchronization and real-time capabilities.

1 Introduction

ABB's core fields of expertise are Power and Automation technologies. Our wide range of products and vast base of customers demand that we focus on constant innovation and are always on the look out for newer, more efficient possibilities to cater to our customers in a better way.

Embedded systems in industrial automation, robotics and power technologies are characterized by certain factors unique to these market segments. They are timing critical, safety critical and have very long product life cycles. These factors play a key role in the evaluation of a next generation operating system for embedded applications. Another very important aspect is real time Ethernet over a distributed environment. Thus from a technical point of view, the evaluation is centered around hard real time capabilities, real-time Ethernet performance, sustainability and scalability.
Open source real time operating systems were given the first consideration due to their obvious technical and non-technical advantages.

2 Choices

In our evaluation, we considered some of the more popular and proven open source projects that were aimed at satisfying needs similar to ours.

2.1 Xenomai and RTAI

Xenomai[1] is a real-time development framework co-operating with the Linux kernel, in order to provide a pervasive, interface-agnostic, hard real-time support to user-space applications, seamlessly integrated into the GNU/Linux environment. Xenomai uses a co-kernel approach and shares hardware interrupts and system-originated events like traps and faults with the Linux kernel using the Adeos virtualization layer.

RTAI[2] is a real-time extension for the Linux kernel - which lets you write applications with strict timing constraints for Linux. Xenomai and RTAI enable smooth migration from traditional RTOS to Linux without having to rewrite entire application, while keeping stringent and deterministic real-time guarantees. Also, there exists an open source hard real-time network protocol stack called RTnet[3] for Xenomai and RTAI.

2.2 RTLinux

RTLinux[4] is an operating system in which a small real-time kernel coexists with the Posix-like Linux kernel. RTLinux supports hard real-time (deterministic) operation through interrupt control between the hardware and the operating system. Interrupts needed for deterministic processing are processed by the real-time core, while other interrupts are forwarded to the non-real time operating system. The operating system (Linux) runs as a low priority thread.

2.3 rt-preempt

rt-preempt[5] patch implements real time behaviour by allowing nearly all of the kernel to be preempted, with the exception of a few very small regions of code. It further incorporates high resolution timers. The premption capabilities and the hrtimers give the rt-preempt patch, its hard real-time[9] behaviour.

This patch is being considered for inclusion into the mainline kernel in parts. This was the primary motivation to consider rt-preempt for an evaluation for ABB's embedded applications.

Of equal importance to us, is to ensure license compliance and to define a support structure that is sustainable over long product life cycles with the close collaboration with the Linux community.

3 Hardware System Structure

The hardware used to evaluate rt-preempt is custom made for ABB and is based on the Lite5200 evaluation board from Freescale. This board is the heart of the paint process system for paint robots.

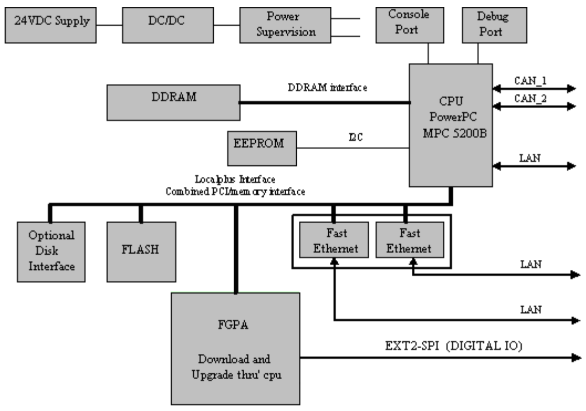

FIGURE 1: *Hardware details of the board*

The board is driven by the freescale processor MPC5200B system on chip, which is an embedded PowerPC processor based on the MPC603e Core. This is a processor rated to 750MIPS and running at 400MHz internally. The board is equipped with 64 MB DDR SDRAM and 32 MB Flash. The DDR SDRAM is running on a dedicated memory bus used only for that purpose. The Flash is connected to the localplus bus which is a multifunction bus also used by the PCI interface, the FPGA and the compact flash interface.

There are three individual Ethernet ports on the board. One is based on the internal controller in the processor and other two are standalone Ethernet controllers - Intel 8255 ER 10/100mbps - connected to the internal PCI bus. PCI arbitration is done by the FPGA. They do not have external EEPROM on the board.

4 Software Deployment

The bootloader used is u-boot-1.2.0[6]. Changes were made to support FPGA download via the PSC2 port.To begin with, we used Linux-2.6.16-rt29-tglx4 with the "ppc" tree of the kernel. With the Linux community deciding to withdraw support for the "ppc" tree, we shifted to the "powerpc" architecture tree on Linux-2.6.20-rt3. Since then we have constantly been keeping abreast of the developments happening on the -rt branch. The following benchmarks have been made on the Linux-2.6.21-rt3 kernel. Bestcomm patch was added to enable Ethernet and DMA support on the board.

4.1 Test Descriptions

The following parameters were deemed to be important benchmarks for the real time operating system on an Automation or Power platform user-space to kernel space transition latency, interrupt latency and network latency.

The FPGA program has been augmented to convert the SPI pins to act as Digital IO ports to facilitate the measurement of performance parameters. The input signal at the required frequency was generated using a signal generator and the output signal was captured on the Cathode Ray Oscilloscope. The difference between the digital input and the digital output was analyzed and the particular latency determined.

A kernel character device driver was written to access the memory mapped digital IO through the FPGA and to handle the FPGA interrupts.

The test suite, is especially tailored to measure parameters important/relevant to the application the board was designed for. It consists of three tests.

4.1.1 Write Latency Test

This test attempts to evaluate the minimum, maximum and the average deviation times between writes to a digital output. The interval between these writes is user defined. The test application creates a thread which sleeps for the user-defined interval of time and then calls a function to write the data to the digital output.

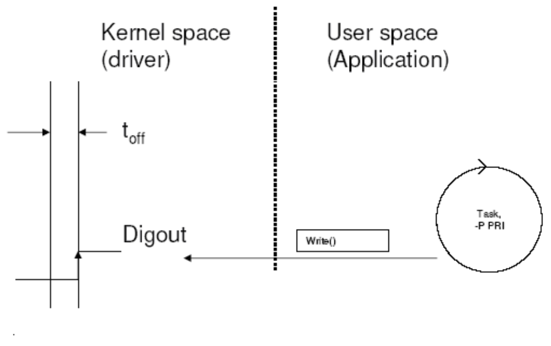

FIGURE 2: *Write Latency Test*

4.1.2 Read, Write and Interrupt Latency Test

This test attempts to evaluate the time taken for the data to travel from the digital input to the digital output. The data from the signal generator is received as an interrupt at the digital input. The application reads the data, inserts it into an event queue and sleeps for a user-defined interval of time. On wake up, the callback write function is called to write the data from the queue to the digital output.

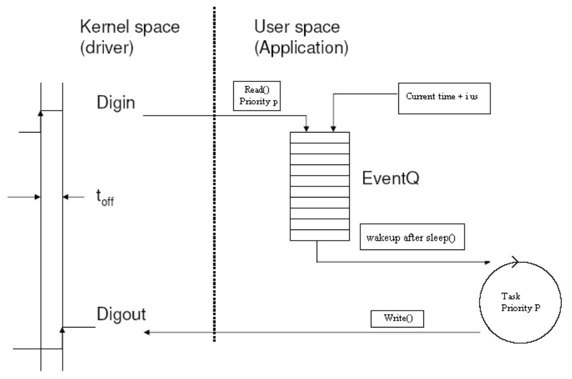

FIGURE 3: *Read, Write Latency Test*

4.1.3 Read, Write, Interrupt and Network Latency Test

This test attempts to measure the communication latency over Ethernet. The data from the signal generator is received as an interrupt at the digital input. The application on board1 reads the data, inserts it into an event queue and sleeps for a user-defined interval. On wake up, the callback write function is called to send out the data from the queue with the timestamp of board1, through the Ethernet network to board2. The application on board2 reads the data from the Ethernet , inserts it into an event queue and sleeps for a user-defined interval of time. On wake up, the callback write function is called to write the data from the queue to the digital output.

An important point in this test case is that the system clocks on the two boards are synchronized through the IEEE1588[7] protocol (precision time protocol).

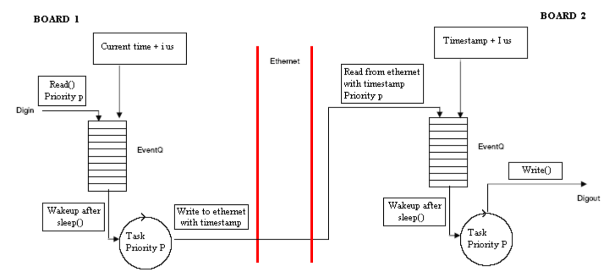

FIGURE 4: *Read, Write and network Latency Test*

5 Test Setup and Results

The test setup consists is as shown below.For network-related tests, the boards are time synchronized using the IEEE1588 Precision Time Protocol. The PTP daemon[10] runs on all the boards connected to the network with one of the boards acting as the master and the rest of them as slaves. The master board periodically launches an exchange of messages with the slave boards to help each slave clock recompute the offset between its clock and the masters clock. This offset will drift with time and so these periodic exchanges mitigate the impact of this drift on the clock synchronization.

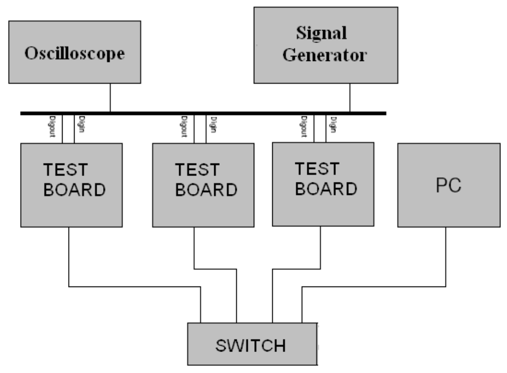

FIGURE 5: *Test Setup*

The setup and the tests reflect a scenario in the robotics domain.

5.1 Results

Two kinds of tests were conducted. Twenty four hour tests which are representative of latencies over a large number of samples. Incremental load tests measure latencies over a range of loads. System load is generated by running stress[8] on the board. The parameters of stress are adjusted to achieve a particular load.The jitter in the following sections mean max-min on that sample space.

5.1.1 Write Latency Test

The user-defined time delay between the writes was given as 500μs.

24 hour test (Results in μs)

	Min	Max	Avg	Jitter
No Load	457.136	539.530	499.508	82.394
With Load	439.285	537.075	494.517	97.790

For this test the maximum allowed jitter is 100μs and from the table it is evident that the jitter is within the specified limits for both load and no-load conditions.

Incremental load test

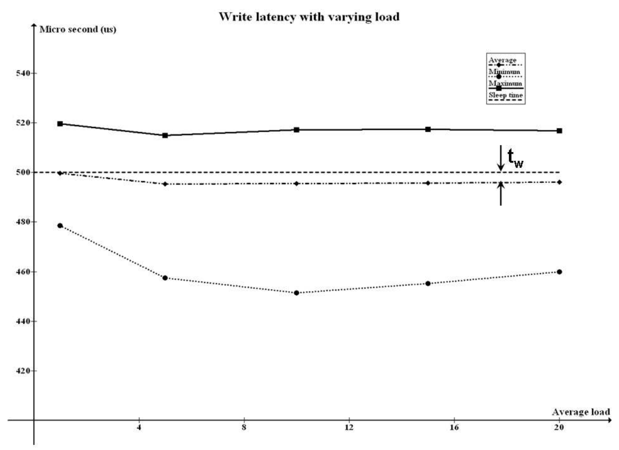

FIGURE 6: *Write Latency Result*

From the above graph, it can be seen that the varying CPU load does not affect the time taken for the writes. It can also be observed that the average values are close to the ideal value, which is the user defined sleep time of 500μs. t_w is the write latency calculated as the difference between the average values and the user-defined sleep time and is approximately 10μs.

5.1.2 Read, Write and Interrupt Latency Test

The input frequency was given as 500Hz and the user-defined time delay between the digital input read and digital output write was given as 200μs.

24 hour test (Results in μs)

	Min	Max	Avg	Jitter
No Load	263.800	343.458	296.240	79.658
With Load	266.043	364.067	286.358	98.024

For this test the maximum allowed jitter is 100μs and from the table it is evident that the jitter is within the specified limits for both load and no-load conditions.

Incremental load tests

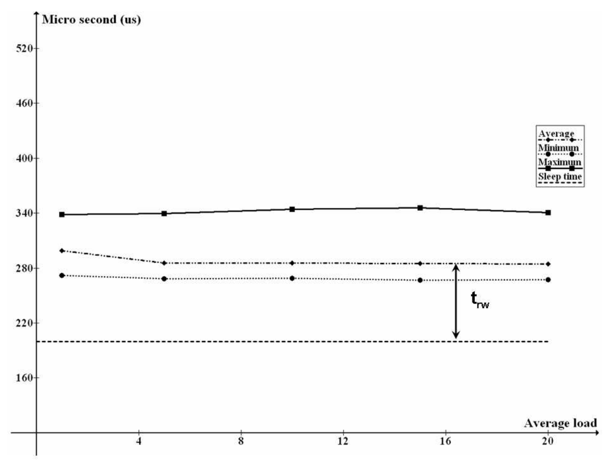

FIGURE 7: *Read, Write and Interrupt Latency Result*

From the above graph, it can be seen that the varying CPU load does not have a substantial effect on the values measured. t_{rw} is the read(interrupt) and write latency calculated as the difference between the average values and the user-defined sleep time and is approximately 100μs. The difference between t_{rw} and t_w gives us the read(interrupt) latency, t_r.

$$t_r = t_{rw} - t_w \approx 90\mu s$$

5.1.3 Read, Write and Network Latency Test

The input frequency was given as 500Hz. The user-defined time delay between the digital input read and Ethernet send on Board 1 was given as 200μs.The user-defined delay between the Ethernet receive and the digital output write on Board 2 was given as 200μs.

24 hour test (Results in μs)

	Min	Max	Avg	Jitter
No Load	511.572	869.064	660.099	357.492
With Load	473.216	854.517	659.304	381.301

For this test the maximum allowed jitter is 1000μs and from the table it is evident that the jitter is well within the specified limits for both no-load and load conditions.

Incremental load test

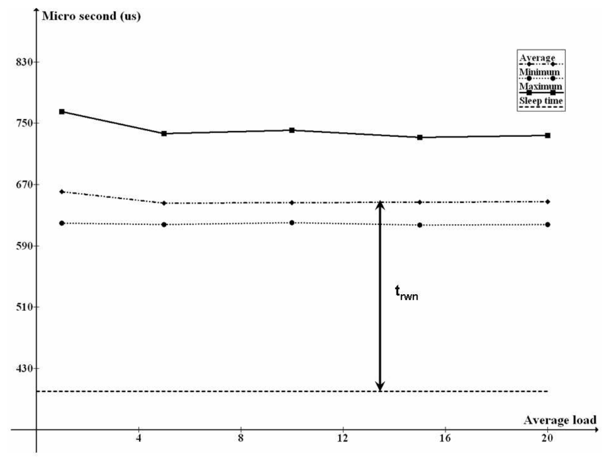

FIGURE 8: *Read, Write and Network Latency Result*

From the above graph, it can be seen that the results are constant across varying CPU load. t_{rwn} is the read(interrupt), write and network latency calculated as the difference between the average values and the user-defined sleep time and is approximately 250μs. The difference between t_{rwn} and t_{rw} gives us the network latency, t_n

$$t_n = t_{rwn} - t_{rw} \approx 150\mu s$$

6 Conclusions

An industrial controller requires an extremely stable operating system. Depending on the application it runs, the real-time requirements can also be challenging. Linux has for a long time proven that its stability is excellent, and now we see that the real-time performance is really moving towards other commercial real-time operating systems. The ability to be able to run a real-time application on the same processor as other standard applications is a winning combination. This is really what favors Linux as a real-time operating system compared to other dedicated real-time operating systems.

During the course of this evaluation we have validated and benchmarked certain real-time performance parameters in the context of paint robots.

The results are optimistic and the prospects are bright for the future with Linux.

7 Acknowledgments

We would like to thank our colleagues Srijit Kumar and Girish Kathalagiri for their time and efforts.

References

[1] *Xenomai - Implementing a RTOS emulation framework on GNU/Linux*, Phillipe Gerum,April 2004

[2] *https://www.rtai.org/*

[3] *http://www.rtnet.org/*

[4] *http://www.windriver.com*

[5] *Internals of the RT Patch*, Steven Rostedt and Darren V. Hart, June 2007.

[6] *http://www.denx.de/wiki/UBoot/WebHome*

[7] *http://ieee1588.nist.gov/*

[8] *http://weather.ou.edu/ apw/projects/stress/*

[9] *http://rt.wiki.kernel.org/*

[10] *http://ptpd.sourceforge.net*