International Journal of Grid and Distributed Computing
Vol. 9, No. 9 (2016), pp.395-402
http://dx.doi.org/10.14257/ijgdc.2016.9.9.33

Architecture of Task Manager for Real Time OS Explaining Real Time Operating Systems Issues

Javed Ahmad Shaheen MSCS

Virtual University of Pakistan
javedmatyana@yahoo.com

Abstract

Complications of embedded applications are increasing. Within due time delivery to the market is also a pressure. So, the demand and use of real time operating systems is also increasing. However, it is an redundant fact that the Real Time Operating System (Real Time OS) can significantly degrade the performance. To face performance degradation, a Real-Time Task Manger (RTTM) has been presented in this paper. The Real-Time Task Manger is a hardware based extension to the processor that decreases performance bottleneck attributed to Real Time Operating System. Real-Time Task Manger decreases these performance bottlenecks introduced by Real Time Operating System by its hardware based architecture. The architecture of Real-Time Task Manger is being discussed in this paper provides an aid to deal with some common operations of Real Time Operating System that reason the performance degradation. For example event management, time management, and task scheduling. These operations have a property of some inborn parallelism. The Real-Time Task Manger uses this property to complete these operations in a constant time, and as a result, minimizes the overhead introduced by Real Time Operating System. The proposed architecture yields two benefits: It reduces the processor time taken by Real Time Operating System, and improves the response time to a considerable amount.

Keywords: *Real Time Operating System, performance, response time, processor utilization, task scheduling, time management, event management, Real Time Operating System overhead.*

1. Introduction

Real Time Operating Systems are essentially important for the development of real time applications. This importance can be sensed by watching the increasing demand of Real Time Operating System in the market. In [3], it has been shown that the shipment of half a billion of dollars will be sold in 2002 for the Real Time Operating System. Real Time Operating System provide benefits of better performance in task synchronization and hardware abstraction multitasking. However, these benefits by the Real Time Operating System are not free of cost. Real time applications suffer several performance issues introduced by Real Time Operating System.

Processor Utilization: It is a fraction of processing power that can be used by an application. Real Time Operating System degrades this processing power used by applications when the RTOS are used to provide services like pre-emption, and multitasking etc.

Response Time: It is the time taken by a real time application to respond to an outer stimulus. Response time depends upon many things such as: RTOS is pre-emptive or not; Outer stimulus is detected by polling or interrupts. There are very wide effects of Real Time Operating System on response time. Generally, the Real Time Operating System increase the response time to different amounts.

Several parameters regarding the performance are affected by Real Time Operating System that cause degradation in performance. It has been traced in [5] that much of the degradation in performance is cause by the effect on core operations. These core operations are time management, task scheduling, and event management. Statistical data on these core operations show that the execution frequency of these operations is relatively high, that these operations perform some mutually related functions, and that these operations depict parallelism in their execution behaviour. Exploitation of this parallelism cannot be made using software based implementations. However, exploitation of this parallelism can be made using hardware architectures that will result in minimizing the performance degradation. Extending the hardware architectures is not a cost issue these days due to falling costs of logics [4]. However, extension in hardware architectures can be made to improve the performance by giving special attention to frequently executing functions on processors. Facts discussed so far show that hardware based solutions will provide greater benefits to Real Time Operating System. These facts are a motivation towards the introduction of Real-Time Task Manger (RTTM). ***Real-Time Task Manger*** is a memory-mapped on chip peripheral hardware architecture. ***Real-Time Task Manger*** is designed to efficiently perform the core operations i.e. task scheduling, time management, and event management. R**eal**-T**ime** T**ask** M**anger** is designed such that it is compatible with variety of Real Time Operating System. The proposed approach reduces both the response time and processor utilization to a high amount.

2. Background

For better comprehension of frequently occurring bottlenecks in Real Time Operating System, some of the details are described here about their nature --- what those bottlenecks are, how they are implemented, and what is their frequency of occurrence.

Task Scheduling: it is a mechanism that decides which task should be permitted to run at a particular given time. Priority scheduling is the most commonly used technique in commercial Real Time Operating System [7]. Every task in the queue is allotted a priority number and the task having the highest priority number is chosen to run on the processor.

Time Management: it is a feature of RTOs under which different tasks are scheduled for a particular time. For example, blocking of a task for a particular time after which the task would go in ready state. Real Time Operating System receives their timing signals from a hardware timer. At every clock cycle from the hardware timer an interrupt is generated. The clock cycles from the hardware timer and clock cycles for the CPU are two different things, and therefore, they must not be mixed up. CPU frequency, usually, is in order of MHz, while the clock ticks from the hardware timer are in an order of kHz.

Event Management: in most of the Real Time Operating System, features are added that help the tasks to communicate and synchronize with each other. This interaction is referred to as *inter process communication (IPC)*. Semaphores and message queues etc are the examples of such communication and synchronization between the tasks. Event management is a process of keeping a record and monitoring the tasks for IPC i.e. tracking that which tasks are blocked due to unavailability of resources and that which tasks should be released due to the availability of the resources.

Table 1 show that these three operations of Real Time Operating System introduce some overhead. This overhead is proportional to the product of complexity and frequency. As seen from the table 1, there may be a linear increase in both the complexity and frequency with the increase in number of tasks; however, there may be a quadratic increase to the Real Time Operating System overhead introduced by both of the components. Moreover, hardware timer clock frequency also causes a linear increment in the overhead of time management and task scheduling.

Table 1. RTOS Operations

	Implemented using --- Overhead	Executed on
Event Management	▪ A separate block queue is associated with reach task. Rest is similar to task scheduling	▪ IPC is used by tasks. IPC by workloads decides it.
Task Scheduling	▪ Ready List (Unsorted) --- Overhead is linear for choosing a task ▪ Ready List (Sorted) --- Little and constant overhead for choosing a task. Linear overhead on adding a task to the ready queue ▪ Bit-Vectors --- Constant overhead for systems with having distinct and fixed priorities	▪ System calls. Expands with workload size. ▪ Timer interrupts in pre-emptive systems. Expands with clock frequency. ▪ When task ends in non-pre-emptive systems. Expands with amount of workload
Time Management	▪ A separate delay counter is associated with each task. Linear overhead for updating counters by timers ▪ Call out tables --- Counters delays are taken from most recent tasks. Changing overhead for updates. Linear overhead for adding tasks.	▪ Timer interrupts in pre-emptive systems. Expands linearly with clock frequency. ▪ When task ends in non-pre-emptive systems. Expands with amount of workload

In pre-emptive systems, the highest increment in response time is observed when the highest amount of time is taken by a critical section of code. However, same is not a case in non-pre-emptive systems. Here, Real Time Operating System operations do not add any significant amount to the response time. However, response time is increased by the execution of workloads. It is due to the reason that execution time of workloads is greater than Real Time Operating System operations. Therefore, longer amount of time is observed in workloads that disable the response of interrupts.

3. Related Work

Rag Nathan and Dick conducted a study on embedded systems to know the power consumption made by RTOs [2]. They used instruction-level simulator. The simulator was a Fujitsu SPARK lite processor which was using embedded software applications running on µC/OS. In their study, they proposed many new methods that can be used to design the software applications to reduce the power consumed by the Real Time Operating System.

Lindh and Adomat [6] suggested performing the real time functions in a separate hardware unit. For this they introduced a real time unit. They suggested two approaches; a software based approach and an exclusively hardware based approach. They showed that the real time unit has enhanced the performance. They also found that there is still limitation for the Real Time Operating System to utilize its the offerings.

Mooney studied multi processor systems. He introduced hardware based solutions for detecting deadlocks, synchronizing locks, and managing the dynamic memory. These solutions have effectively minimized the problems of atomic data access and resource conflicts. The solution presented in this paper is independent of these solutions as it suggests hardware based solution for Real Time Operating System functions. Moreover, Mooney proposed hardware based cyclic scheduler. Contrary to our design, this approach is application specific and it does not cover the mechanism of alternate scheduling like EDF etc.

4. The Real Time Task Manager (REAL-TIME TASK MANAGER)

The R**eal**-T**ime** T**ask** M**anger** is on-chip hardware peripheral. R**eal**-T**ime** T**ask** M**anger** is used to implement a task database. So, it assists the RTOS functions i.e. task scheduling, event management, time management. There is a memory-mapped interface between the R**eal**-T**ime** T**ask** M**anger** and the core.

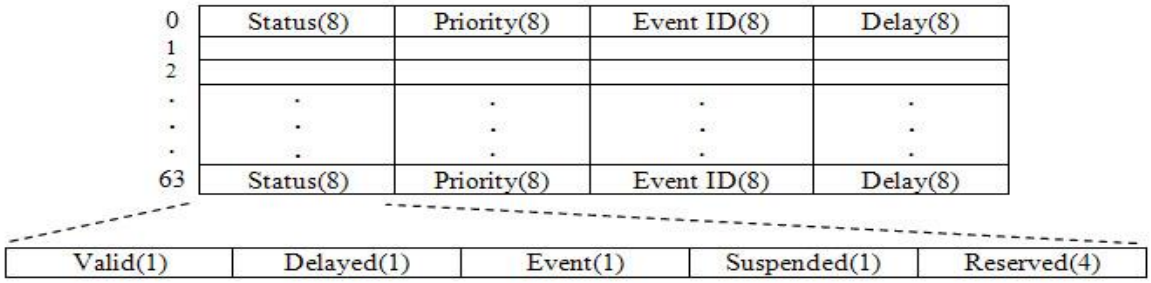

Figure 1. Data Structure of RTTM

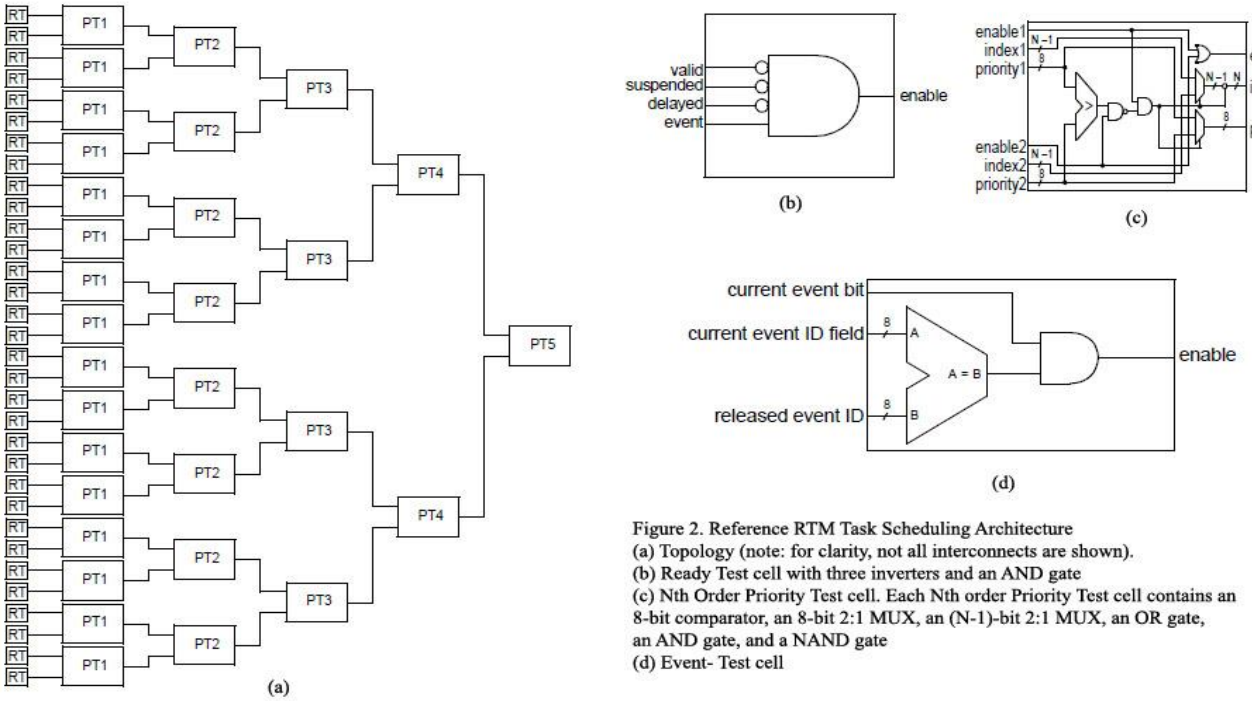

Figure 2. Reference RTM Task Scheduling Architecture
(a) Topology (note: for clarity, not all interconnects are shown).
(b) Ready Test cell with three inverters and an AND gate
(c) Nth Order Priority Test cell. Each Nth order Priority Test cell contains an 8-bit comparator, an 8-bit 2:1 MUX, an (N-1)-bit 2:1 MUX, an OR gate, an AND gate, and a NAND gate
(d) Event- Test cell

Figure 2

R**eal**-T**ime** T**ask** M**anger** (RTTM) uses this interface to communicate with the core. Architecture of R**eal**-T**ime** T**ask** M**anger** is shown in Figure 1. Details of a single task are shown in each row of the RTTM. Each row of the RTTM can be accessed using global address space. Each row can be edited or selected like an ordinary array of structures. Each row is divided into four parts. Each part is individually addressable. Each row has a *status* part (composed of some bits) that shows the status of the relating row. There is a *valid* bit that shows whether or not the corresponding row is used by a task. The *delayed* bit shows whether or not a task is in waiting state. If the bit is set then the task will be waiting for number of clock ticks mentioned in *delayed part* of the row, after that, the task

will be ready to run. The *event* bit shows whether or not a task is pending on an event. If the bit is set then the task is pending on an event whose id has been mentioned in the *event id* part. The *suspended* bit shows whether or not the task is suspended. When the ready bit is set and all other three bits (delayed, even, and suspended) are cleared, this setting shows that the task is ready to run. *Priority* part of the row is used to assign a priority number to the corresponding task. The number of rows in R**eal**-T**ime** T**ask** M**anger** architecture could by any from 64 to 256. The real time operating system, using registers, can get information from RTTM. Instruction can also be issued to R**eal**-T**ime** T**ask** M**anger** via these registers. The R**eal**-T**ime** T**ask** M**anger** schedules fixed priorities. Task with the greatest priority can be retrieved from the RTTM. Mechanism of time management by RTTM is as following:

- The number of clock ticks for which a task is to be delayed is specified by RTOS and is stored in delay part
- The number is decremented in all the rows by each clock tick
- When the number reduces to 0, the delayed bit updated accordingly

Mechanism for even management by REAL R**eal**-T**ime** T**ask** M**anger** is the same as fixed-priority scheduling. When an even occurs, Real-Time Task Manger selects a task with the greatest priority that is pending on that even. Note that at this time, Real-Time Task Manger does not select a ready task with the greatest priority.

5. Architecture

Let us see that how complex it is if we want to implement the hardware architecture of Real-Time Task Manger. The architecture shown in Figure 1 has 64 rows. Each row has 40 bits. So, to implement the architecture, 2560 flip flops are needed. This number is small and therefore can be implemented easily. All the common limits parameters are present in the architecture, and therefore, the architecture is enough for most of the real time applications. The mechanism of task scheduling is accomplished with the help of 64 Ready Test (RT) cells and 63 Priority Test (PT) cells arranged in a binary tree. Due to the limited space, the mechanism has been shown for 32 entries in figure 2. The only work of Ready Test cells is to find which tasks are ready to run. Each Priority Test cell has two inputs; ready bit and priority number of the task. The out of Priority Test cell is a task which is ready and has the highest priority number out of the two input tasks. The ready task with highest priority among all 64 tasks is outputted at the root of tree. The structure of each Nth level Priority Test cell is constructed with a 8-bit comparator, one AND, one OR, one NAND, and two 2:1 Multiplexers. The growth of RT or PT cells for number of tasks is linear. RTM constructed for *N* tasks, it will have *N* number of RT cells and *N-1* PT cells. The RTTM takes logarithmic time to compute a task with highest priority. For *N* tasks the computational time delay is ($\log_2 N$). This computational delay is enough fast for the maximum number of tasks commonly used in Real Time Operating System.

Let us discuss the time management mechanism of Real-Time Task Manger. Time management mechanism is simple. Only 64 Delay Decrement (DD) cells are needed as shown in Figure 3. Each of the DD cell is composed of a 16-bit adder and an *AND* gate. Delay fields are decremented by DD cells on each clock tick. Eventually, the delayed bit is cleared when the delayed field reaches to zero. The growth of DD cell with respect to the growth of number of tasks is also linear. However, the computational delay is constant for this mechanism.

Event management mechanism is much similar to task scheduling mechanism of Real-Time Task Manger as shown in figure 2. However, there is one difference here i.e. Event Test cells are used rather than the Priority Test Cells. Due to the similarity, it is possible to use the tree of Priority Test Cells to be used for task scheduling and event management. The event test cells are designed to check whether the event bit is set and whether the

event ID is the same as the ID of released resource. All the test cells are made up of an AND gate and a comparator of 8-bits. The logics used for event management grows linearly. As it grows in a shape of binary tree so the computation delay is same for event management and task scheduling i.e. O (log (n)).

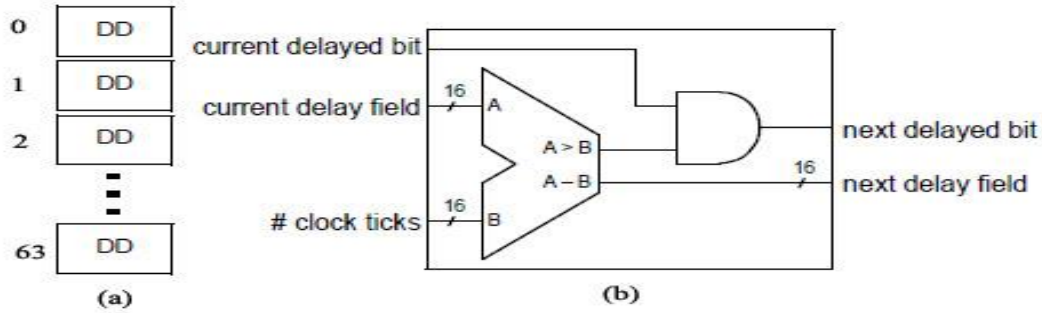

Figure 3. Reference RTM Event and Time Management Architecture
(a) Topology (note: for clarity, not all interconnects and shown)
(b) Delay Decrement Cell

To know whether the hardware based structure of Real-Time Task Manger is feasible or not, the die area is important to be determined. Using the current techniques for implementing the caches register files [6], RTTM needs round about 2600 bits. This is a feasible number to be implemented for RTTM. This number is the same as used by a register file of 32-bits by 64-words which has been easy to be implemented. Although this number is relatively greater than a simple core but it is much smaller than a complete chip. It is due to the reason because a conventional processor is comprised of a core and some other on-chip components i.e. memory and other I/O components.

6. Future Work

For determining the success of presented model of Real-Time Task Manger in practical real-time systems, exact improvements will be measured by making experiments on real-time systems that are using real-time operating systems. All types of measurements will be made on a simulator (C-based and in-house). The chosen processor will be a 32-bit processor. It would be a high-performance VLIW processor. The selected simulator would be exactly similar to a real processor and it would be able to execute the same binary files that are executed on real systems. For these experiments and analysis two different types of real time operating systems will be used: first one would be µC/OS, which is a famous pre-emptive RTOS used commercially [2], the second one would be NOS, which is an in-house non-pre-emptive RTOS, and it also has the representing ability of most of the Real Time Operating System [2]. Both the simulators (µC/OS and NOS) will be used to measure RTOS Overhead and Response Time by using Real-Time Task Manger model and without using the Real-Time Task Manger model.

7. Conclusions

Hardware based structure for RTTM (Real-Time Task Manger) has been reviewed and presented in this paper. RTTM structure implements some real time operating system operations. These operations are Task Management, Event Management, and Task Scheduling. These RTOS operations have inherent properties of parallelism. Taking the benefit of parallelism, these RTOS operations can be performed in small time and thus the transparency can be decreased to a great amount. RTTM decreases the processor time used by the RTOS and the maximum response time with magnitude order time.

References

[1] J. L. Pruitt, W. W. Case, *"Architecture of Real Time Operating Systems",* NASA Contract No. NAS8-31222, the Marshall Space Flight Centre, 2003.
[2] Sandro Penolazzi, Ingo Sander and Ahmed Heman, "*Predicting Energy and Performance Overhead of Real-Time Operating Systems"* Dept. of Electronic Systems, School of ICT, KTH, Stockholm, Sweden.
[3] Mohamed Salan, Vincent J. Mooney, *"Hardware Support for Real-Time Embedded Microprocessors System on Chip Memory Management",* ACM Conference, August 2003.
[4] Lucas Kreger-Stickles and Mark Oskin, "*Microcoded Architectures for Ion-Trap Quantum Computers",* IEEE conference, September 2006.
[5] Dick, Lakshminarayana, "*Power Consumptions in Embedded Operating Systems*." 37th Design Automation Conference, Los Angeles, 2000.
[6] Mooney and Blough, "*A Framework for Hardware-Software RTOS for SOCs*," IEEE Design and Test of Computers, December 2002.
[7] Coopee. "*Embedded Intelligence*." InfoWorld, November, 2000.
[8] Akgul and Mooney, "*The System-on-a-Chip Lock Cache*," September 2002.
[9] Mooney and Micheli, *"Hardware/Software Codesign of Run-Time Schedulers for RTS"*, Design Automation of Embedded Systems, September 2000.
[10] Andrew Nere and Mikko Lipasti, *"Cortical Architectures on a GPGPU",* ACM Conference, March 14, 2000.
[11] Richard J. ENBODY, Kelley PELLINI, and William MOORE, "*Performance Monitoring in Advanced Computer Architecture"*, ACM Conference, June 2004.
[12] Philip C. Treleaven, "*The New Generation Of Computer Architecture*", Computing Laboratory, University of Newcastle upon Tyne, Newcastle upon Tyne, England
[13] Eric Chi, Stephen A. Lyon, Margaret Martonosi, "Tailoring Quantum Architectures to Implementation Style: A Quantum Computer for Mobile and Persistent Qubits", Dept. of Electrical Engineering, Princeton University

Author

Javed Ahmad Shaheen, MSCS, Virtual University of Pakistan