Experimental Analysis of the Linux RT-patched for Data Acquisition applied to Power Sector

Roberto Alexandre Dias
Federal Institute of Santa Catarina
Mechatronics

Tiago Emanoel de Souza
Federal Institute of Santa Catarina
Mechatronics

Valdir Noll
Federal Institute of Santa Catarina
Mechatronics

ABSTRACT
The use of SoCs allow a compact and low cost post-processing system for monitoring electrical magnitudes. The main idea of this work is to have a small hardware which is deterministic enough to execute acquisition and sufficiently power to process data at a low cost.

In this paper, it is presented two problem domains of electrical systems measurement, one with acquisition of 1 kHz and the other with 5 kHz. It was built an acquisition test bed with a SoC with standard and real-time Linux to compare and evaluate its timing constraints.

General Terms
Data Acquisition

Keywords
Real Time, Linux, acquisition system, System on Chip (SoC).

1. INTRODUCTION
Data acquisition is important to preventive maintenance and monitoring systems for the utilities industry. Specifically in the energy sector, a demand for most efficient and sustainable power generation systems increases the need for a post processing systems and visualization data. Typical scenarios include acquisition of electrical and mechanical parameters for applications in power sector.

The monitoring system for power generators group requires compact, low cost and powerful processing systems with high definition graphical interfaces. The use of System on Chip (SoC) platforms meets this goal. SoCs are not expensive, with high processing power and easy integration with measurement systems. In addition, the use of SoC with Linux Operating System(OS) improves the development curve through availability of open source applications and libraries.

In this work a data acquisition system based in a SoC over Linux OS (non real-time and real-time) was evaluated. A test bed was developed to measure the data acquisition time for typical applications in energy sector.

2. SYSTEMS ON CHIP – SOC
The progress of microelectronic technology allowed miniaturization and higher performance of electronic components. With these advantages, the amount of features in a single Integrated Circuit (IC) increased.

A SoC can be defined as "an IC that integrates all components of a computer or other electronic system into a single chip"[1].The benefits are low cost, low power consumption, ideal for complex systems with space limitations [1].

The block diagram of Allwinner A20 SoC in Figure 1 shows the supported variety of features, processors, I/O interfaces and memories. Today's best examples of SoC's use are on smart phones and tablets due to space limitations and low power consumption constraints.

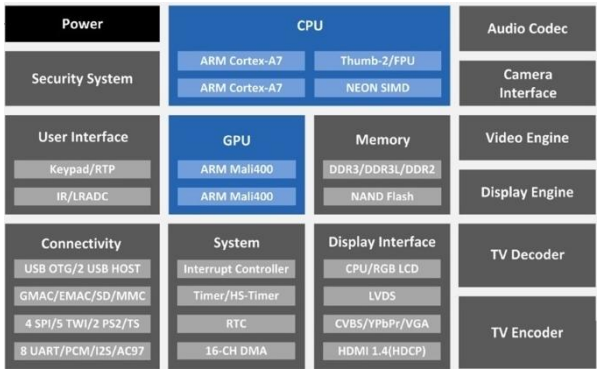

Figure 1: Block diagram of Allwinner Tech A20 SoC [9].

Since smart phones and tablets are all about graphic and sound features, it is notable the concern of multimedia processing on SoCs. An example is the A20 that has NEON processor with Single Instruction Multiple Data (SIMD). This processor has parallel processing of 8, 16, 32 and 64 bit and single precision floating point. Which is used for signal processing algorithms such as video encoding/decoding, 2D/3D graphics, gaming, audio, speech and image processing, telephony, and sound synthesis [2].

As this project is about signal acquisition and post-processing, it will target especially the FFTW algorithm, which has optimized implementation for NEON and SIMD technology [3].

3. THE PROBLEM
The intended application for this system is the monitoring of electrical systems such as power generating groups. In addition, the main goal is evaluate if the use of SoC is feasible for solving two problem domains:

1. Acquiring and post processing electrical energy parameters like voltage, current, active power, and reactive power.
2. Acquiring and post processing mechanical parameters in power generators like vibrations in bearings

In the first domain the involved signal frequency is around 50 and 60 Hz up to tenth harmonic (500 to 600 maximum frequency).

In the second domain, the signal frequency is around 500 Hz up to tenth harmonic (5 KHz maximum frequency).

The use of SoC could allow a compact and low cost post processing system for these two domains. Besides offering complex and real time computation (e.g. software FFT computation, stochastic processing, graphical processing) and a sophisticated Human Machine Interface (HMI), that are needs in the Power Sector. An example of HMI is shown in Figure 2.

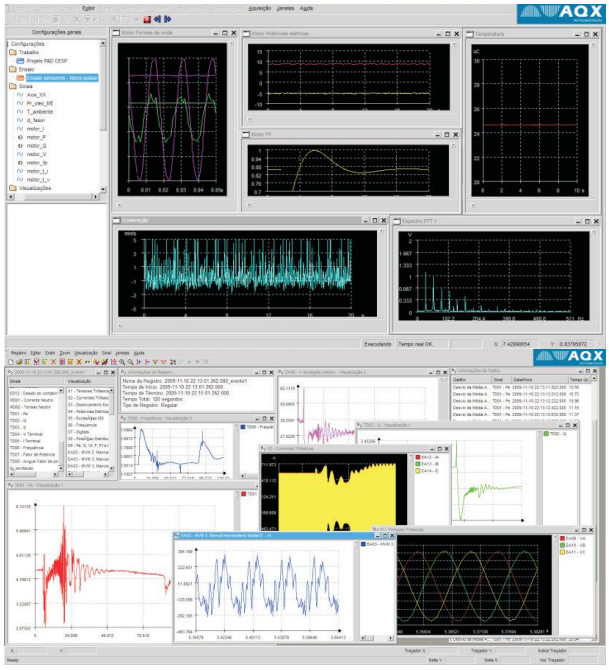

Figure 2: Graphical Interface for electrical monitoring systems [4]

In this work, two performance parameters are measured:

- Acquisition time for eight analog input channels of 16 bits resolution;
- Delay variation of acquisition time (jitter) for eight analog input channels of16 bits resolution.

The delay time between acquisitions is the most important pre-requisite for a correct conversion between analog to digital signals using a FTT technique, because this, it was choose how the main parameter to evaluation.

It was used standard and real time Linux OS for measuring these parameters.

The platform and test bed are shown in section 5.

4. RELATED WORKS

The paper [5] shows the RT-patch implementation. It explains the use of a real-time OS to provide deterministic behavior and not necessarily improving performance of real time applications. As well as in our results, the author illustrates the concept for a hypothetical algorithm that can complete some calculation in 250 microseconds in a standard operational system (non real-time) and it can execute same calculations in 300 microseconds(deadline) in a real time one.

In the first case (non real-time), the non-deterministic behavior can be impractical for real time application because in some instances the execution times exceed the deadline of 350 milliseconds. In the second (real time), although it had longer runtime, the jitter is smaller and compatible with deterministic real time applications. The work shows the real time APIs, provided by RT-patch to control latencies using high-resolution timers and Priority Inheritance (PI). The PI allows applications to use a fast mutex (futex) completely in user space. The author compares the deterministic behavior through execution of the gettimeofday() function in standard and RT-patched Linux. The Figure 3 and Figure 4 show this test and prove that latency in RT-Linux is better than STD Linux.

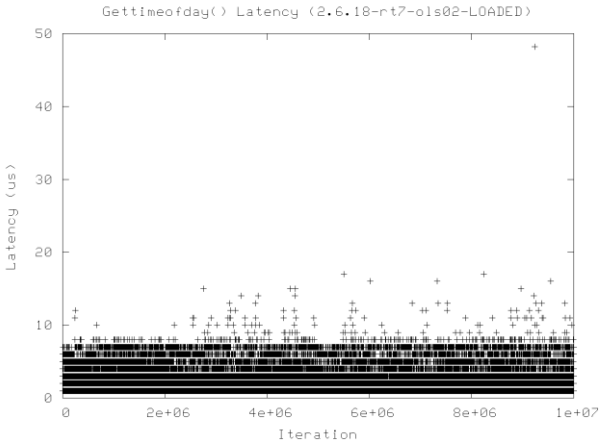

Figure 3: GTOD Latency with RT-Linux [5].

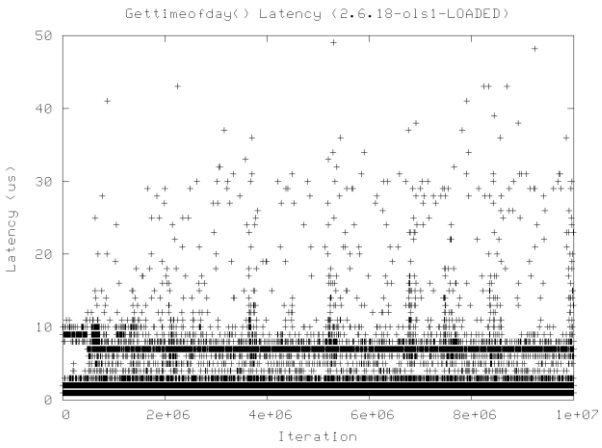

Figure 4: GTOD Latency with STD Linux [5].

In paper [6] the authors present a test bed to evaluate the response of Linux OS RT-patched kernel version 2.6.25-4-rt4. It executes a synthetic real time application under a best effort background application with a single user.

In these tests is measured real time application's jitter and activation time using two experiment conditions: (i) without RT-patch and (ii) with RT-patch for four sets of execution duration (10ms, 5ms, 1 ms e 0.5 ms). As the result, the jitter´s performance in RT-patch proves to be better without RT-patch.

In paper [7] the authors present a study and evaluation of the use of a standard (non-real time) Linux OS for real-time robotics and manufacturing control systems, with commercial off-the-shelf (COTS) embedded hardware. The authors argument that the dramatic increase of power processing in general purpose embedded systems allow developing of real time applications in industry on certain fields of application. By using RTC threads [8], they obtain good results for 1 KHz acquisition and actuators frequencies for industrial applications.

5. DATA ACQUISITION SYSTEM USING SOC AND LINUX

To attend the goal discussed in the section 3, a system was developed according to the model on Figure 5. The main idea is to have a small hardware, which can execute acquisition and processing data in the same piece.

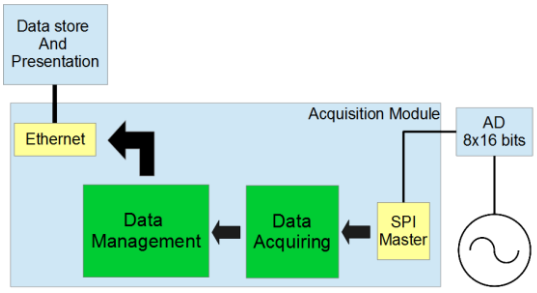

Figure 5: Acquisition schema.

In one side, the "Data Acquiring" block which is responsible for getting data from A/D chip through the Serial Peripheral Interface Bus (SPI) has to be able to acquire all 8 channels of 16 bits every 200µs, each generate a 640kb/s of data load. On the other side, the "Data Management" block has to be able of processing the sampled data and send it by the Ethernet interface on a rate that doesn´t exceed the data sampled buffer.

Due to the high processing load, it is necessary a SoC with the peripherals and processing power consistent with that rate. In this work it was used the Allwinner A20 SoC[9]. This is a SoC that has a dual core ARM7 processor with a 1.2GHz clock; some of its features are listed below [9]:

- 256KiB L2-Cache (shared between two cores);
- 32KiB (instruction) / 32 kB (Data) L1-Cache per core;
- Large physical Address Extensions (LPAE) 1TBLPDDR2/DDR3/DDR3L controller;
- NAND Flash controller and 64-bit ECC;
- GPU Mali400MP2;
- Video decoder of 2160P;
- Video encoder of H.264 1080P@30fps;
- HDMI 1.4, CPU/RGB/LVDS LCD interface;
- NEON processor with SIMD capability.

A simple way to use an analog to digital converter (ADC) on Linux is through the SPI. SPI is a standard serial four-wire synchronous data bus that can operate in full duplex. Devices communicate in master/slave mode with a single master initiating data frames [10].

Looking toward the prerequisites of this work, two ADCs ICs were chosen for testing purpose, the ADS8568 from Texas Instruments [11] and the ADAS3022 from Analog Devices [12].Both are 16-bit 8-Channel, successive approximation register (SAR) based ADC and support SPI interface, analog input signals with amplitudes up to ±12V and signal acquisition up to 650kSPS.

The Allwinner A20 has three available SPI interfaces. Each SPI interface has two available slave select (SS) lines for use [13]. It can be used one ADC chip per interface or two on the same one using SS0 and SS1lines. For this project, it was used the first option. The SPI is available in the Linux kernel but a module must be installed when compiling it[11] and the FEX file must be configured. The FEX is a hardware configuration file, and among other things, it defines ports configuration of the SoC [14].

The benefits of the chosen architecture is to use the powerful hardware largely used on tablets and smart phones, this includes processors with multiples cores and several peripherals at low cost. The other benefit is open-source community linux-sunxi dedicated to develop drivers and porting for Linux platform that provides documentation and a git repository [15].

Looking for hardware that can be easily attached to a carrier board, it was chosen the AW-SoM A20 SODIMM Module, which brings the Allwinner A20 chip, 1GB/RAM and 4GB/flash memory, ethernet phyceiver and a Real Time Clock on a module with 68x52mm size [13].

The main advantages of the AW-SoM A20 Module, shown on Figure 6, is that it makes all features of the Allwinner A20 be easily accessible with less effort, with lower development cost and with an easy to plug and change adapter like SODIMM.

Figure 6: The AW-SOM A20 SODIMM Module [13].

The AW-SOM provides a full Linux image (uboot – kernel – rootfs) and a self-repository. The image can be transferred to the module through a USB port. From the repository the Linux kernel can be downloaded to be personalized.

The use of SoC with Linux brings some advantages that was sought on this work, like royalties free libraries and development tools, software packages and ease to deploy and maintain [6].

As the ability of integrating several features was specially sought on this work, it was explored the advantages of Linux such as the use of many protocols and applications like ethernet, TCP/IP stack, dhcp-client, ssh-server, file-system, etc. But "Linux is a General Purpose Operating System (GPOS) and as such is designed to provide good overall throughput rather than guarantee deterministic response for applications requiring real time" [16] [17]. A preemptive GPOS allows the scheduler switch to a higher priority task to take over from a lower priority task and can do the context switch so soon the process of higher priority has been released. That usually also leads to better deterministic response. [17].

To force the acquisition application to have the highest priority the nice command was used. Nice is a Linux application to run a program with modified scheduling priority, it adjusts the niceness of the program. It ranges from -20 (most favorable scheduling and no niceness) to 19 (least favorable and full of niceness) [18].

Changes in the scheduling priority brings an improvement to determinism (which will be seen on the performance test section) but, despite of the great advantages of Linux use, it is necessary to face the disadvantage of dealing with a big non-

deterministic jitter caused by Linux scheduling policies and by synchronization mechanisms, such as semaphores and mutexes[15][17]. The attempt to reduce the jitter to make this project viable is the main cause of this study.

There are two approaches to try to bring determinism to Linux, the "patching kernel approach" and the "micro-kernel approach"[16][7]. The "micro-kernel approach" [17] or "sub-kernel approach" [7] divides the typical features of a normal kernel in two layers. Where sub-kernel is a very small and minimal operating system that provides all the real-time functionalities such as scheduling and interrupts [16][17]. On the other, layer the typical kernel, run as a low priority process. This approach provides a good set of task synchronization and communication mechanisms [7].

The "patching kernel approach" is a patch of a diff file on the source code of Linux kernel in order to improve its real-time performance. A diff file keeps the difference of two files, when it is patched, these differences are applied to the original source code [5]. In this work, it was chosen the "patching kernel approach" in order to use the same kernel and drivers available in the open-source community. Also because of the advantage of comparing the same kernel version with and without real-time features. The specific patch was the RT-PREEMPT patch from http://rt.wiki.kernel.org project.

The RT-PREEMPT patch does the following change to make the Linux real-time [19]:

- Making in-kernel locking-primitives preemptible though reimplementation with rt-mutexes;
- Critical sections protected by i.e. spinlock_t and rwlock_t are now preemptible;
- Implementing priority inheritance for in-kernel mutexes, spinlocks and rw_semaphores;
- Converting interrupt handlers into preemptible kernel threads;
- Converting the old Linux timer API into separate infrastructures for high-resolution kernel timers plus one for timeouts, leading to user space POSIX timers with high resolution.

The next section shows the experimental test bed to evaluate de use of SoC with Linux OS to make data acquisition system for the two real time applications domains listed in section 3.

6. VALIDATION EXPERIMENTS

A test bed acquisition system was implemented to measure the acquisition time and jitter for two problems domain: (i) 1 kHz maximum signal frequency and (ii)5 kHz maximum signal frequency.

A Linux application was developed on SoC AW-SOM to send data by SPI acquisition command, after that the A/D converter sends back 128 bits of data (16x8 bits). The time interval between an acquisition request command and the next request is called "Measure Time" and it is logged in the Tektronics TDS2024B digital oscilloscope. A sample acquisition waveform is showed in Figure 7.

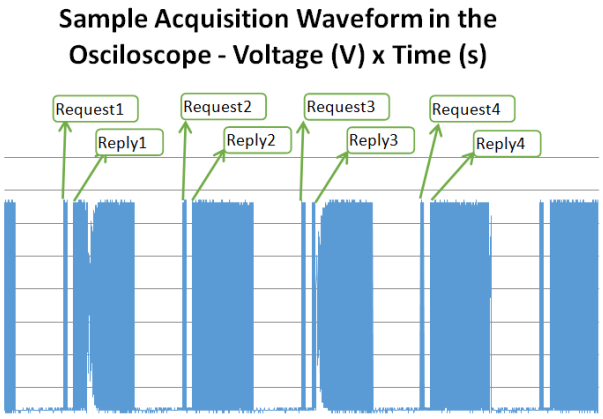

Figure 7: Sample acquisition waveform in Tektronics TDS2024B

To evaluate system's performance of RT-patch, two sets of experiments, each one with 200 samples, were executed. In the first test, the RT-patch was disabled and in the second, it was enabled again.

In both experiments, two applications were running in the SoC. The first application simulates a processing background stressing one core of the AW-SOM with several computation loops. The second is a desired data acquisition application. The Figure 8 shows a code fragment of the background application.

```
#define LENGTH 2000000
while(1){
    for(i=0,z=20;i<LENGTH;i++,z+=3){
        x[i]=i*z/55.456435;
        x[i]=213*i;
    }
    ...
}
```

Figure 8: Code fragment of background application.

By running this application, one CPU core was stressed near to 100%. The acquisition application runs in another core.

In the second round of experiments the RT-patch was enabled, the data acquisition application was executed by calling scheduled prioritization functions of the RT-patch API. The Figure 9 shows a fragment of code of this application, with max priority OS schedule set to 90.

```
#define PRIORITY 90

//set priority
param.sched_priority = PRIORITY;

//enable realtime fifo scheduling
if(sched_setscheduler(0, SCHED_FIFO, &param)==-1)
    return;

while(1){

    //wait untill next read
    clock_nanosleep(0, TIMER_ABSTIME, &t, NULL);

    //read SPI and format data
    buffer=(char *)spi_read(16,file);
    b[3] = (buffer[0]<<8)+buffer[1];

    //calculate next read
    t.tv_nsec+=interval;
    tsnorm(&t);
}
```

Figure 9: Code fragment of real time data acquisition application

The Figure 10 shows the acquisition time variation (jitter) for the two experiment sets in the first application domain (1 kHz maximum signal frequency). In this scenario, a SPI master clock was configured to 250 kHz with SPI slave A/D converter with 16 bits and 8 data channel.

In Figure 10 it's possible to see that the RT-patched time is equivalent to anon real time, the jitter and measure time is slightly higher in RT-patched. The table VII.1 summarizes the comparison of these two sets of experiments.

The Figure 11 shows the jitter of the two experiments in the second application domain (5 KHz maximum signal frequency).

In this scenario, a SPI master clock was configured to 2 MHz with SPI slave A/D converter with 16 bits and 8 data channel

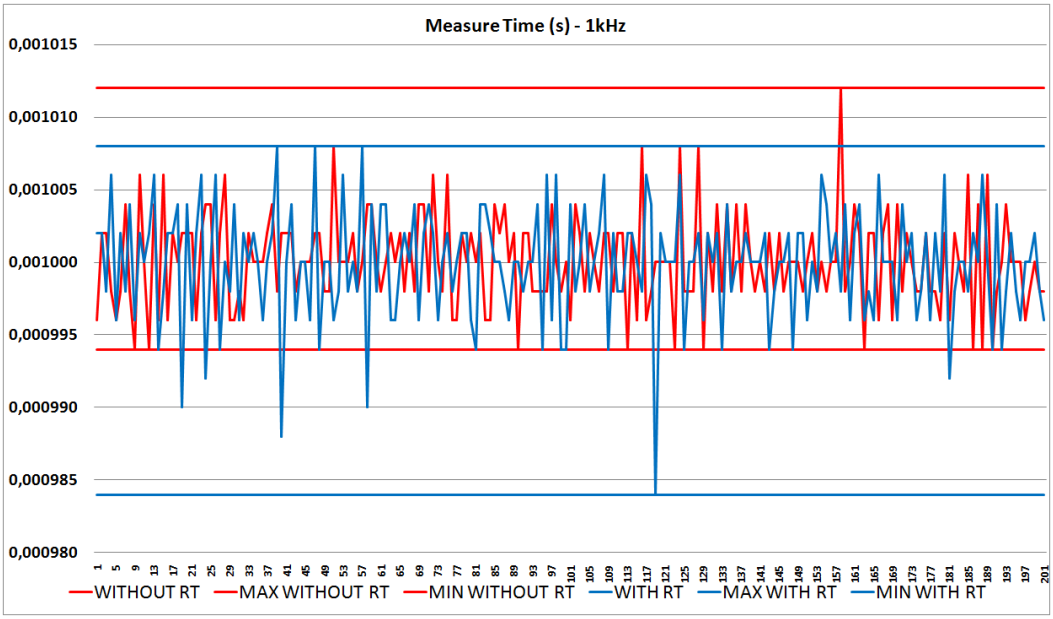

Figure 10: Measure Time for non-real time (blue color) and RT-patch experiments sets (red color) at 1 KHz maximum signal frequency.

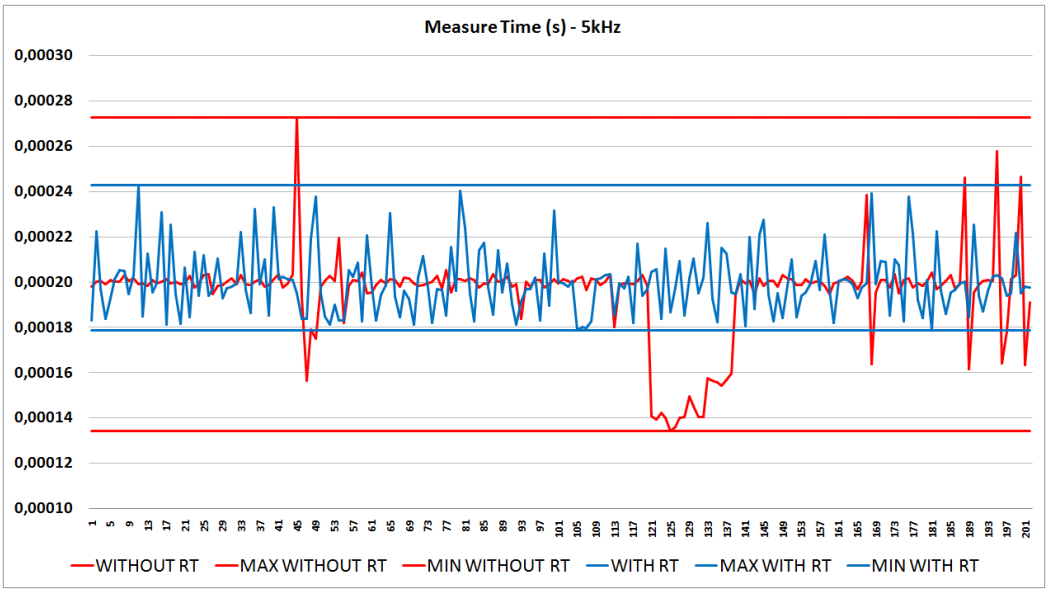

Figure 11: Measure Time for non-real time (red color) and RT-patch experiments sets (blue color).

The Table 1 shows that for 1 KHz maximum signal frequency the RT-patch is not necessary.

Having only the standard SO installed in the AW-SOM is efficient enough to acquire and process the desired signal.

Table 1: Experimentation comparison of non-real time and RT-path sets (1 kHz)

	Non RT (ms)	Jitter % Non-RT	RT-patched (ms)	Jitter % (RT)
Mean	1.000000		1.000000	
Std.dev.	0.003216		0.003836	
Max	1.012000	1.20	1.008000	0.80
Min	0.994000	-0.60	0.984000	-1.60
Jitter		**1.80**		**2.40**

In the second experiment, shown in Table 2,the global jitter with non-real time is 69.34%. With RT-path the global jitter is reduced to 32.9%.

Table 2: Experimentation comparison of non-real time and RT-path sets (5 kHz)

	Non RT (ms)	Jitter % Non-RT	RT-patched (ms)	Jitter % (RT)
Mean	0.194899		0.199939	
Std.dev.	0.019153		0.013983	
Max	0.272800	36.67	0.242800	22.87
Min	0.134400	-32.66	0.178800	-9.514
Jitter		**69.34**		**32.39**

The results show a significant reduction of the jitter by using the RT-patch, which means that the acquisition time is suitable for all application domains discussed in section IV. The higher jitter is not appropriate for FFT processing but some alternatives for mitigation can be used, like FFT interpolation [20].

7. CONCLUSIONS AND FUTURE WORK

The current work shows the comparison of an acquisition and post processing system in a SoC with a standard Linux and RT-Linux. It was delimited two problem domains, which were specified for 1 KHz and 5 KHz maximum signal frequencies. A test bed was implemented to acquire and measure 200 SPI reading samples of each acquisition system.

It was viable to meet the needs of the first problem domain (acquisition at 1 KHz rate) through a jitter of 2.4% on RT-Linux and 1.8% on the standard-Linux.

For the second problem domain (acquisition at 5 kHz rate) was verified a notable improvement of the jitter on RT-Linux, which was of 32.39% against the 69.34% of standard-Linux, still the jitter is too high for the target application.

As future work, it is proposed the use of a Field-programmable gate array (FPGA) for the acquisition part, as the FPGA can be almost fully deterministic, with a negligible jitter and the possibility of high acquisition rates. The SoC can still be used for the post-processing and transmission part. The block diagram of this proposition can be seen on Figure 12.

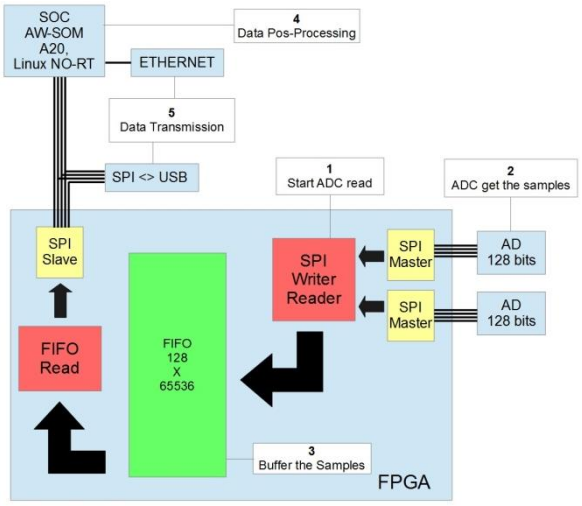

Figure 12: Block diagram of the acquisition system with FPGA

8. REFERENCES

[1] Rajesvari.R, Manoj.G, Angelin Ponrani. M. "System-on-Chip (SoC) for Telecommand System Design". International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 3, March 2013.

[2] "ARM NEON technology brief". Available online at: http://www.arm.com/products/processors/technologies/neon.php

[3] Pacheco. L, 2013: "Desenvolvimento De Sistema Distribuído Microprocessado Para Diminuição De Tempo De Transitório De Bancadas De Ensaio De Compressores". Available online at: https://repositorio.ufsc.br/bitstream/handle/123456789/103561/317173.pdf

[4] "AQX700+ Product Brochure". Avaible online at: http://www.aqtech.com.br/produtos/AQT700/Folder_AQT700.pdf

[5] Rosted, Steven. Hart, Darren V. "Internals of RT Patch", In Proceedings of Linux Simposium, June 2007, Ontario, Canada, Pages 161-172.

[6] Betz, Wolfgang. Cereia, Marco. Bertolotti, Ivan C. "Experimental Evaluation of the Linux RT Patch for Real-Time Applications", In Proceedings of IEEE Emerging Technologies & Factory Automation, 2009, ETFA 2009.

[7] Bruzzone, G. Caccia, M. Ravera, G. Bertone, A. "Standard Linux for Real-Time Robotics and Manufacturing Control Systems". Robotics and Computer Integrated Manufacturing. Elsevier. 2009. Pages 178-190.

[8] Oikawa, S. Tokuda, H. "User-Level Real-Time Threads". In Proceedings of IEEE of Real-Time Operating Systems and Software. RTOSS ´94. 1994. Seatle, WA.

[9] "A20 Processor Features and Highlights" Available online at: http://www.allwinnertech.com/en/clq/processora/A20.html

[10] Grusin M., at Sparkfun, 2014: "Serial Peripheral Interface (SPI)", available online at: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all.

[11] "ADS8568: 12-, 14-,16-Bit, Eight-Channel, Simultaneous Sampling ANALOG-TO-DIGITAL CONVERTERS Data Sheet", 2011 Available online at: http://www.ti.com/lit/ds/symlink/ads8568.pdf.

[12] "ADAS3022: 16-Bit, 1 MSPS, 8-Channel Data Acquisition System Data Sheet" Available online at: http://www.analog.com/static/imported-files/data_sheets/ADAS3022.pdf

[13] "AW-SoM A20 Dual Core SODIMM Module Datasheet", Available online at: http://docs.aw-som.com/public/products/AWSOM-A20-SD_DATASHEET.pdf

[14] "FEX Guide - linux-sunxi.org", august, 2014. Available online at: http://linux-sunxi.org/Fex_Guide.

[15] [15] "Linux-sunxi.org - What is sunxi", august, 2014. Available online at: http://linux-sunxi.org/Main_Page

[16] Arthur, V., Emde, C,. Guire, N. M., 2007: "Assessment of the Realtime Preemption Patches (RT-Preempt) and their impact on the general purpose performance of the system", Real-Time Linux Workshop 2007, Linz, Austria

[17] Vun, N., Hor, H. F., Chao, J. W., 2008: "Real-time Enhancements for Embedded Linux", in 14th IEEE International Conference on Parallel and Distributed Systems, pages 737-740.

[18] "nice(1) Linux man page", 2010. Available online at: http://linux.die.net/man/1/nice

[19] Fu, L., Schwebel, R., 2014: "RT PREEMPT HOWTO". Available online at: https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

[20] Wen, He et all. "Simple Interpolated FFT Algorithm Based on Minimize Side lobe Windows for Power-Harmonic Analysis". In IEEE TRANSACTIONS ON POWER ELECTRONICS, VOL. 26, NO. 9, SEPTEMBER 2011.