A real-time system based on FPGA to measure the transition time between tasks in a RTOS

Lidia H. Shibuya, Sandro S. Sato, Osamu Saotome, Fernando G. Nicodemos

Instituto Tecnológico de Aeronáutica – ITA
Praça Marechal Eduardo Gomes, 50 - Vila das Acácias - CEP 12.228-900
São José dos Campos – SP – Brasil

lishibuya@uol.com.br, shoiti@ita.br, osaotome@ita.br,
fgnicodemos@terra.com.br

Abstract. *This paper describes the conception, the design, the development, the implementation and the experimental results of a real-time electronic instrumentation system that measures accurately the transition time between tasks running under a Real Time Operating System (RTOS) supervision.*

1. Introduction

The use of Real Time Operating System (RTOS) has been increasing over the years in embedded system design. A RTOS is designed to meet rigorous time constraints. Nevertheless, selecting a RTOS for an embedded application can be a complex process especially when it is to be used in critical applications. For these critical applications, e.g. space applications or medical applications, the use of hard-real time system is a requirement [Burns, 1991]. Using Kopetz's definition (2002): *"A real-time computer system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced."*

One important feature of a RTOS is the fact that the scheduler performs more deterministic scheduling of tasks. The scheduling policy used by the RTOS scheduler gets impacts in the overall system performance. The use of an RTOS does not guarantee that the system will always meet the deadlines, once it also depends on how the overall system is designed.

It means that every single task shall have a well defined time to be processed and to output a result. It also suggests that the inherent transition time to switch among tasks should be considered during the development of the software for hard-real time systems. Due to the several possible internal processors architectures, including for embedded applications, that implements for example, different levels of cache memory and depths of pipeline, the performance of the processor can change. In addition, the performance of different RTOS can vary enormously.

Many RTOS have presented some specific software components to measure their characteristics. These components are usually software routines attached to the system, as for example the debug capacity of the RTEMS gdb compiler [OAR, August 2008]. When the performance is measured using these attached software routines, the results can differ from each RTOS, because each RTOS usually implements different

functions, as a result, the data gathered from different RTOS components cannot be compared.

Efforts have been made to measure the performance of RTOS as shown in Anh and Tan (2009), Martinez et. al (1996), Kar and Porter (1989) and Sacha (1995). These RTOS performance analyzers are based on benchmarking method of frequently used system services. The performance of an RTOS is measured verifying if the task complies with its deadline. As there are various types of applications with each having very different requirements, benchmarking against any generic applications will not be reflective of the RTOS strengths and weaknesses.

Another approach is presented by Lamie and Carbone (2007), where they provide some APIs which can be used in different RTOS to verify the performance. But in this proposal, there is software interference on the performance measurements. According to the target, some changes also may be required. The JEWEL tool proposed by Lange et. al (1992) was developed to analyze the performance of distributed systems. Also Wybranietz and Haban (1988) proposed integrated tool for monitoring distributed systems continuously during operation.

The main objective of the test environment proposed in this paper is to measure absolute time characteristics of a RTOS, that influences performance, for embedded applications, without the insertion of overhead, neither including changes on the embedded software. This approach allows time comparisons that may be helpful to analyze different RTOS performances in different processors. To show the proposed real-time system based on FPGA to measure the transition time between tasks in a RTOS, this work contains 5 sections. In Section 2 an overview of the proposed solution is presented, introducing the elements used on the implementation. Section 3 presents in details the system implementation. In Section 4, experimental results are described. The paper ends with a conclusion, acknowledgment, and references.

2. Proposed Solution

This paper proposes a real-time system based on FPGA to measure the transition time between tasks in a RTOS. The use of FPGAs allows many processes to start at the same time [D'Amore, 2007] which creates an efficient tool to analyze the performance of a device running RTOS, and there is no overhead due to software initialization or software modifications that usually occur on the debuggers.

In order to determine the transition time between the tasks in a RTOS, the first step was to decide which device should be chosen as the target. In this case, the Device Under Test (DUT) is an ERC32 development kit [ATMEL, August 2003]. The ERC32 is a radiation-tolerant 32-bit RISC processor developed for space applications, [ATMEL, August 2004] and [ATMEL, March 2005].

The second step was to choose a RTOS. The chosen DUT accepts the real time operating system RTEMS (Real-Time Executive for Multiprocessor Systems) [OAR, 2006].

The third step regards to the choice of the task model to be implemented on the DUT. The task should produce stimulus for the data acquisition module, where these stimulus could be read and stored in a temporary memory. Once the data acquisition

module reads the stimulus, this data were acquired by dedicated software, in order to analyze the data and show the results in graphical format.

The general architecture of the Project is shown in Figure 1. The dedicated software, running on the PC, collects and provides the analysis. The FPGA was configured to monitor the GPI (General Purpose Interface) collecting data and sending it to the Personal Computer (PC). The DUT was configured to run the RTOS RTEMS to generate signals to the GPI.

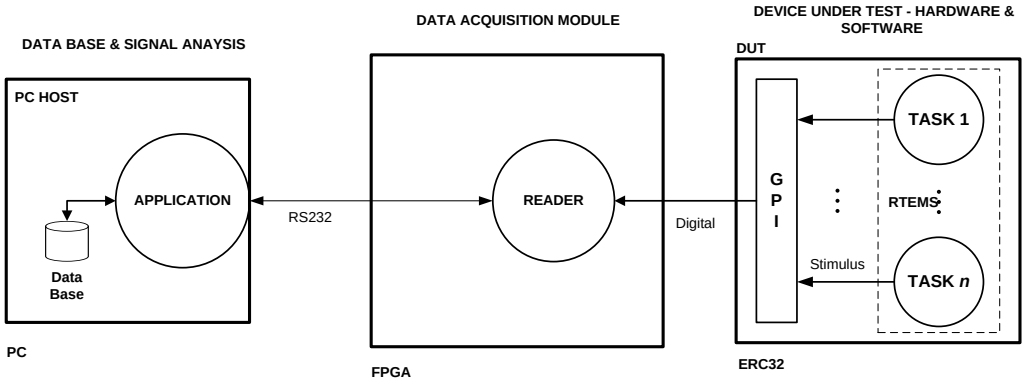


Figure 1. Hardware General Architecture

3. System Implementation

The proposed solution shown in section 2 was organized and implemented in three components: System Under Test, Data Acquisition Module and Software Analysis Module.

The system under test is composed by an ERC32 processor (SPARC V7 architecture) running the RTOS RTEMS. The RTEMS was configured with four tasks that run equivalent routines.

The data acquisition module was implemented on a FPGA, which continuously monitors the signals from the DUT and when it detects that the task changed the Data Acquisition Module, stores the information that will be processed after the software analysis module.

The analysis software was developed in Visual Studio. It receives all the data collected from the acquisition module and decodes them in order to generate graphics that simplifies the view and the analysis process.

3.1 Device Under Test - DUT

The Device Under Test (DUT) is an ERC32 processor. This processor is used mostly in aerospace application. It is developed based on SPARC V7 architecture. The tools required to develop RTEMS codes and to connect to the development kit (DUT) are based on Linux. This project uses Fedora 7 platform.

The ERC32 processor has a GPI (General Purpose Interface) that is used to signalize when a task is preempted. This GPI interface is composed by a register of 8 bit which can be configured as inputs or outputs. To identify the GPI, the GPI[0~7] notation is used, and the GPI[0], GPI[1] through GPI[7] is used to identify only one interface. For this work the entire GPI was configured as outputs.

The running software was developed in C language. The software was build using RTEMS development tools. The main characteristics of the implemented tasks and the RTEMS configuration are:

- Four equivalent tasks with the same priority level were implemented using RTEMS.
- Each task was implemented as an infinite loop that produces a periodic square wave in the GPI.
- The RTEMS was configured to use round-robin scheduling with a fixed time slice.

Figure 2 shows the general idea of the task model used in the software architecture. The cooperative context switching task model proposed by Lamie and Carbone (2007) was chosen.

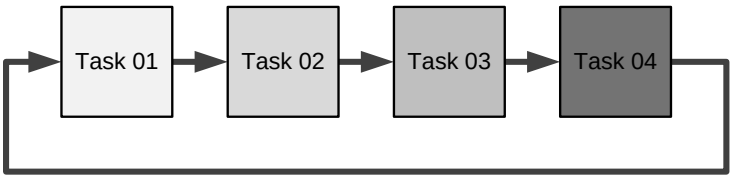


Figure 2: Task model

Figure 3 shows an overview of the system under test and the output expected on the GPI.

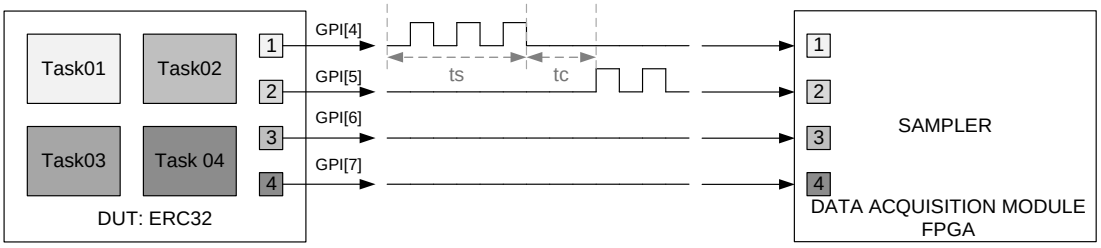


Figure 3: General overview of the device under test and the output on GPI[4~7]

In Figure 3, on the left, each task produces a square wave in one GPI output. On the right, the output expected on the GPI, when the scheduler achieves the time set on the time slice, represented by (ts). There is transition time, represented by (tc), between the Task01 and Task02. Along the time, after each time slice, there is a transition time (tc) between the tasks. It is important to emphasize that each task produces a square wave on a specific GPI output and keeps all the other GPI interfaces in zero, e.g, task01 produces a square wave on GPI[4] output, task02 produces a square wave on GPI[5], task03 outputs to GPI[6] and task04 outputs to the GPI[7]. The period of the square wave can be changed through an internal counter on the task routine. The RTEMS configuration parameters were configured as follows.

```
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_MAXIMUM_TASKS             5
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_EXTRA_TASK_STACKS
(4*RTEMS_MINIMUM_STACK_SIZE)
#define CONFIGURE_USE_MINIIMFS_AS_BASE_FILESYSTEM
#define CONFIGURE_MICROSECONDS_PER_TICK     10000 // the default is 10000
#define CONFIGURE_TICKS_PER_TIMESLICE       50    // the default is 50
```

The piece of code shows task01 implementation. It is important to remember that the four tasks have the same priority level and perform the same code. The only difference among the tasks is the fact that each task produces a square wave in one specific GPI, from GPI[4] through GPI[7].

```
/* Task 01 - GPIO 4 */
rtems_task Task_GPIO4(rtems_task_argument unused) {
  unsigned char count = 0;
  for (;;) {
    for (count=0; count<16;) {
      *leds = 0x10; /* GP4 <= '1' */
      count++;
    }
    for (count=0; count<16;) {
      *leds = 0x00; /* GP4 <= '0' */
      count++;
    }
  }
}
```

Figure 4 depicts the GPI output on GPI[4] and GPI [5] after the implementation on the DUT (ERC32). The measured transition time among the tasks on the scope was about 135us.

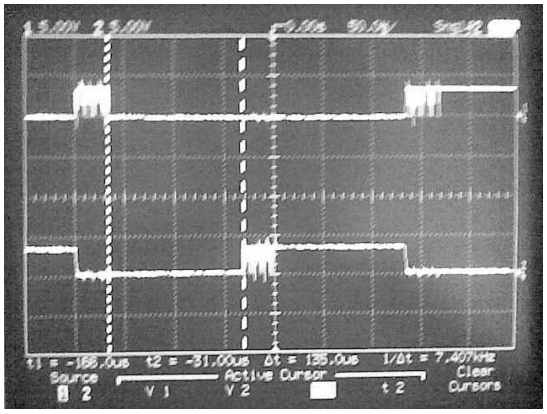


Figure 4: Output on GPI measured on the scope

3.2 Data Acquisition Module

The Data Acquisition Module makes use of an ALTERA FPGA. The development environment for the FPGA is composed by the Quartus® II software [ALTERA, October 2007] and a Stratix II development kit [ALTERA, May 2007] and [ALTERA, January 2007].

The FPGA was programmed with VHDL and ALTERA Mega-Function Blocks. The Data Acquisition Module can be divided in two functional blocks: the Sampler and the Communication block.

Figure 5 shows the block diagram of the Data Acquisition Module implemented in FPGA. The communication controller transfers all the data storage to the PC.

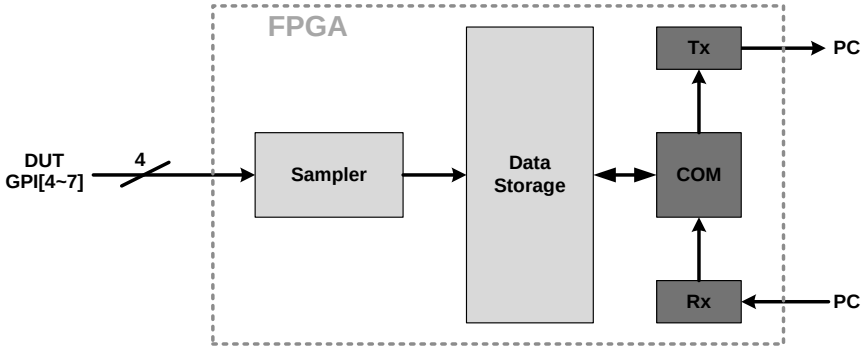


Figure 5: Overview of FPGA implementation

The Sampler Block performs the monitoring and analyzing function. This block has a free-running counter (ticks counter). This free-counter tick value is stored when the hardware detects a task transition on the GPI. At this moment, the minimum historical data are stored, to permit infer the transition time between the tasks. To monitor the GPI, the data acquisition module reads the GPI as an 8 bits register. The four low significant bits (GPI[0~3]) are always read as zero.

Figure 6 represents the moment of a transition time. The samples occur on the rising edge of the data acquisition module clock. A First In First Out (FIFO) register stores the three transition in the GPI that represents the transition time, ticks t_5, t_7 and t_{14}. The data are stored in the FPGA memory and when the FPGA receives a command from PC, the FPGA dumps the memory to PC.

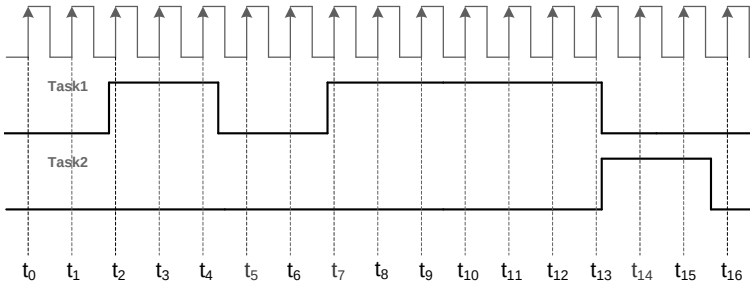


Figure 6: Example of a GPI output and the sample times for transition time analysis

The Communication Block was developed to send data to PC and to receive commands from PC. This block is basically a serial interface with configurable baud rate.

The data acquisition module presents some inherent characteristics due to the hardware description used in the implementation and also due to the task behavior, a periodic square wave. These characteristics contribute in a non-desirable manner on the transition time.

Figure 7 shows the condition where the non-desirable additional contribution has the minimum value. In this case, the data acquisition module stores the time presented by tm (ts – time slice, tc – transition time between the task). At the moment that the square wave toggles its status from zero to one, the transition time (tc) starts. The next task starts the square wave from zero to one. In this case, the transition time measured (tm) is almost the same as the transition time between the tasks (tc).

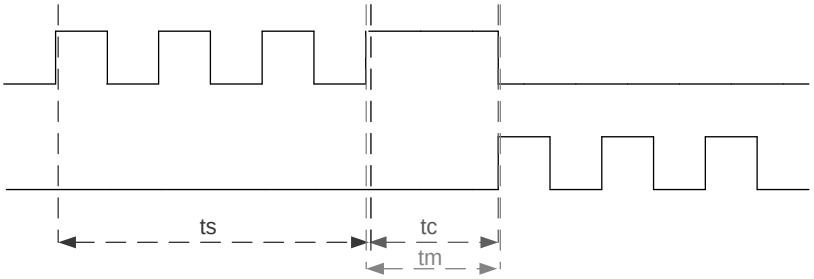


Figure 7. Minimum non-desirable contribution on the transition time measured by the data acquisition module

There is, although, one condition where the non-desirable contribution has the maximum value. This condition is presented on Figure 8. In this case, the task toggles its GPI from one to zero and on the moment before of switching again to one the transition time (tc) starts. If the next task starts from zero, the data acquisition module is not able to store the transition time until the task switches to one. The picture shows the transition time measured by the data acquisition module. As one can see, the non-desirable contribution in this case is about one period of the square wave.

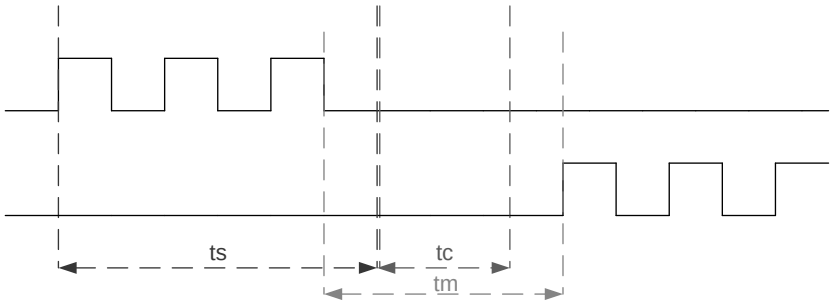


Figure 8. Maximum non-desirable contribution on the transition time measured by the data acquisition module

The non-desirable contribution can be reduced increasing the frequency of the square wave for each task, reducing consequently the period, as shown in Figure 9. Once the data acquisition module works with a clock more than 1000 times faster than the frequency of the square waves on GPI, it's possible to increase the square wave frequency without loss of accuracy of the measurements.

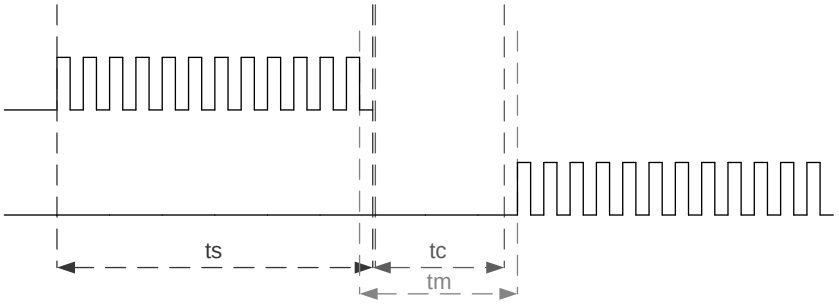


Figure 9. Square wave with higher frequency reduces the worst condition for the data acquisition module.

3.3 Analysis Module

The main function of the Analysis software is to receive, to verify and to generate the graphs. The development tools used for analysis module are composed by the Visual Studio .NET Framework to acquire the data and the Dundas software to the graphical interface [Dundas, 2009].

The PC host, with the developed analyzer software uses the serial interface. The transfer rate, frame format (start bit, parity and stop bit) must be configured, using also the developed software. The Figure 10 shows the developed software window.

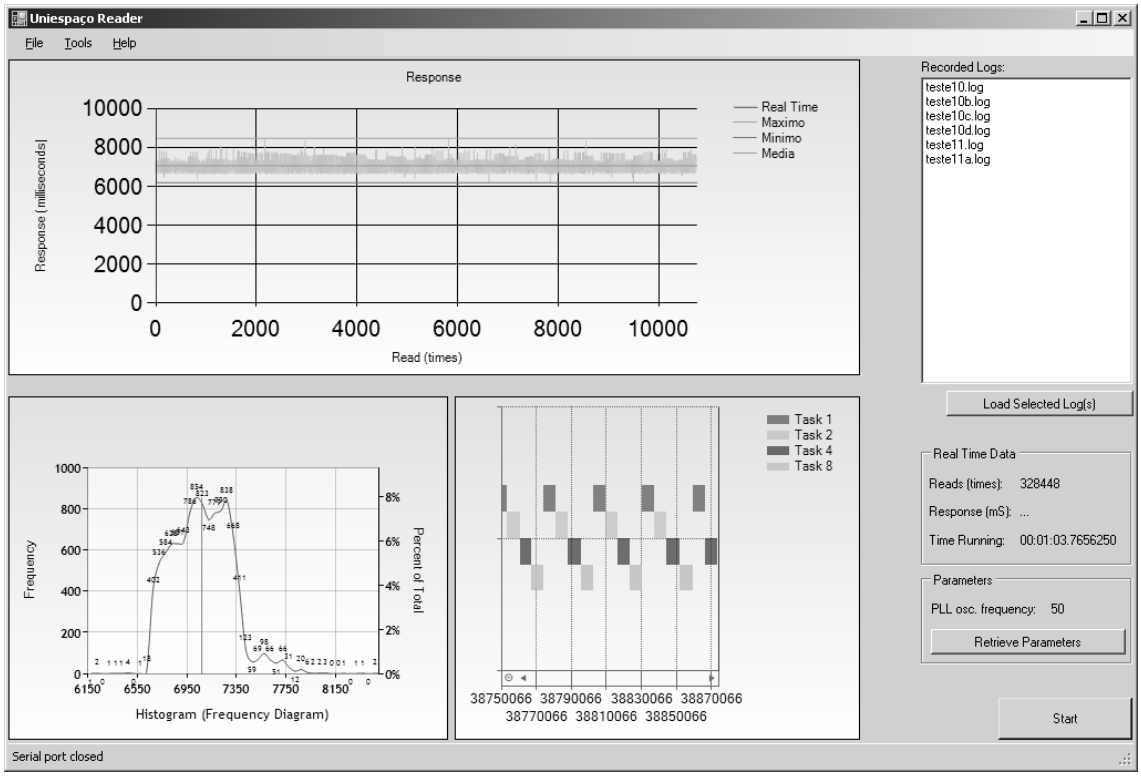


Figure 10. Data Acquisition and Analysis software

Figure 10 shows the developed software window. On the top, the x-axis represents the number of samples acquired and the y-axis represents the measured transition time (number of ticks) for each cooperative context switching. In this graph there are also three lines representing the maximum value, the minimum value and the average value resulting from the analysis. On bottom-left, a histogram is presented to show that the measure has an average behavior. On bottom-right the Gantt graph represents for how long one task had been running on the processor. The purpose of this graph was to show the round-robin scheduler.

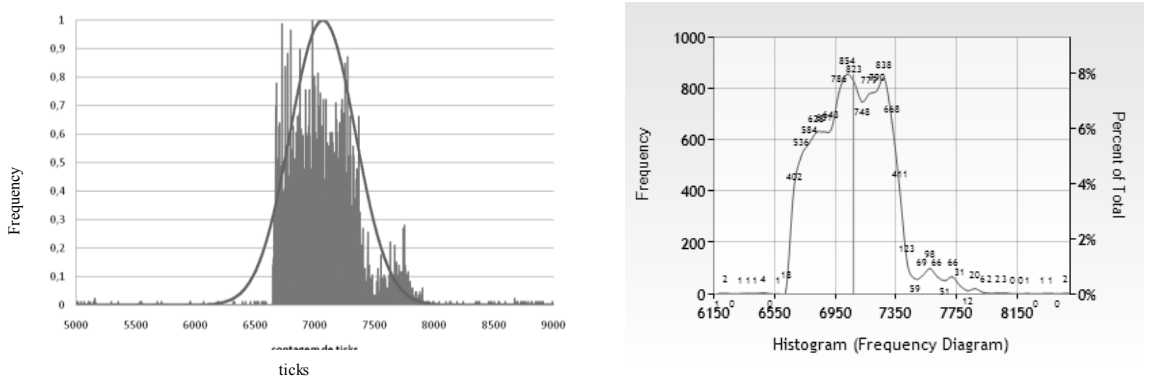
4. Experimental Results

The results depicted in this section were acquired using the developed software. There is also a comparison among the data acquired by the developed software and a manual analysis of the acquired data. The data were acquired using the parameters:

#define CONFIGURE_MICROSECONDS_PER_TICK 1000

#define CONFIGURE_TICKS_PER_TIMESLICE 5

The FPGA was configured to work with a clock of 50MHz, the number of ticks (in average) measured using the analysis software was about 7000 ticks, which results in a transition time of about 140us (7000(ticks)/50000000Hz).



(a) Ticks (b) Histrogram

Figure 11. Results one acquisition: (a) ticks and (b) Histogram

Figure 11a, on the right, depicts the results of one acquisition from the PC host software and the Figure 11b, on the left, depicts the results of the same acquisition, but analyzed manually, using the Excel. As one can see, both pictures have the same shape. The attenuation on the small picture can be explained once the developed software for the PC host considers a range of values (ticks) to plot the graph and the manual analysis is plotted using the ticks (x axis) incremented one by one. These transition time values were compared with the expected value of 135us, observed on the oscilloscope in the lab.

As it was explained on Section 3.2, the results presented in both graphs in Figure 11 show the measurements considering all the possibilities of additional contributions due to the task implementation. The difference between the maximum and minimum value taking into account the average value is about ±395 ticks, which represents a range of ±7,9us.

5. Conclusion

The results of this implementation show that it is possible, using the proposed real-time system based on FPGA, to measure the transition time between tasks in a running RTOS. These measurements can be helpful to model more precisely the tasks that one RTOS shall perform and to estimate with a better level of accuracy the real time performance accuracy of the software, particularly, contributing to the study of safe critical applications.

6. Acknowledgements

We want to thanks AEB and UNIESPAÇO program for the financial support of the project.

7. References

ALTERA (January, 2008) “Stratix II Device Handbook, Volume 2 – SII5V2-4.4”,January.

ALTERA (May, 2007) “Stratix II Device Handbook, Volume 1 – SII5V1-4.3”, May.

ALTERA (October, 2007) “Introduction to the Quartus® II Software – Version 7.2”, October.

ATMEL (August, 2003), “Evaluation Board TSC695 – User Guide – Rev. 4139F–AERO–08/03”, http://www.atmel.com, April.

ATMEL (August, 2004), “Low-Voltage Rad-Hard 32-bit SPARC Embedded Processor – Datasheet – Rev. 4204C–AERO–08/04”, http://www.atmel.com, April.

ATMEL (March, 2005), “TSC695 SPARC V7 Processor (ERC32) – Development Tools – Rev. 7503A-AERO–03/05”, http://www.atmel.com, April.

Burns, A, (1991) “Scheduling hard real-time systems: a review”, Software Engineering Journal, IEEE. p.p 116-128

D’amore, R. (2005), “Vhdl – Descrição e Síntese de Circuitos Digitais”, LTC, 275 pages.

Dundas (2010), “Dundas Dashboard Documentation”, http://support.dundas.com/Dashboard1.Documentation.ashx, April.

Kar, R. P., Porter, K., (1989) “Rhealstone: A real-time benchmarking proposal”, Dr. Dobb's Journal

Lamie, W., Carbone, J. (2007) “MEASURE YOUR RTOS'S REAL-TIME PERFORMANCE”, www.embedded.com

Lange, F., Kröger, R., Gergeleit, M., (1992) "JEWEL: Design and Implementation of a Distributed Measurement System", IEEE Trans. on Parallel and Distributed Systems, Vol. 3, No. 6, pp. 657-671

Martinez, A., Conde, J. F., Vina, A., (1996) “A comprehensive approach in performance evaluation for modern real-time operating systems”, Proceedings of the 22nd EUROMICRO Conference, pp. 61,

OAR (August, 2008), “Getting Started with RTEMS - Edition 4.8.1, for 4.8.1”.

OAR, (2006), “RTEMS 4.8.1 On-Line Library”, http://www.rtems.org/onlinedocs/releases/rtemsdocs-4.8.1/share/rtems/html/, April.

Sacha, K. M., (1995) “Measuring the real-time operating system performance”, Seventh Euromicro workshop on Real-time systems proceedings, Odense, Denmark, pp. 34–40

Wybranietz, D., Haban, D. (1988) "MONITORING AND PERFORMANCE MEASURING DISTRIBUTED SYSTEMS DURING OPERATION", ACM SIGMETRICS Performance Evaluation Review, Volume 16, Issue 1, pp 197 – 206