

Evaluation of Real-time operating systems for FGC controls

Konstantinos Chalas, CERN, Geneva, Switzerland

September 2015

Abstract

Power Converter Control for various experiments at CERN, is conducted using a machine called Function Generator Controller. The current generation of FGCs being deployed is FGC3. A certain number of FGCs require very fast and precise control, and for these systems, there is uncertainty of whether the existing hardware will be able to provide the level of determinism required. I have worked in the CCS section as a summer student on a project to study the potential of ARM-based CPUs to provide a real time behaviour fit for a future high-performance FGC4. In this paper, i will present the results of my research into real-time variants of Linux and other real-time operating systems on two different ARM CPUs.

1 Introduction

1.1 What is Real-time?

Real-time programs must guarantee response within strict time constraints (i.e. “deadlines”). Deadlines must be met, regardless of system load. Real-time systems should take into account worst case scenarios. This systems are getting more and more important for different uses in the industry, and most commonly for embedded systems projects. Currently, there are a lot of available Real-time operating systems in the market, both open-source and proprietary. In my research, i decided to focus on open source, free software and POSIX compatible Real-time operating systems, since longevity (above 20 years of maintenance) is an important factor.

1.2 ‘Real Time’ vs. ‘Real Fast’

A lot of people tend to confuse Real Time performance with processing power and fast input/output. While the two of them are closely connected, they can not be considered equal by any stretch of the imagination. To quote Doug Niehaus, “Realtime is not as fast as possible - realtime is as fast as specified”.

General purpose operating systems (e.g Linux and Windows) are optimized for throughput and multitasking. The performance of a Real Time operating system is measured by latency. In my experiments, my main interest was interrupt latency, the time that elapses from when an interrupt is generated to when the source of the interrupt is serviced.

1.3 Types of Real-time systems

The trade-off between throughput and Real Time performance have set a precedent of multiple types of Real Time systems. The three basic categories are:

Soft

Soft Real-time operating systems can afford to miss some deadlines but eventually performance will degrade if too many are missed. A good example is the sound system in your computer. If you miss a few bits, you will hardly notice any effect, but miss too many and you're going to eventually degrade the quality.

Hard

Hard Real-time operating systems should guarantee that the deadlines should be met 100% of the time or give a timely indication, announcing that the deadline is going to be missed. Failure to do so, can even result in equipment failure, injury or even loss of life. An example is a process control program, where timing failures result in product manufacturing defects.

95% Hard

The Real-time requirements should be met at least 95% of the time. For example, in a audio recording studio, some samples can go missing, but if this percentage is too low, there will be no impact on quality

2 Evaluated Real-time operating systems

2.1 Criteria for selecting a Real-time operating system

The decision process for selecting and evaluating a Real-time operating system was made up of four factors:

- Available for the ARM architecture
- POSIX compatibility
- Free and open-source software
- Active, healthy community

2.2 Linux PREEMPT-RT

Linux is a general purpose operating system. It was not created with the mindset of a Real-time operating system. However, the popularity and availability of Linux, inspired some people to add hard Real-time capabilities to its Kernel. In order for this to be accomplished, the mainline Kernel have to be patched, configured and compiled with the PREEMPT-RT patch. The Linux PREEMPT-RT approach meets all the criteria that i mentioned.

2.3 Xenomai-3 Cobalt Kernel

Xenomai takes a different approach from the Linux PREEMPT-RT. Instead of relying on the monolithic Kernel that Linux provides, it supplements Linux with a Real-time co-kernel running side by side. This microkernel is built into the Linux Kernel, and takes advantage of the modular capabilities of the Linux Kernel. The Real-time activities, including the interrupt handlers, scheduling real-time threads is taken care by the micro-kernel. The activity of the Cobalt core is prioritized over the native Kernel activities.

3 Experimental setup

The setup was based on two low-end single-board computers. That is, the Beaglebone Black revC and the Raspberri Pi 2 B. They are both widely available all over the world and easy to get to. This is a summary of their hardware, as well as some pros and cons:

Beaglebone Black

- 1GHz single-core ARM Cortex A8
- 512MB RAM
- \$50 retail price
- Open-source hardware
- Produced by Texas Instruments (Long Term Support ensured)

Raspberry Pi 2

- 900 MHz quad-core ARM Cortex A7
- 1GB RAM
- \$35 retail price
- More oriented towards hobbyists
- Produced by Broadcom

4 Measurement

Since the objective is learning about Real-time operating systems, emphasis was given to metrics about latency and jitter. The famous and cited commonly

amongst users of RT-PREMMPT tool, cyclicttest was used to explore the worst case latency that each SBC has to offer.

Moreover, since Real-time systems are all about worst case, i had to put some pressure into my measurements. This is why, while running cyclicttest, i run hackbench and compiled the Linux Kernel source code. In case you are not familiar with hackbench, it creates a specified number of pairs of schedulable entities which communicate via pipes or sockets. All the measurements were taken over long periods of time (i.e 10 hours), since, like i already mentioned, worst case is what we care about, and what better way to find it other than stressing a system for 10 consecutive hours. For our case, 100 microseconds is the hard real time requirement.

Last but not least, i had to profile the Converter Controls libraries, in order to found out what kind of run time should we expect from the single-board computers. Run time is as, or more important as latency for our ecosystem. The run time of Converter Control libraries consist of:

- Calculate reference
- Run regulation
- Run reference state machine
- Simulate voltage source and load response times
- and other important ingredients

5 Results

Let us dive into the results and conclude as to whether or not a our Real-time operating systems put into trial are efficient enough for our needs

5.1 Latency and Jitter

	Mean	Worst case	Std. deviation	Hard
BBB with PREEMPT-RT	44.045 us	202 us	14.121 us	97.506%
RPI 2 with PREEMPT-RT	21.682 us	188 us	8.379 us	99.994%
BBB with Xenomai	11.973 us	48 us	3.462 us	100%

The Beaglebone running Linux with PREEMPT-RT had the worst performance from all the setups. The Raspberry Pi 2 has much better Real-time performance, however the worst case latency is up to 188 us, which is a big disadvantage, when the target is 10 kHz. Xenomai overthrows RT-PREEMPT on every single aspect, and keep in mind that it is benchmarked using the Beaglebone Black.

5.2 Converter Controls Libraries

Let's not forget that the actual need for a Real-time operating system, besides determinism, is to actually run an application, after it recovers from latency. For example, there is no use for a 5 us latency, when the application itself takes 100 us. Therefore it is important to profile our application.

ccRt Iteration	Mean	Worst case	Std. deviation	Below 50 us
RPI 2 with PREEMPT-RT	17.825 us	134.231 us	5.504 us	99.850%
BBB with PREEMPT-RT	54.361 us	208.274 us	12.968 us	50.867%
BBB with Xenomai	23.146 us	169.127 us	4.137 us	99.774%

As we can observe, the gap between the Xenomai and RT-PREMMPT systems is huge. It is unfortunate that Xenomai does not support Raspberry Pi. Judging from the improvement that Xenomai offered to the runtime of the application for the Beaglebone Black, it would do wonders for the run time on the Raspberri Pi 2

6 Conclusion

Both of our computers approach hard Real-time behaviour, however 100% hard is currently impossible to meet with either operating system. If someone manages to port Xenomai over to Raspberry Pi 2, i am certain that a hard Real-time system for the Converter Control software would be feasible. Also, a new board produced by Texas Instruments, called Beagleboard X15 is coming out by the end of the year. It has almost twice the horsepower of Raspberry Pi 2 and Xenomai developers will likely support it.