National Conference on Science, Engineering and Technology (NCSET – 2016)
Volume: 4 Issue: 6
ISSN: 2321-8169
63 - 64

Improving the Interrupt latency in real time Operating system on STM32F04 Discovery kit

Nikhil m.n[1], Nikhil Mattani[2],Prof.Prakash.V[3],
School of Electronics Engineering,
VIT University, Chennai campus,
Chennai, India.
E-mail: mnnikhil92@gmail.com, nikhil.mattani2015@vit.ac.in,prakash.v@vit.ac.in.

Abstract: Time critical and complex embedded system requires real time operating system [RTOS] for their effective operation. Every RTOS is governed by various performance parameters like Schedulingalgorithm,interruptlatency,contextswitching,memory management etc. The objective of this work is to improve the performance of RTOS by reducing interrupt latency period. An algorithm will be proposed to reduce the interrupt latency in an open source RTX RTOS and it's performance will be analyzed in real time using STM32F4 Discovery kit and benchmarking tools.

*****

I. INTRODUCTION:

Nowadays embedded systems are being used in each and every field mainly to provide security and comfort to human beings. In this process embedded systems have become so complex and time critical that there is a bigger need of time guarantee in their real time task.Hence Real time operating system are used to provide suffort for scheduling,memory management,inter process communication and interrupt and event handling.

Interrupts play a very important role in embedded system, as the events are basically generated by interrupts. The characteristic feature of this interrupt is they can concurrently access the kernel data, hence there is a need to maintain the integrity of kernel data during interrupt processing.The protection to the kernel data can be provided by disabling the interrupts during the critical section execution.

Even though disabling the interrupts will protect the kernel data, because of which interrupt latency will be introduced. Interrupt latency is the time period between the arrival of interrupt at the processor to the start of the interrupt service routine[ISR].

Ideally the interrupt latency should be zero.But practically it depends on the hardware and the operating system used,and it ranges from micro seconds to nano seconds.In this paper a new method is proposed to reduce the interrupt latency in Real time operating system and the proposed algorithm is verified on STM32F4 discovery kit.

II. LITERATURE SURVEY

The main factor which influences the interrupt latency is the length of the critical section. General approach to protect the integrity of kernel data is by disabling interrupts. This can be done by setting the processor's interrupt mask bit before the critical section and clearing it after the critical section. However this approach makes the system to turn into inconsistent interrupt state.

There are various techniques proposed to reduce kernel-induced interrupt latency. In[6] an additional kernel is placed between the hardware and the original kernel, hence there is no need to use interrupt disable instruction. In [5] the use of interrupt disable instructions is eliminated in most of the critical section by using mutex for synchronizing. Although this approaches helps to reduce interrupt latency they require complete modification of the kernel code.

In [7] a relationship is established between the critical section and linux interrupt handler.A fast interrupt response time is achieved by constructing a lock queue in critical section.In [4] this technique the interrupt handlers are modified to act as a kernel thread and a priority order is established between the kernel threads and interrupts.But the major drawback of this techniques is the interrupt latency is reduced based on the conventional method of arranging the real time task,and does not consider any special requirements that arise during the execution of the kernel thread.

III. Proposed Method

In this paper a method is proposed to reduce the interrupt latency and also for executing emergency job by low priority task at a specific point in time. In this technique the execution of interrupt service routine is postponed to the end of the critical section because of which there is no need to disable the timer interrupt and it can enter the kernel at any point of time. The important factors which are taken into consideration in this paper are as follows

i) Advance programmable timer interrupt is used to trigger the timer interrupt.
ii) Emergency jobs are executed directly in timer interrupt service routine.
iii) A new emergency critical section is defined where the timer interrupt and kernel threads can access the resource exclusively.
iv) Only during emergency critical section interrupts are disabled
v) In general interrupt prohibited section all the interrupts other than the timer interrupt are executed after the critical section.

IV. Delayed Interrupt Mechanism

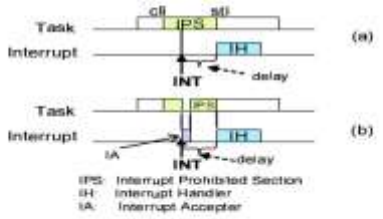

Fig 1: Basic concept of interrupt execution

In the original kernel whenever a task enters an interrupt prohibited section it disables all the interrupt by using an instruction called {cli},and after the execution of the critical section it enables the interrupt by using an instruction called {sti} so that they can enter the kernel again.
But in this technique the interrupts are not disabled during the critical section, and they can enter the kernel at any point of time. However in order to protect the integrity of the kernel data whenever the interrupt occurs the kernel just acknowledges and registers the interrupt and the control quickly returns to the main task execution, as depicted in the figure 1.
Even though the normal execution {fig 1.a} and the execution of the kernel thread in our technique looks similar, there is a major difference in the execution of timer interrupt which is as shown in the figure 2

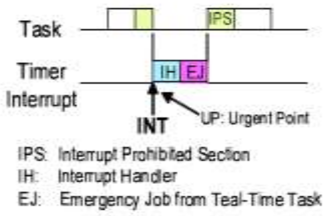
Task
IPS
Timer
Interrupt
IH
EJ
INT
UP: Urgent Point
IPS: Interrupt Prohibited Section
IH: Interrupt Handler
EJ: Emergency Job from Teal-Time Task

Figure 2:Execution of timer interrupt

Timer interrupt service routine includes two sections as shown in figure one is the actual timer interrupt service routine and the second one is the execution of emergency jobs as requested by low priority real time task.

V. Implementation

The proposed system is implemented on RTX rtos and STM32F4 discovery kit.
The interrupt latency of the present system is measured as shown in below figure

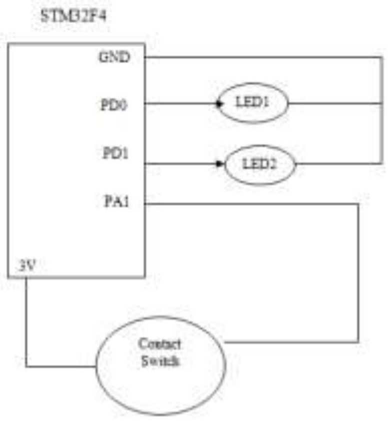

Fig 3:Interrupt latency measurement

PD0 and PD1 are the digital out lines to blink an LED,PA1 is an external line 1 interrupt in stm32.PA1 has a position of 6 and a priority of 13 and it is a “settable” priority.The address of this external interrupt is 0x0000 0058.
STM32F4 is programmed to continuously blink the LED1 and whenever an switch is pressed the external interrupt on line 2 gets activated and led 2 is made to blink 5 times in interrupt service routine.Here the interrupt latency can be measured using system and thread viewer option which is available in atollic truestudio development tool or by using digital oscilloscope. The worst case interrupt latency will be around 0.5 micro seconds.

VI. RESULTS

In this paper a technique is proposed to reduce the interrupt latency based on the delay mechanism.In this interrupts from external devices are not disabled and they are simply delayed in critical section.Also a system call can be provided to make a low priority task execute emergency job at a specific point in time preempting higher priority task.
A prototype system is implemented in RTX rtos on STM32F4 board.If the delayed interrupt mechanism is used the interrupt latency can be reduced to 2.2% original kernel latency and the time required for executing emergency job will be 27.9 microseconds.

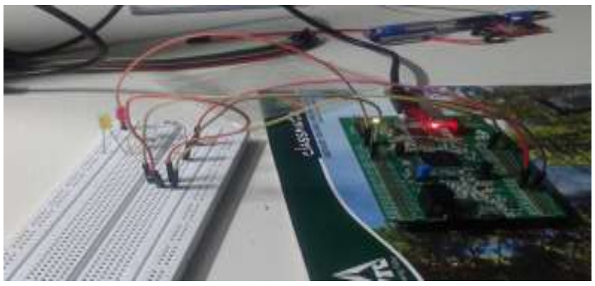

Fig 4:Experimental setup

References

[1] Maobing Dai, Pao “PIL: A Method to Improve Interrupt Latency in Real-Time Kernels”, Eighth IEEE International Conference on Embedded Computing; IEEE 2010
[2] Haloshen Dai and Yutaka Ishikawa,” Delayed Interrupt Processing Technique for Reducing Latency of Timer Interrupt in Embedded Linux” 2009 International Conference on Computational Science.
[3] IngoMolnar[online],"http://people.redhat.com/mingo/realtime-preempt/"
[4] M. Barabanov and V. Yodaiken, “Introducing Real-Time Linux.” Linux Journal, February 2007.
[5] Huaidong Shi, Ming Cai, Jinxiang Dong, “Interrupt Synchronization Lock for Real-time Operating Systems.” Sixth IEEE International Conference on Computer and InformationTechnology (CIT), 2006
[6] Luis E. Leyva-del-Foyo, Pedro Mejia-Alvarez, Dionisio de Niz, “Predictable Interrupt Management for Real Time Kernels over conventional PC Hardware,” 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS),2006.
[7] B. Srinivasan, S. Pather, R. Hill, F. Ansari, and D. Niehaus, “A Firm Real-Time System Implementation Using Commercial Offthe-Shelf Hardware and Free Software,” Proc. Fourth IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS),2008