Journal of Theoretical and Applied Information Technology
10th February 2013. Vol. 48 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HARDWARE IMPLEMENTATION OF TASK MANAGEMENT IN EMBEDDED REAL-TIME OPERATING SYSTEMS

[1]SHI-HAI ZHU
1Department of Computer and Information Engineering, Zhejiang Water Conservancy and Hydropower College
Hangzhou, Zhejiang Province 310018, China
E-mail: yyzz98@163.com

ABSTRACT

Embedded Real-Time Operating System is widely used in more and more application fields, such as aerospace, missile guidance, automobile electronics and construction of nuclear power plants. In order to meet the needs of market, it is necessary to improve fundamentally the performance of embedded real-time operating system. Logically speaking, hardware is equivalent to software. Any instruction can be executed by either hardware or software. Along with the development of semiconductor technology and the trend of hardening software, the boundaries between software and hardware have become more and more blurred; all these have provided more convenience to implement embedded real-time operating system by hardware. As is known to all, task management is one of the most basic functions of operating systems, therefore we take real-time operating system μC/OS-II as an example to perform hardware design of task management in this paper, and implement the hardware circuits to create, delete tasks and the logic circuit of ready list. The simulation results show that the hardware realization of task management holds the correctness of system calls besides reduces their execution time and the overhead of processors.

Keywords: *ERTOS, Task Management, FPGA, Hardening Software, TCB.*

1. INTRODUCTION

Embedded Real-Time Operating System (ERTOS) is widely used in the strong real-time systems, such as a new generation of fighter planes, aerospace systems; The traditional embedded real-time operating system compiles its kernel with application programs together, and runs onto the same microprocessor, but the kernel task generally has the higher priority than application programs, the latter can't effectively make use of microprocessors and storage space, which reduces the execution efficiency of application programs[1]. As a result of limitation of memory space and processing ability, it is difficult for the low-end embedded operating system to effectively support tasks similar to having high complexity and large number of concurrency. What we really need for real-time operating system is that it takes up less processor time and storage space, at the same time with higher efficiency and stability, lower power consumption.

Currently the main method to improve the performance of embedded operating system is to rely on using high frequency, long-digits microprocessors or improved software algorithms. There are a lot of researchers at home and abroad to study these algorithms, for example, task scheduling, interrupt control, and system security control. Their purpose is to improve the processing power and security of embedded operating systems; but these improvements often lead to other problems, such as high energy consumption, etc.; and because programs are executed by the processor in sequential order, the improving strength is always limited; In addition, no matter how perfect a software system is written, it always has loopholes. As is well known that the parallel processing ability of hardware circuits is enormous, many hackers think it is nearly impossible to crack hardware, therefore its security is beyond question. But if an embedded operating system is implemented entirely by hardware, then it will lack of appropriate flexibility and reusability, and with high expense.

Now a better solution is to adopt mixed operating system implemented by both software and hardware. A part of functions of operating system is realized by hardware, and the part not suitable for the realization of hardware is still implemented by software. With the continuous development of semiconductor technology, the cost of hardware becomes more and more low; hardware method is widely used to achieve those functions of original software solution. Generally speaking, task management is one of the most basic functions of operating systems. This paper takes real-time operating system μC/OS-II as an example to perform hardware design of task management, implements the hardware circuits to create and delete tasks, and the logic circuit of ready list.

2. LITERATURE REVIEW

Jaehwan Lee et al. put forward the concept of hardware real-time operating system (HRTOS) from the 1980s, and proposed to use specific hardware IP-kernel to realize RTOS scheduler after analysis and comparison of RTOS scheduler based on hardware and software implementation [2]. Task scheduling is the key of RTOS, communications among the tasks, external events processing and interrupt handling are all dependent on task scheduling. And with the improvement and enhancement of system functions, the relationship among tasks becomes more complex, and more peripherals need to be dealt with, all these require task scheduling continuously to participate in, causing the system performance and real-time response ability to decline sharply [3]. If task scheduling is implemented partly by hardware, it is no doubt that its performance can be improved greatly, thus the performance of the whole RTOS is improved accordingly [4].

Takumi Nakano[5] developed a kind of silicon chip called STRON-I (Silicon OS) in the mid 1990s, and put forward the concept of silicon OS, using VLSI technology to implement operating system (TRON) by a chip hardware, so that the operating system can work harmoniously with microprocessor chip in parallel way, further ensure high reliability of real-time operating system. Silicon OS divided traditional real-time operating system kernel uITRON into three parts, that is, Micro Kernel, Interface (Software Kernel) and Silicon TRON. Hard kernel part adopted HDL language to implement task management, interrupt management, information exchange and communication, which were usually implemented by software in usual real-time operating system.

Professor Peter Waldeck at Queensland University in Australia published a paper about hardware and software partitioning at the beginning of this century, in which he put forward the mature conversion among hardware and software modules, and the mutual communication method among those modules [6].

Moonvin Song, Sang Hong, Yunmo Chung at kyung Hee University in South Korea combined configurable CPU with RTOS by using FPGA, and obtained an efficient RTOS; in this design, in order to reduce the power consumption of RTOS, they implemented the context switch operations by hardware among the most time-consuming task switching process and also interrupt handling [7]. Next they made some experiments on the realized operating system in the multi-channel speaker system, with the result that performance improved by 60% compared with the traditional software real-time operating system.

The related research literature shows that high reliability and real-time performance of real-time operating system can be improved mainly from two aspects of hardware and software. Event management, time management, task scheduling and switching among real-time task management in real-time operating systems have caused a great deal of overhead, which is difficult to reduce by software technology, thus forming the performance bottleneck of RTOS. In order to solve this problem, we try to implement a part of functions of real-time operating system by hardware, for example, task manager can be implemented by hardware FPGA, making it a hard-core chip to execute with CPU in parallel way. Task manager by hardware implementation will undoubtedly reduce the overhead of CPU, so as to improve the task schedulability of real-time operating systems.

3. DESIGN OF MIXED REAL-TIME OPERATING SYSTEM

3.1 Reference Operating System

During the design of hardware and software mixed real-time operating system, we need to choose an open-source operating system as the reference system, which needs to have such characteristics as open-source, simple structure, easy to use, high reliability and higher real-time performance.

μC/OS-II is a strong real-time operating system written by the embedded system expert Jean J. Labrosse. It is written by C language, and its source codes are open. It has many advantages, for example, efficient, taking up less space, strong real-time performance, extensible and so on. It has been successfully transplanted to a lot of commonly used embedded microprocessors because of these advantages. In order to conduct application program development, we only need to have C compiler, assembly and link tools. In addition, it can manage 64 tasks and schedule them according to their priority. It can also make the task to turn its state from ready into executive only if it has the highest priority. Each task in the system has its own individual priority, that is, there are not two tasks which have the same priority.

3.2 The Choice of Development Tools and Construction of Platform

During the process of preparing the paper, we have used the following software tools: Xilinx Company's ISE is used for hardware simulation to obtain the hardware implementation data; EDK tools are used to compile the target software to generate executable codes, which are downloaded to run on PowerPC, thus we can get software realization data. EDK can be divided into two parts, one is software development kit SDK and the other is Xilinx platform studio XPS. The programmable chip used by our experiments is XUP Virtex-II Pro XC2VP30 produced by Xilinx Company [8].

4. HARDWARE IMPLEMENTATION OF TASK MANAGEMENT

Task management is one of the most basic functions of an operating system. First we perform the hardware design of co/OS-II task management. Its task management can be divided into the following parts: The creation, deletion, suspension, recovery, inquiry and scheduling of tasks, etc.

4.1 The Basic Operations of Task Management

The task of μC/OS-II is composed of three parts: Task program code, task stack and TCB (Task Control Block).

During the hardware implementation of task management system calls, data structures such as task code segment address, task priority, the parameter pointer of task and the stack pointer distributed to a task are all stored in TCB.

(1)Task Creation

First, ready task list is read to judge whether the task to be created already has existed, if so then a creation error will be returned, otherwise the task priority will be written into the ready list. Next the module state of the task creation is set to busy, the data to create a task is written into stack, its relevant TCB will be initialized, OS scheduler will be called and the module state of the task creation is set to free.

(2)Task Deletion

Ready list and waiting list are inquired to judge whether the current task priority already has existed, if not then a deletion error will be returned, otherwise the records will be deleted in the relevant lists. Next the Hook module will be called to clean up the task stack and TCB to return Derr. The state of current module is commonly controlled by Cerr and Derr, and the state of current module is set to free 0 after a task creation or deletion is completed.

(3)Task Suspension and Recovery

The current state of the task is changed, and the values in the ready list and waiting list are modified. During changing task states, we can find the corresponding TCB according to its ID; modify its state register value, the present value in the ready list, and the data in the waiting list corresponding to its resource application. A task scheduling will be performed after all the above operations have been completed.

We give simplified logic in Fig. 1. In order to save the hardware resources, we reuse each logic component as much as possible. The whole figure can be divided into three parts: task creation specific part, task deletion specific part, and their public part.

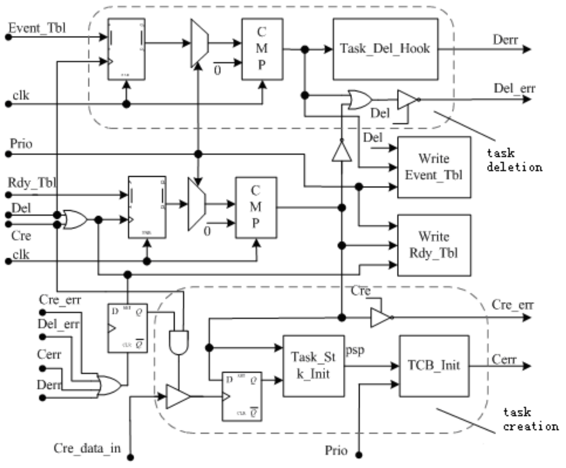

Fig. 1 Task Creation And Deletion

4.2 Ready List

Ready list and waiting list are one of the most frequently operated data structures in task management part. We put forward a more superior hardware implementation method in this paper and logic diagram is shown as Fig. 2 below.

The ready list adopts clock synchronization signal, all the storage units are set 0 by using Clr signal when it is initialized. The main operations of ready list can be divided into two kinds of reading and writing. Reading ready list can be divided into two kinds: inquiring the current highest priority task and inquiring if a task is in the ready state. In order to inquire the current highest priority task, the control circuit sends reading signal to all of the data storage units, the system can get the highest priority task according to the eight binary digits of the two outputted to the data lines, and the priority is outputted to Prior signal. On the other hand, in order to inquire if a task is in the ready state, the priority of current task is inputted as Sid signals, and is compared with the output of the current data line. If the current priority exists, then Ud signal is set high output, otherwise low. In order to write the ready list, the higher 3 bits of the priority are sent to Sid0 ~ Sid2, lower 3 bits to Sid3 ~ Sid5, the desired storage unit is chosen after decoding, and can flip the stored data according to writing signal.

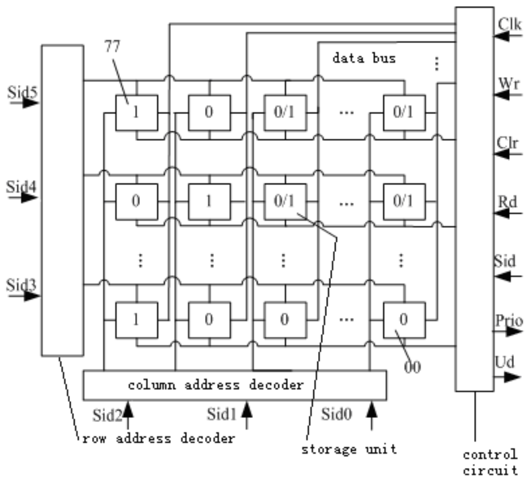

Fig. 2 Logic Diagram Of Ready List

4.3 Task Scheduling

The task scheduling of μC/OS-II is commonly called by other system functions with the result that the highest priority task can obtain the microprocessor resource. Any task state change will cause a task scheduling, but not necessarily produces switching.

Fig. 3 describes the simplified logical diagram of task scheduler. We can set the Q-output of the trigger to 0 by Set signal. If we determine the current module state is free, and not in the interrupt service state, then we can compare the task with the highest priority in the waiting list and the one which is running, if the former is lower than the latter, then we will do nothing, otherwise we will call the switch function.

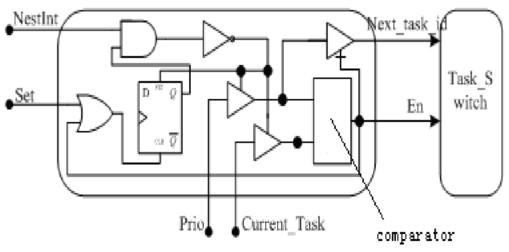

Fig. 3 Logic Diagram Of Simplified Task Scheduler

4.4 Simulation and Experimental Results

The simulation result of hardware implementation of task management is shown in Fig. 4.

(1) Create tasks

During the simulation process, we create three tasks in turn, whose priority in the system is decimal 7, 1 and 6 respectively. After we create the task whose ID is decimal 7 we conduct task scheduling, and obtain the highest priority task is decimal 7 in the ready list, therefore Next_task_id is set to 7; Next we create the task whose priority is decimal 1, and conduct task scheduling again. Due to the scheduling rule is that the higher priority task will have the right of using CPU, so Next_task_id is set to 1, and conduct task switching; Task scheduling is performed for the third time after the task whose priority is decimal 6 is created, but currently the highest priority task is running, so we don't conduct task switching.

(2)Inquire Tasks

We can complete the task information inquiry by calling functions. If the processor inquires the task whose priority is 0, then the output result includes task priority and other TCB data.

(3)Suspend tasks

We can suspend currently running task whose priority is 1, which will automatically evoke scheduling module and conduct task scheduling again, then the task whose priority is 6 will obtain CPU to run. The suspended task will keep waiting state before it is recovered.

(4)Delete tasks

Suppose we have created tasks whose priority is decimal 5, 2, 4 respectively as shown in Fig. 4. If we delete task 2, then we can trigger a task scheduling.

Because the highest priority task having the ready state is 4 in the current system, task 4 will have the right to use CPU to run.

(5)Recover tasks

If we recover the task whose priority is 1, then its state will change and be written back to ready list, thus a task scheduling will be triggered with the result that task 1 will have the right to use CPU. After task switching, the task whose priority is 1 will be executed.

We know that the system will obtain higher efficiency if task management is implemented by hardware from Fig. 4. Task creation and deletion will need three clock beats respectively; By contrast, task suspension, recovery and inquiry will need only one clock beat respectively.

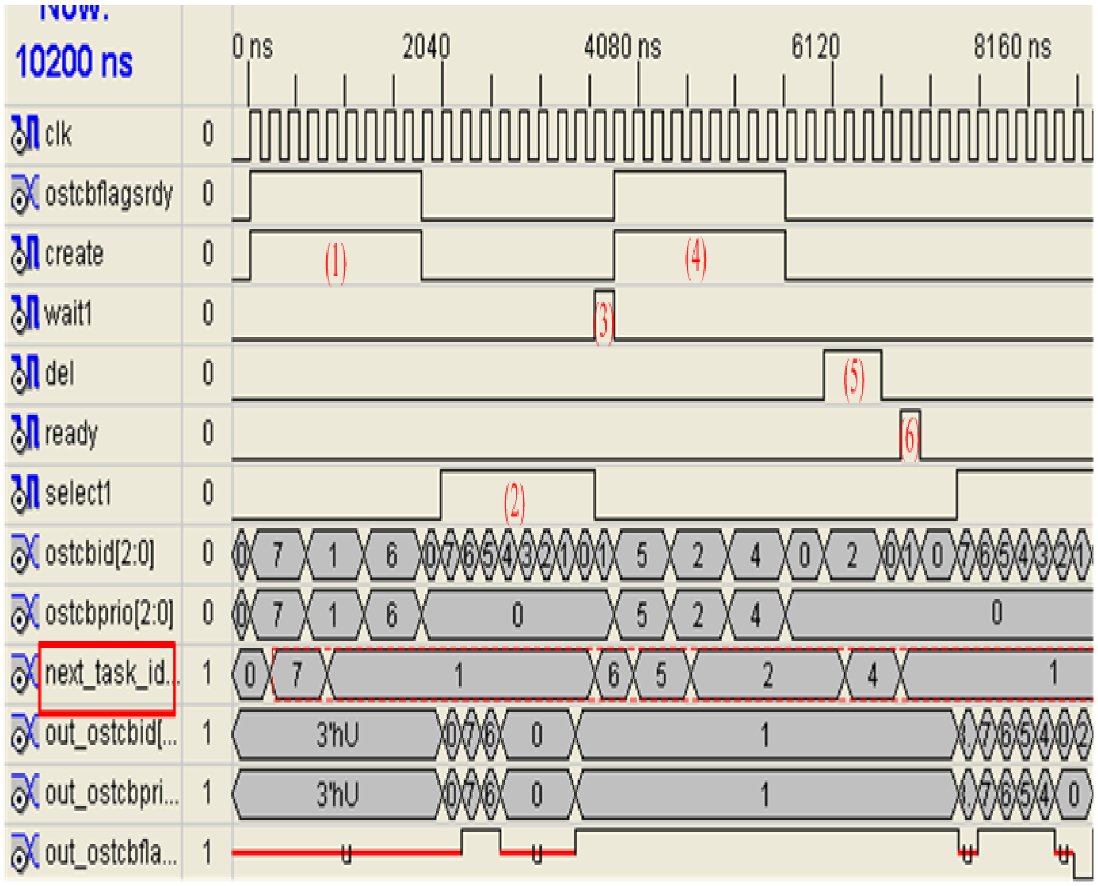

Fig. 4 Diagram Of Simulation Results

5. CONCLUSIONS AND FUTURE WORK

In view of the problem that traditional real-time operating system kernel occupies more system resources, and affects greatly the real-time performance of the whole system, we put forward the solution to implement task management by separate hardware circuits in this paper. It has been emphasized that implementing the process of task management of real-time operating system by FPGA. The simulation results show that the hardware realization of task management holds the correctness of system calls, as well as reducing corresponding execution time and the overhead of processors. Therefore, we make beneficial explorations for the realization of hardware RTOS.

At present, a significant trend is that computer software and hardware combine together closely. Softening hardware and hardening software have become two kinds of parallel development orientation of computer systems. With the development of embedded systems, it is difficult for the performance of traditional embedded real-time operating system to meet the need of high-end applications. At the same time, with the development of SOC (System On Chip) technology, hardening software has gained wide development space. We have implemented part functions of embedded real-time operating system by hardware so far, next we will consider how to finish the hardware-software partitioning of embedded real-time operating system, how to design and realize a hardware-software hybrid operating system. We believe it will be a new, very influential project.

ACKNOWLEDGMENTS

We would like to thank the research foundation of national natural science fund project of China for contract 60972127, under which the present work was made possible.

REFERENCES

[1] Adomat J, Furunas J, Indh L, et al. Real-Time Kernel in Hardware RTU: A step towards deterministic and high performance real-time systems[C].In Proceedings of eighth Euromicro Workshop on Real-Time Systems, 1996, pp. 683–688.

[2] V. Mooney III, J. Lee, A. Daleby, K. Ingstrom, T. Klevin and L.Lindth. A comparison of the RTU hardware RTOS with a hardware/software RTOS[C], Design Automation Conference (ASP-DAC' 2003).

[3] Vincent J. Mooney III. Hardware/software partitioning of operating systems[C], Design, Automation and Test in Europe Conference (DATE' 2003), vol.2, pp. 338–339.

[4] Mooney V.J, III, Blough D.M.A Hardware-Software real-time operating system framework for SOCs [J].IEEE Design and Test of Computers Magazine, 2002, 19(6):44-52.

[5] T.Nakano, U. Andy, M. Itabashi, A. Shiomi and M.Imai. Hardware Implementation of a Real-time Operating System [J]. Proceedings of the Twelfth TRON Project International Symposium, IEEE Computer Society Press, Nov, 1995. pp. 34-42.

[6] PETER WALDECK, NEIL BERGMANN. Dynamic hardware-software partitioning on reconfigurable system-on-chip [J].System-on-Chip for Real-Time Applications, 2003,30:102-105.

[7] MOONVIN SONG, SANG HONG,YUNMO CHUNG. Reducing the overhead of real-time operating system through reconfigurable hardware [C]. Digital System Design Architectures, Methods and Tools, DSD 2009: 311-314.

[8] So HK. BORPH: An operating system for FPGA-based reconfigurable computers [D]. Berkeley: University of California, 2010.