Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 3, Number 1, pages 10–12, January 2014

Test Methodology for Real-Time Operating System

Siaw Chen Lee, Soon Ee Ong
IC Design Engineering
Altera Corporation (M) Sdn Bhd
Bayan Lepas, Penang, 11900 MY
sclee@altera.com, seong@altera.com

Noohul Basheer Zain Ali
Dept. of Electrical and Electronics
Universiti Teknologi Petronas
Tronoh, Perak, 31750 MY
noohul.z@petronas.com.my

***Abstract*—There exist many varieties of Real-Time Operating System (RTOS) in the market, most of them are software based while some are hardware based. During the selection of RTOS, we often need to know the performance of RTOS to ensure it meets the requirements of the real-time system we are designing. Not all RTOS provides specification or data that is useful for consideration or calculation. This paper proposed a set test methodology for designer to quickly measure the performance of a RTOS. The test methodology proposed in this paper covering parameters that are practically useful for real-time system designer to be used for RTOS selection judgment or improvement. These parameters includes computing process overhead, task level interrupt latency, intertask communication & synchronization latency and performance jitter.**

***Keywords*-Real-time Operating System; RTOS; Test Methodology**

I. Introduction

One of the key factor that differentiate Real-Time Operating system from general computing operating system is the deterministic of response timing [2]. Real-time operating system must have a deterministic response time (i.e. latency) to ensure the quality of service [1]. Other than latency, overhead spent by RTOS processes are also important. The overhead should be kept minimum for less wastage of computing power. The latency timing and overhead timing should also been kept in small variance (i.e. jitter of latency and overhead time) to maintain the time deterministic.

This paper describes the test methodology that can be quickly and easily implemented in RTOS for measurement of performance critical parameters, namely computing process overhead, task level interrupt latency, intertask communication & synchronization latency and performance jitter.

The remaining sections of this paper are organized as follows. Section II briefly introduces the related works in RTOS measurement. It is followed by Section III that explains the test methodology. The described test methodology is then implemented on a software based RTOS ,μC/OS-II, and hardware based RTOS, SEOS, for demonstration at Section IV. Finally, Section V concludes the paper.

II. Related Works

Tsoukarellas, Manthos A. and colleagues published a systematic strategy for testing of real-time operating system [4]. The test consist of black-box testing and white-box testing. Flows on test cases creation for coverage on all RTOS functions are presented, including task related testing and semaphore related testing.

On the other hand, Wegener and groups also developed a classification-tree method for RTOS testing [5]. It turns functional test case design into a process comprising several structured and systematized parts to make it easy to handle and documented. The classification-tree method was improved version of category-partition method defined by Ostrand and Balcer [6]. The basic idea is to partition separately the input domain of the test object under different aspects, assessed as relevant for the test, and then to recombine the different partitions to form test cases.

To bring the test methodology further, Dino's team developed a language called TRIO for automated derivation of functional test cases for real-time system [7]. This language can be used on RTOS as well. The TRIO language is an extension of classical first-order temporal logic with respect to a current, implicit time instant. Unlike classical temporal logic, TRIO allows the specifier to express strict timing requirements by means of two basic operatorsFutr and Pastwhich refer to time instants whose distance, in the future or in the past, is specified precisely and quantitatively.

All works shown above defines RTOS testing in great detail with extensive coverage. TRIO also provide a good tool for automated testing. Despite of the completeness in test coverage, all above mentioned test methodology required extensive of time and effort to implement. For the case of early RTOS assessment, system designer may sometimes want a quick test on the RTOS performance. With that as basis, system designer can then pin point the RTOS that meets system requirements. Full test coverage will then being implemented after the system development phase.

III. Test Methodology

As mentioned in previous section, system designer may sometimes want a quick test methodology to assess the performance of RTOS to select or to confirm that the performance meets system requirements. This paper identify four critical parameters for RTOS performance testing. These parameters are able to give the system designer a good

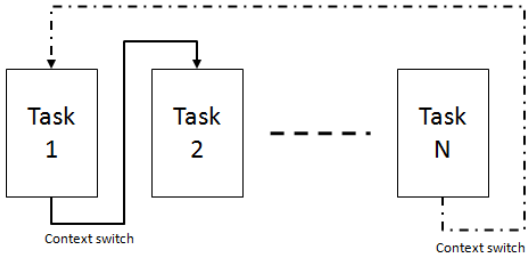

Figure 1. Computing Task Loop

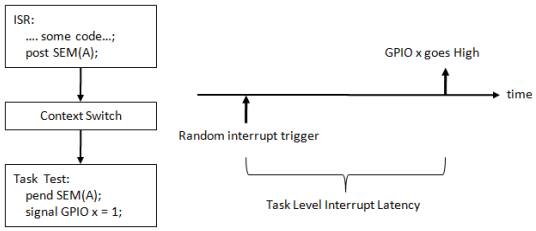

Figure 2. Task Level Interrupt Latency

pictures on the overall performance of the RTOS. With the same data, system designer can also quickly determine if the RTOS is meeting the requirements of the system. The four parameters are computing process overhead, task level interrupt latency, intertask communication & synchronization latency and performance jitter.

A. Computing Process Overhead

Computing Process Overhead is the overhead time consumed by RTOS on top of the actual computing process in the system. In order to measure computing process overhead, A task loop can be created with a fixed number of test tasks. The length of benchmarking task should be in adequate length of time. Standard benchmarking package such as whetstone benchmarking or Dhrystone benchmarking are recommended candidate to be the test tasks as it would provide adequate exercise on the computing process with reasonable amount of time. In this test, fixed iterations of loop are executed, as illustrate in Figure 1. Actual process time should be taken from the beginning of test task and the end of the test task as the differential time, DT. This Total time, T, taken to execute all the loops is also measured. The computing process overhead can be calculated by comparing the total time against the differential time, as shown in formula 1.

$$Overhead = T - (DT * Numberof Loops) \qquad (1)$$

B. Task Level Interrupt Latency

Task Level Interrupt Latency is to measure the responsiveness of the task toward an interrupted event. In other words, it measure the latency from the occur of event until the starting of processing in the correspondent task. For this test, an external interrupt source is setup to trigger an interrupt service routine randomly. The interrupt service routine is coded to post a semaphore, in which will cause context switch to switch to the task that handles this external interrupt event. Upon receiving of the semaphore, the task will force a GPIO pin to high. Time latency is measured at the point when external interrupt get triggered to the point when task trigger the said GPIO, as shows in the timing diagram at Figure 2.

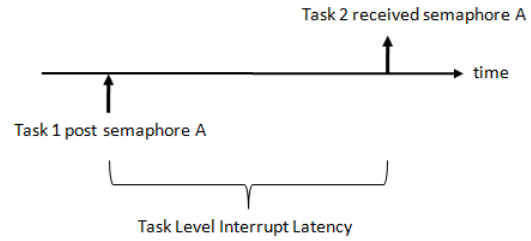

Figure 3. Intertask Communication & Synchronization Latency

C. Intertask Communication & Synchronization Latency

Intertask Communication & Synchronization Latency is the parameter to tells the latency between a task posting a signal till the reception of another task. For this test, two tasks are created whereby one task will post a semaphore/message while the other task is pending to receive the semaphore/message. Latency is measured for time taken between the posting and receiving of the semaphore/message as shows in the timing diagram at Figure 3.

D. Performance Data Jitter

With enough data set collected from the above three parameters, the performance jitter can be computed by calculating the variance of data using the equation 2. σ is the jitter number, while n is the number of data set.

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n} \qquad (2)$$

IV. Test Methodology Implementation

A. Test Platform

Two RTOS was selected to conduct the performance test to illustrate the implementation of the described test methodology. The software based RTOS selected is μC/OS-II [1] and hardware based RTOS selected is SEOS [3]. An Altera Cyclone-II FPGA development board with NIOS-II soft core processor, flash memory, SDRAM and I/O module is used for test platform development. Avalon interconnect bus is used as system interconnect. The test is separated into two sections. First section measure the performance of SEOS and second section replace SEOS with μC/OS-II.

Table I
PERFORMANCE COMPARISON RESULT

Measurement Parameter	Computing Process Time (μs) SEOS	μC/OS-II	Performance Different
Computing Time (loops)	13,405,265	19,608,805	31.64%
Task Level Interrupt Latency	49.967	319.577	83.52%
Inter-task Comm. and Sync. Latency	64.12	228.46	71.93%

Table II
JITTER PERFORMANCE COMPARISON

	Performance Jitter (%) SEOS	μC/OS-II
Computing Overhead	0.00014%	0.0086%
Task Level Interrupt Latency	1.590%	25.365%
Inter-task Comm.& Sync. Latency	0%	0.103%

B. Result

Table I denotes the comparison of the three measured parameters. SEOS computing time is 31.6% less compares to μC/OS-II which indicated that it has lower system overhead. The task level latency of SEOS is of 83.5% lesser compares to μC/OS-II. As for intertask communication and synchronization latency, SEOS is 71.9% lower than μC/OS-II. The performance jitter is also computed based on 20 sets of data collected. A hardened RTOS is suppose to have performance advantage over software based RTOS. The experiment is consistent with the expected result.

V. CONCLUSION

This paper presented a test methodology for RTOS that is able to allow the system designer to easily and quickly obtained practically useful performance data of a RTOS. This is particularly useful when system designer needs to quickly assess the performance of a RTOS to decide on RTOS selection or to determine if the RTOS is meeting requirements.

REFERENCES

[1] J. Labrosse, *MicroC/OS-II: The Real-Time Kernel*. R&D Books, Lawrence, KS, 1999.

[2] P. Laplante, S. Ovaska, *Real-Time System Design and Analysis: Tools for Practitioner*. Wiley-IEEE Press, 2011.

[3] Ong, Soon Ee, Siaw Chen Lee, and Noohul Basheer Zain Ali, *"Hardware implemented real-time operating system."* Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays. ACM, 2013.

[4] Tsoukarellas, Manthos A., Vasilis C. Gerogiannis, and Kostis D. Economides, *"Systematically testing a real-time operating system."* Micro, IEEE 15, no. 5 (1995): 50-60.

[5] Wegener, Joachim, et al, *"Systematic testing of real-time systems."* 4th International Conference on Software Testing Analysis and Review (EuroSTAR 96). 1996.

[6] Ostrand, T., and Balcer,M, *"The Category-PartitionMethod for Specifying andGenerating Functional Tests."* Communications of the ACM (1988), 31 (6), pp. 676-686.

[7] Mandrioli, Dino, Sandro Morasca, and Angelo Morzenti, *"Generating test cases for real-time systems from logic specifications."* ACM Transactions on Computer Systems (TOCS) 13.4 (1995): 365-398.