PI-RTOS: Platform Independent RTOS

Julia Andrews
Mtech Scholar
Government Engineering College
Idukki

Philumon Joseph
Assistant professor
Government Engineering College
Idkki

ABSTRACT
This paper proposes a Real-Time Operating System kernel for the 32-bit Leon3 processor. A system is said to be Real Time if it is required to complete its work and deliver its services on time. In a real-time system the correctness of its output, is an important factor, depends not only the logical computations carried out but also the time at which the results were delivered to the external interface. A Real time operating system (RTOS) is a class of operating system intended for real time applications. The requirements for developing an RTOS include RMS scheduling algorithm, file management scheme, interrupt handling, Timer etc. Most of these functions are in POSIX 1003.1b compliant. tsim simulator is used for compilation and debugging..

General Terms
Interrupt handling, File management, Timer.

Keywords
PI-RTOS, Leon3, RTOS, Rate monotonic scheduling.

1. INTRODUCTION
The primary role of an operating system's (OS) is to manage its resources to meet the demands of target applications. The target application environment of Traditional Timesharing operating systems demands fairness and high resource utilization whereas Real-time applications demand timeliness and predictability. A late or a missed response from real-time systems shall leads to the failure of the total system. For hard real-time system, a failure would end up in catastrophic consequences up to and including loss of human life. Aircraft collision avoidance, anti-lock brake, pacemaker and anti-missile systems are some hard real-time system. In the case of soft real-time applications like communication switching systems and streaming audio or video, predictable response is required but occasional failures can be tolerated.

The ability of the operating system to provide a required level of service in a bounded response time (POSIX Standard 1003.1) is the real-time in operating systems [1]. The RTOS is designed in such a way that it maintains a balance between a reasonably rich feature set for application development and deployment, thereby not sacrificing predictability and timeliness. Too late or too early correct output could be useless, or dangerous.

An advanced algorithm is used by an RTOS for scheduling and the flexibility of scheduler is what enables a wider and computer-system orchestration of process priorities; but a real-time OS is more frequently dedicated to a small set of applications. A real-time OS is concerned, with minimal interrupt and thread switching latency; and is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

The main differences of an RTOS compared to an ordinary Operating System (OS) are that they have a predictive time management and they are deterministic. An ordinary OS often tries to perform all actions with average throughput in mind; this in turn reduces the average case at the cost of the worst-case. An RTOS must try to manage all system calls and task switches in an analysable, predictive manner by providing a known worst-case behaviour oftenly. There are different RTOS present with different features like LynxOS [2], VRTX [3], VxWorks [4], Win-CE [5] etc. LynxOS [2] is one of the RTOS, which uses non-pre-emptive, FIFO, and Round Robin Scheduling algorithms. LynxOS only checks the correctness of the output without bothering about timeliness. But in PI-RTOS the scheduling is done based on the periodicity, and importance is given for time taken to complete the execution and correctness of the output.

The kernel of an OS consists of core components for computing, including CPU scheduling as well as process management. It consists of mainly the Resource Management subsystem and the Process Management subsystem. The resource management subsystem is a set of supporting functions for various system resources and user interfaces and the process management subsystem is a set of transition manipulation mechanisms for processes and threads interacting with the system kernel and resources. The main features of an RTOS are its scheduling algorithm (RMS) [6] which work based on periodicity. Interrupt handling which includes context switch operations. File management that includes different file related operations like file close, file open, fstat, stat etc.

The remainder of the paper is organised as follows. Section II explains the Architecture of the PI-RTOS. Section III contains the Features of the proposed RTOS and in Section IV Performance evaluation of the proposed RTOS is given.

2. ARCHITECTURE
PI-RTOS is designed for handling different tasks, time, file operations and interrupts as a platform independent structure real-time system. But, the context switching carried out in case of an RTOS is platform dependent. The conceptual architecture of PI-RTOS is as shown in Figure 1, where interactions between system resources, components, and internal control models are illustrated. The architecture of PI-RTOS is mainly divided into 2: an Application layer and a System layer. The system layer in turn consists of the Hardware Dependent part and the Hardware Independent part. Hardware Dependent part is the one which depends on the processor which is used to run that particular code (application). In PI-RTOS, LEON3 [7] processor is used and it is a 32-bit processor based on the SPARC V8 [8] architecture. Hardware Independent part in comparison to its counterpart does not change the code even if the processor that is used to execute them is changed.

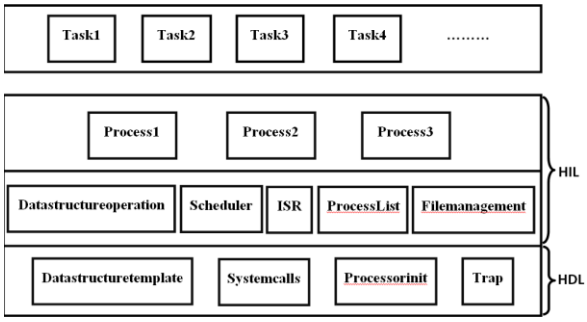

Fig 1: Architecture of PI-RTOS

In this RTOS, different tasks are maintained as a process along with their periodic nature. Tasks that are having the same periodicity are grouped as a single process and execution of the tasks in a process is based on their priority value.

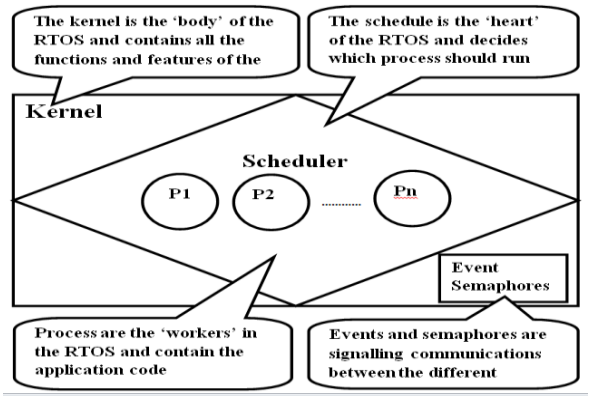

Fig 2: RTOS Kernel Core

3. PI-RTOS FEATURES

The RTOS have the ability to schedule tasks and meet deadlines, error recovery and low task switching latency, small footprint and overheads. It is the kernel, which is the core of an OS that provides task scheduling and task dispatching. The main features of PI-RTOS are given below:

3.1 Scheduling

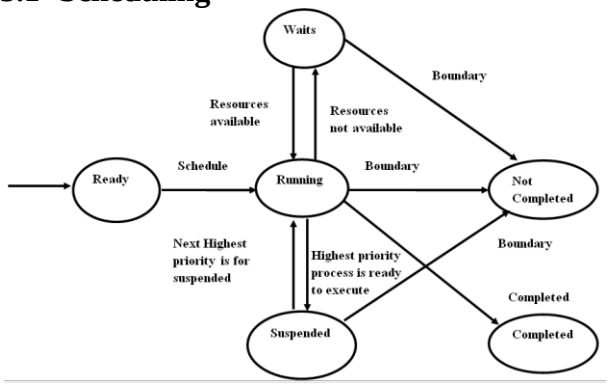

Fig 3: State transition diagram of the process scheduler

Most commonly used RTOS scheduling algorithms include Cooperative scheduling, Preemptive scheduling, Rate-monotonic scheduling, Round-robin scheduling, Fixed priority pre-emptive scheduling, an implementation of preemptive time slicing, Fixed-Priority Scheduling with Deferred Preemption, Fixed-Priority Non-preemptive Scheduling, Critical section preemptive scheduling, Static time scheduling, Earliest Deadline First approach, Stochastic digraphs with multi-threaded graph traversal etc.

The PI-RTOS proposed in this paper makes use of the RMS scheduling algorithm. This is due to the periodicity in task scheduling in RMS scheduling algorithm. The **Rate-Monotonic Scheduling** (**RMS**) [6] is a scheduling algorithm which is used to schedule different tasks in a periodic nature. RMS is mainly used in real-time OS with a static-priority scheduling class. The static priorities of the job are assigned on the basis of the cycle duration. ie, the cycle duration is shorter then job's priority is higher. These operating systems have deterministic guarantees with regard to response times and are generally pre-emptive. Rate monotonic analysis is used in conjunction with those systems to provide scheduling guarantees for a particular application. State transition diagram of the process scheduler is given in figure 3.

In case of rate-monotonic scheduling algorithm, optimality means the imposition of constraints upon the process system. These include:

- Processes of fixed numbers;
- Periodic nature for all processes;
- Based on their period, all processes have deadline;
- Before running subsequent instances, current instance of a process must be complete;
- Their own known worst-case execution times for all processes;
- No synchronisation is permitted between processes;
- Initial release of all processes at time 0.

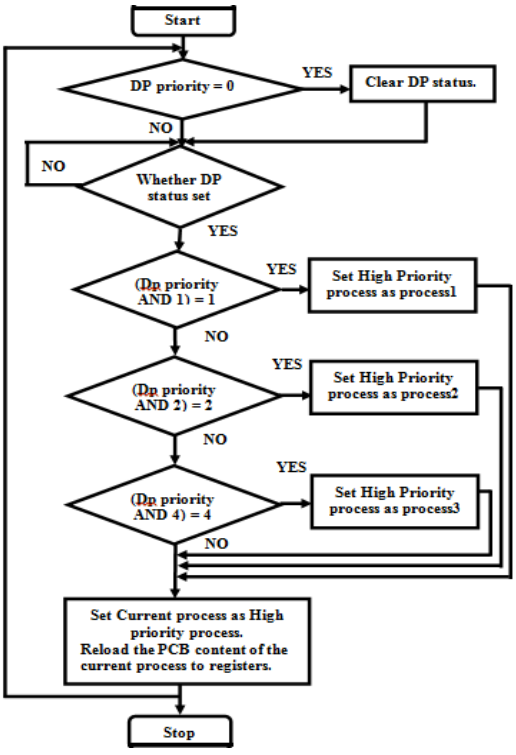

Fig 4: Flow chart of scheduler

3.2 Interrupt Handling

The highest priority task is blocked by an interrupt handler from running and the interrupt handlers are kept as short as possible so that the RTOS can keep thread latency to a minimum. If possible, the interrupt handler defers all interactions with the hardware. Typically all that is necessary is to acknowledge or disable the interrupt, so that when the interrupt handler returns this won't happen again, and notify a task that work needs to be done. For this, a driver task needs to be unblocked by releasing a semaphore thereby setting a flag. A scheduler often provides the ability to unblock a task from interrupt handler context.

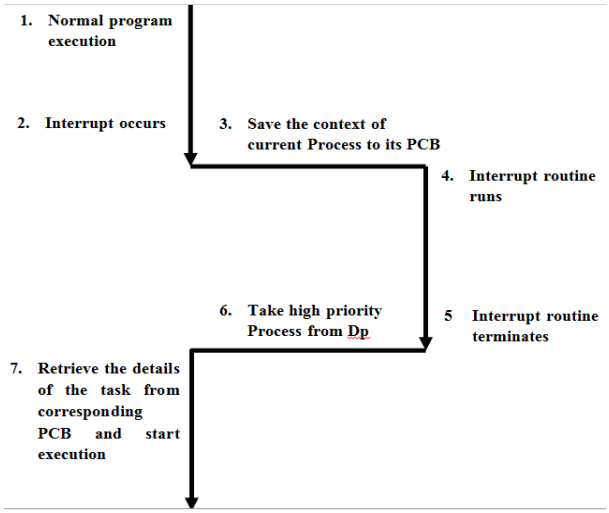

Fig 5: Interrupt Handling in normal execution of a program

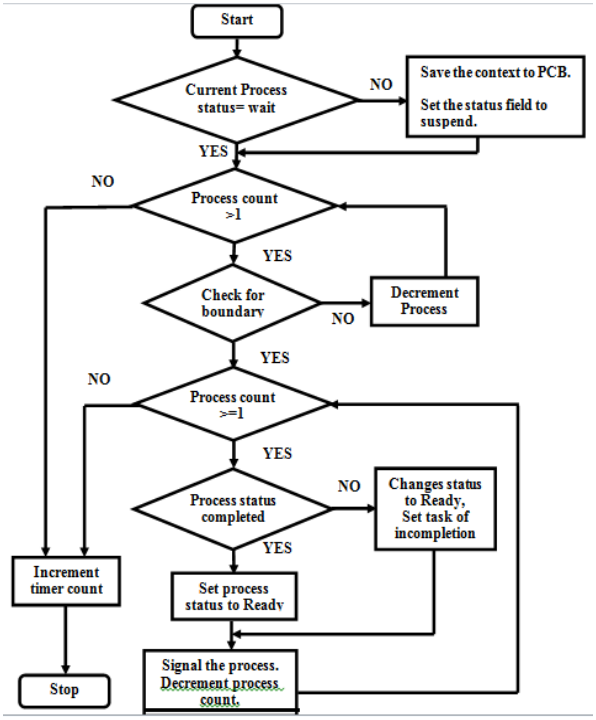

Fig 6: Flow chart of interrupt handling

3.3 File Management

The data that works with on computers is kept in a hierarchical file system. File system directories have files and subdirectories beneath them. Although with the use of the computer operating system the user can keep image data organized; how the user can name files and folders, arrange these nested folders and handle the files in these folders are the fundamental aspects of file management. The organization of data by the operating system can be enhanced by the use of cataloging programs. These programs make organizing and finding image files easier than simply relying on the computer's directory structure. The other feature of catalog programs is that they can streamline backup procedures for better file protection.

3.4 Timer

A **timer** [8] is a specialized type of clock for measuring time intervals. A stopwatch is a timer that counts upwards from zero for measuring elapsed time. On the other hand, a device which counts down from a specified time interval is usually called a *timer*.

4. PERFORMANCE EVALUATION

The PI-RTOS proposed in this paper as well as most of the paths are tested and verified using tsim simulator.

4.1 Test Cases

Some of the test cases identified for testing the scheduling and working of interrupts are listed below:

Case 1: Process 1, 2, 3 complete in first period.

In Case 1 all processes complete within the first Interrupt, because the execution time of all the process will be below the first periodicity.

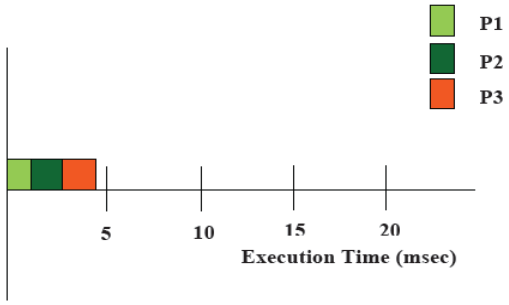

Fig 7: Case 1 graph

Case 2: Process 1 completes and Process 2 is suspended at 1st interrupt and all processes complete before 2nd interrupt.

In Case 2, first process completes within the first Interrupt, second process is suspended and third process is therefore not called. The first interrupt occurs in first period and the second interrupt occurs in twice the time taken by the first period and third interrupt occur in thrice the time taken by that of the first period. In this case, execution time of first process is less than first period. So before the first interrupt, second process is started but not completed. At the first interrupt, a second process is suspended; then the first process is reloaded and its execution completed.

Case 3: Process 1, 2 completes and 3rdprocess is suspended at the 1st interrupt. 3rdprocess completes before 2nd interrupt.

In Case 3, first process and second process are completed within the first Interrupt and third process is suspended. Total execution time of the first and second process is below that of the first interrupt.

Case 4: Process 1 completes and process 2 completes exactly at the 1st interrupt.

In Case 4, first process and second process are completed within the first Interrupt . The second process completes exactly at the time of first interrupt. Total execution time of the first and second process is exact first interrupt.

Case 5: Process 3 gets suspended 3 times while 1 and 2 gets completed in the first period itself.

In Case 5, first process and second process are completed within the first Interrupt and the third process is suspended in the next 3 interrupts.

Case 6: Process 1 completes, 2 and 3 gets suspended.

In Case 6, first process is completed within the first Interrupt but second process and third are suspended for the next 3 interrupts.

Case 7: Processes does not finish executing before specified periods.

In case 7, the execution time of the process increases because of some reasons, so the processes are not completed within the time period.

4.2 Latency

Latency is a time interval between the stimulation and response. From a more general point of view, it is the time delay between the effect and the cause of some physical change in the system being observed. **Interrupt Latency** is the time that elapses from when an interrupt is generated to when the source of the interrupt is serviced. Interrupt latency, also called as the Interrupt Response Time, is the length of time that it takes for a computer interrupt to be acted on, after it has been generated.

Table 1. Interrupt Latency

Interrupt Latency	
LynxOS	13
VRTX	4
VxWorks	3
Win-CE	9.5
PI-RTOS	4.5

In computing, a **context switch** is the process of storing and restoring the state (context) of a process or thread so that execution can be resumed from the same point at a later time. Context Switch Latency is the time taken to perform context switch. Context switch latency obtained for the proposed PI-RTOS is 5.3 microsec.

5. CONCLUSIONS

In this paper the platform independency of PI-RTOS in Leon3 is presented. It mainly focuses on designing an RTOS with high predictability and timeliness. This paper provides a detailed overview for developing an embedded system using Leon3 and provides the details about the features of PI-RTOS like (inbuilt features) scheduling, interrupt handling, file management and semaphore.

Tsim simulator is used for testing and compilation. Using simulator testing is completed successfully.

6. ACKNOWLEDGEMENT

We would like to thank everyone in the Department of Computer Science and Engineering, Govt Engineering College, Idukki. Also would like to thank the research team at VSSC, ISRO Trivandrum. Who helped in understanding the RTOS systems concepts, and other concepts related to it .The author would like to thanks the anonymous referees for their valuable comments, which greatly improved the readability of the paper.

7. REFERENCES

[1] "Standard for Information Technology—Portable Operating System Interface (POSIX®)," IEEE Std 1003.1-2004, The Open Group Technical Standard Base Specifications.

[2] "LynxOS user's guide", LynxOS Release 4.0.

[3] "VRTX: A Real-Time Operating System for Embedded Microprocessor Applications**",** Ready, J.F., Micro, IEEE (Volume: 6, Issue: 4).

[4] " VxWorks RTOS", Kosuru Sai Malleswar.

[5] Win-CE:"Assessing the real-time properties of Windows CE 3.0", Netter C.M, Baceller, L.F. Object-Oriented Real-Time Distributed Computing,2001. ISORC-2001. Proceedings 4th IEEE International Symposium on.

[6] "Rate Monotonic Analysis for Real-Time Systems",Lui Sha, Mark H. Klein, John B. Goodenough; The Springer International Series in Engineering and Computer Science Volume 141, 1991, pp 129-155.

[7] Leon3: M. Danˇek et al., UTLEON3: Exploring Fine-Grain Multi-Threading in FPGAs,DOI 10.1007/978-1-4614-2410-9 2, © Springer Science+Business Media, LLC 2013.

[8] "Standard Products UT699 LEON3FT/SPARCV8 MicroProcessor", Functional Manual February 2014, www.aeroflex.com/LEON.