International Journal of Hybrid Information Technology
Vol.7, No.3 (2014), pp.337-344
http://dx.doi.org/10.14257/ijhit.2014.7.3.31

An Improvement of Task Scheduling Algorithms and Hardware Scheduler of Real-time Operating System

Yue-hua Li, Xue Liu,Yun-feng Ding,Hao-xin Cui,Yong-bin Du and Yan Li

The Computer and Control Institute,Harbin University of Science and Technology, China
1356074469@qq.com

Abstract

To the existing real-time operating systems implemented by software, it is hard to continually enhance their real-time performance by simply improving their scheduling algorithms. New scheduling algorithms of μC/OS-II real-time operating systems and their hardware implementation are shown in this paper. They breakthrough the limit of the number of total task, and hardware task scheduler was used to select tasks from ready list instead of the original ready table, improving the efficiency and maintaining the correctness of the real-time system. This system design adopted VHDL hardware description language and implemented and simulated in ISE 8.2. Based on the result, new task scheduling algorithms and hardware scheduler decrease the cost of hardware logic and spending of running time, while the new task management and TCB data structure keep the correctness of the original software version.

Keywords: *Task management; Lists of ready tasks; Hardware Implementation; μC/OS-II; FPGA;*

1. Introduction

With the development of embedded technology, RTOS (Real-time Operating System) has been increasingly applied in areas of embedded system, such as: aeronautics and astronautics, Industrial control, automotive electronics and nuclear power station [1]. To the existing real-time operating systems implemented by software, it is hard to continually enhance their real-time performance by simply improving their scheduling algorithms. However, using hardware logic to realize the efficiency of task scheduling, interpretation process and timer management can dramatically increase their real-time performance and certainty. The hardware logic is independent from CPU and would not cost CPU resource, so this method has the value of research and practice [2-3].

2. State of Tasks

The essential mechanism of task scheduling is the state of tasks. To understand what the highest priority task really is, one first need know the states of running tasks. In a real-time multitasks operating system, each tasks is an infinite loop that can be in any one of five states: Dormant, Ready, Running, and waiting for an event or Interrupted [5]. Any tasks can only be in these five states. The dormant state corresponds to a task that resides in memory but has not been made available to the multitasking kernel. A task is ready when it can execute but its priority is less than the currently running task. A task is running when it has control of the CPU. A task is waiting for an event when it requires the occurrence of an event (waiting for an I/O operation to complete, a shared resource to be available, a timing pulse to occur, time to expire etc.). Finally, a task is interrupted when an interrupt has occurred and the CPU is in the process of servicing the interrupt. Figure 1 shows the functions provided by μC/OS-II to make a task switch from one state to another.

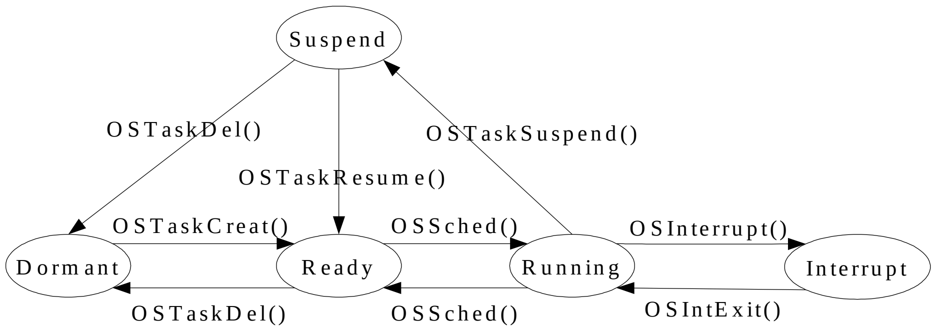

Figure 1. Task switching of five states

3. Data structure and Algorithm of Task scheduler

3.1. Task Control Blocks

The core technology for a real-time operating system is its task scheduling mechanism. In order to restore the states and other information of task running, after the establishment of every single task, a task control block (TCB) should be created to manage and schedule this task. TCB is a data structure that is used by μC/OS-II to maintain the state of a task when it is preempted [6].

So, the crux of the matter is to redesign the whole data structure of TCB to not only be compatible to μC/OS-II but suitable for implementing in the registers inside FPGA [7]. This data structure should also be easy for assigning priority based on scheduling algorithm, comparing the priorities of tasks and then sorting the tasks. According to the requirement above, the date structure can be designed as listed below.

```
type tcb is record
OSTCBId             : std_logic_vector( n downto 0);
OSTCBPrio           : std_logic_vector( n downto 0);
OSTCBStat           : std_logic_vector( n downto 0);
OSTCBDly            : std_logic_vector( n downto 0);
OSTCBStkSize        : std_logic_vector( n downto 0);
OSTCBEventPtr       : std_logic_vector(n downto 0);
OSTCBStkBottom      : std_logic_vector( n downto 0);
end record;
```

OSTCBId is the registers of task IDs in the operating system to mark every task. OSTCBStat is the registers of current state of the task; reading or writing which it can return or change the current state. OSTCBPrio is the priority of the task. OSTCBDly is the registers of delay time, used when the task needs to be delayed for a certain number of clock ticks. OSTCBStkSize is the registers of a variable that holds the size of the stack in number of elements instead of bytes. OSTCBEventPtr is the registers of a pointer to an event control block. OSTCBStkBottom is a pointer to the task’s stack bottom. This data structure only keeps the major items in TCB structure of μC/OS-II in order to be compatible with the original μC/OS-II TCB. However, because of the limit of hardware resource, some items are unnecessary to implement in FPGA register.

In the new TCB, the most fundamental data items are OSTCBPrio and OSTCBDly. When tasks are established, operating system gives a unique initial value to OSTCBId, a proper priority level to OSTCBPrio allowing the same priority level with the other tasks and initial value 0 to OSTCBDly. In the process of task running, the numbers of OSTCBId and OSTCBPrio cannot be change by tasks themselves or the operating system. At every a clock tick, OSTCBDly registers of each tasks add 1 till this task executes. The smaller the number of OSTCBPrio is, the earlier it will execute. If the number of OSTCBPrio is the same, the task with a bigger number of OSTCBPrio will execute first. These principles guarantee the operating system a preemptive kernel system, which means the highest priority task ready to run is always given control of the CPU. After the task finishes its execution, the OSTCBDly register will be reset.

3.2. Algorithm of tasks scheduling

The essential function of a real-time operating system is task scheduling, which based on algorithm of tasks scheduling, scheduler should determine which is the next task that need to execute and operate the stacks to perform the context switch [8-9]. Thus, scheduler needs to maintain a group of task lists, such as ready list and waiting list. Every time scheduler, based on priority of the task, sorted all ready-to-run tasks to select the tasks with the highest priority. Algorithm used in this paper is a modified version of the original μC/OS-II algorithm that is added algorithm of comparison of the same priority level. Thus, if the number of OSTCBPrio is the same, the task with a bigger number of OSTCBPrio will execute first, which guarantee scheduler always choses the task with the highest priority and waiting the longest time in the list.

One could see through the analysis above, because priority is corresponding to the task, TCB can be determined when the scheduler find out which is the highest priority task in the ready list. This kind of function is represented in the OSTCBPrioTbl[], so when certain priority has been determined, scheduler can index the TCB data structure from OSTCBPrioTbl[] table. While if there one priority level links multiple tasks, then instead of pointing to one single TCB data structure, the item in OSTCBPrioTbl[] points to a list of TCB data structures in which TCB data structure has the same priority levels. The connection of OSTCBPrioTbl[] and TCB items shows in Figure 2. Examples of OSTCBPrioTbl data structure.

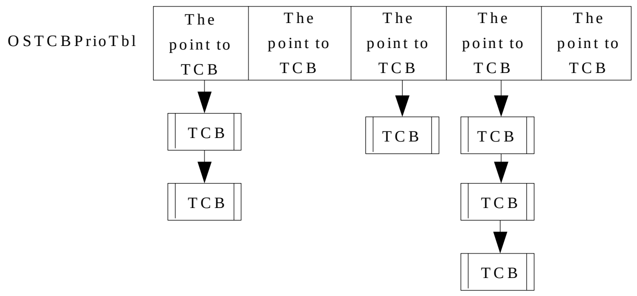

Figure 2. Examples of OSTCBPrioTbl data structure

The primary functions of task management in μC/OS-II is to create a task, delete a task, check the size of a task's stack, schedule tasks, suspend and resume a task, and get running information about a task [10]. So the input ports are to input parameter1, which is when creating of a new task, here input the address of task codes, priority level of the task, address of the stack bottom of the task and running information about the task, and parameter 2, which is the task ID number which users assigned to. The output port is to output the states of the task based on the ID number. Each ID number links to a TCB data structure. Hardware circuits of task management show in Fig. 3. Hardware circuits of task.

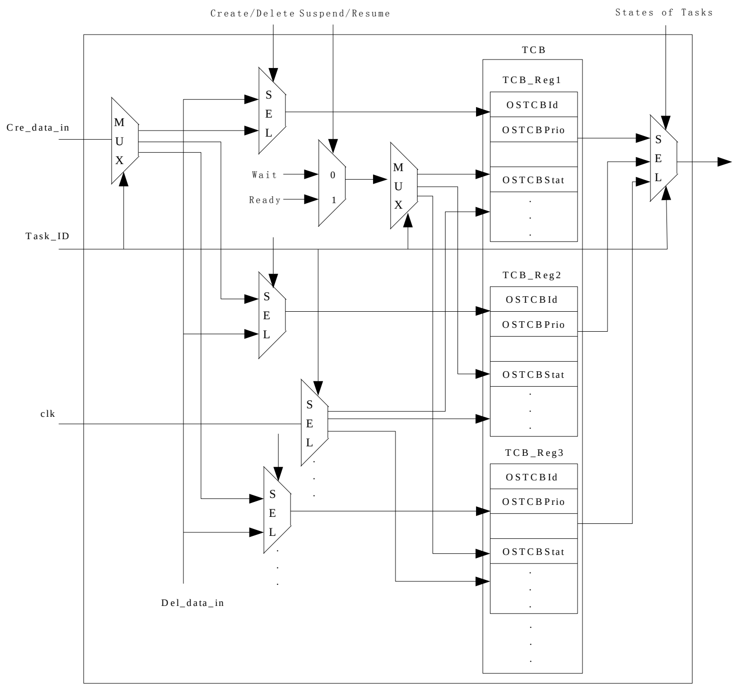

Figure 3. Hardware circuits of task management

In order to implement this algorithm in a way of high efficiency, on adopt combinational logical circuits to create this hardware task scheduler, showing in Fig. 4. Hardware circuits of the task scheduler.

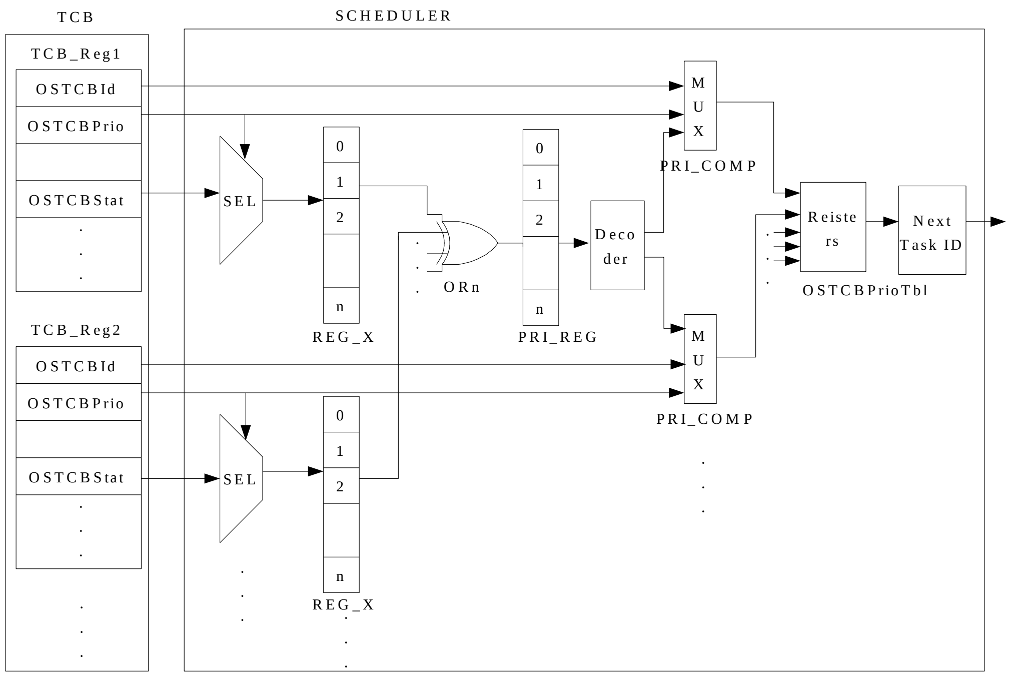

Figure 4. Hardware circuits of the task scheduler

This task scheduler sort tasks according to their priorities. It directly links to the state registers of all tasks, as long as the scheduler detects one that changes, it will cause a reschedule of all tasks. The data selectors directly connect with TCB registers. They, based on the priority level, output OSTCBStat to corresponding output bit in REG_X registers, and then all REG_X registers perform logical OR, and send the results to PRI_REG registers (definition of PRI_REG shows in Figure 5. The PRI_REG register).

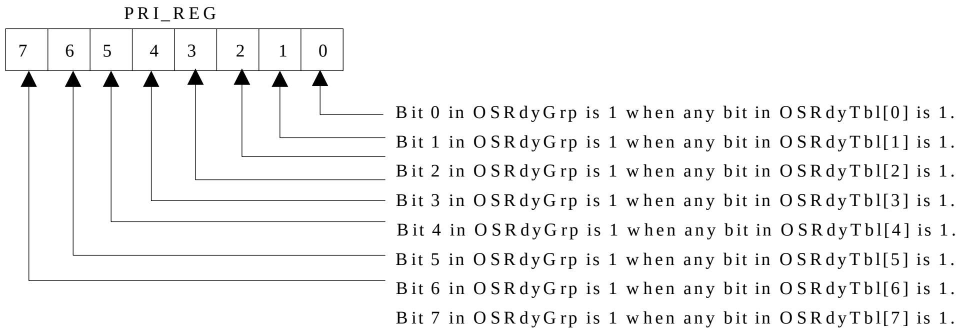
PRI_REG
7 6 5 4 3 2 1 0
Bit 0 in OSRdyGrp is 1 when any bit in OSRdyTbl[0] is 1.
Bit 1 in OSRdyGrp is 1 when any bit in OSRdyTbl[1] is 1.
Bit 2 in OSRdyGrp is 1 when any bit in OSRdyTbl[2] is 1.
Bit 3 in OSRdyGrp is 1 when any bit in OSRdyTbl[3] is 1.
Bit 4 in OSRdyGrp is 1 when any bit in OSRdyTbl[4] is 1.
Bit 5 in OSRdyGrp is 1 when any bit in OSRdyTbl[5] is 1.
Bit 6 in OSRdyGrp is 1 when any bit in OSRdyTbl[6] is 1.
Bit 7 in OSRdyGrp is 1 when any bit in OSRdyTbl[7] is 1.

Figure 5. The PRI_REG register

The number of bits in PRI_REG register is equal to the number of priority levels. If the bit in the register shows 1, it means the task that the bit is linked to is at ready state. The priority of the task of high bit is higher than that of low bit. Decoders read the values of PRI_REG in registers, decode them and them send the highest priority out. Based on the index of task IDs and the priorities PRI, comparer PRI_COMP will compare each of priority number of the ready tasks to tasks with highest priority number. If they are the same, then output he ID number of the task to OSTCBPrioTbl register, otherwise output 0 to OSTCBPrioTbl register. If there are more than one tasks with highest priorities, then output the task that has been waiting the longest time in the waiting list.

4. The results and analysis

In order to test the correctness and high efficiency of the scheduling algorithm shown in the paper, the whole system has been implemented by VHDL hardware description language simulated in ISE 8.2 design suite. The functional simulation of hardware task management and scheduler shows in Figure 6. The simulating graph of task management and scheduler.

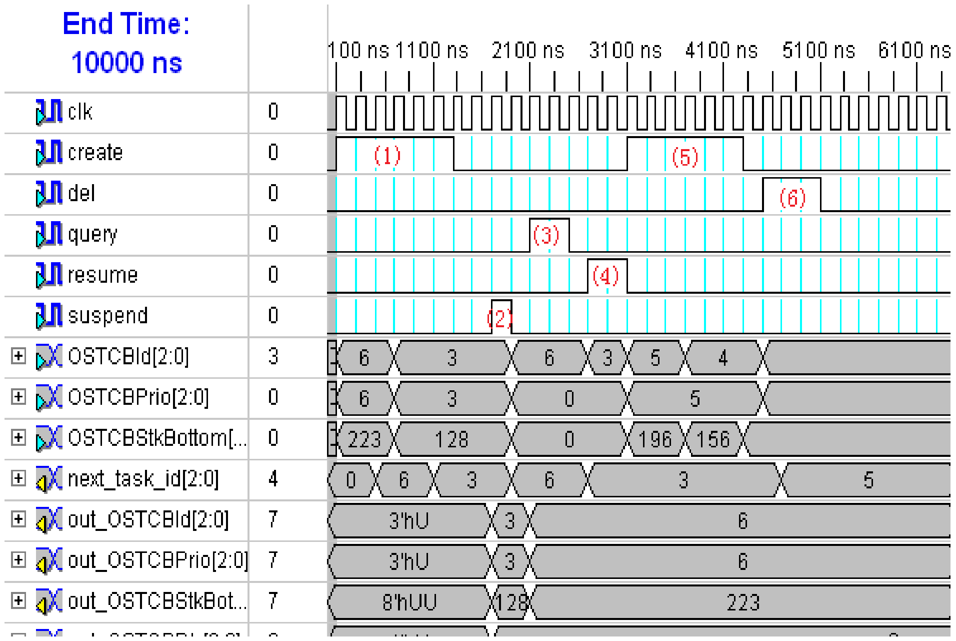

Figure 6. The simulating graph of task management and scheduler

The results of simulation are:

1. First create a task of ID number 6 and priority number 6. At this time, there is not any other tasks running in the operating system, so when the clock comes, the value of Next_task_id register is 6. And then create a task of ID number 3 and priority number 3, which is higher than task 6, so the scheduler preempts the running state of the task 6 and give the control of CPU to task3, that is Next_task_id is 3.

2. Suspend the task 3, which will cause rescheduling. At this time, task 6 is the task with the highest priority, so execute task6.

3. Give ID number 3 and 6 to the task management, which will return the states of TCBs of task 3 and task 6. The return will be send to the current state register.

4. Resume task 3, that is task 3 is the task with the highest priority, so the scheduler preempts the running state of task 6. Task 6 will continue to execute.

5. Create two tasks of ID number 5 and 4 as well as the same priority number 5.

6. Delete task 3. At this time, there are two tasks waiting in the ready list and task 5 waits longest in the ready list. Thus, the ready-to-run task is task 5.

Suppose there are 8 tasks running in the operating system, the usage of hardware resources in entire system show in Table 1. The practical usage of hardware in

XC2VP30FF896C platform. According to Table 1, resources of FPGA and performance meet the need of real-time operating system.

Logic Utilization	Used	Available	Utilization
Number of Slices	491	13696	3%
Number of Slices Flip Flops	712	27392	2%
Number of 4 input LUTs	350	27392	1%
Number of bonded IOBs	81	556	14%
Number of GCLKs	8	16	50%

Figure 7. The practical usage of hardware in XC2VP30FF896C platform

The analysis of the results verifies the correctness and high efficiency of using hardware logic to implement operating system scheduler, which meets the requirement of real-time operating system.

5. Conclusion

Based on fully analyzing the mechanism of task scheduling of real-time operating system as well as the merits of FPGA hardware logic, this paper designs the algorithm and logic circuits of hardware task management and scheduler instead of traditional software one and implement and simulate the whole system on ISE 8.2 of Xilinx Company. This paper mainly redesigns the task management and schedulers to meet the limit of hardware logic as well as high efficiency and improve the algorithm by adding ready lists, which will support tasks with the same priority numbers. So, this improvement of algorithm breaks through the limit of the number of total task and makes possible run some tasks that are not necessary to be strict real-time. Because of the new ready list, the paper redesigns the scheduler to select the task with the same priority that waits the longest in the read list in order to support tasks with the same priorities. This paper use registers inside FPGA to implement the redesigned TCB data structure, which maximizes the potential parallel performance of multiple tasks. Instead of using ready table originally used in μC/OS-II, this paper use ready list and directly links it to the scheduler, saving time to read from memory. In a word, hardware task management and scheduler have its value of research and real life.

Acknowledgement

This study was supported by the National Natural Science Foundation of China (No. 61103149), the Education Department Foundation Heilongjiang Province of China (No. 12521100), the Technology Innovation Talent Research Foundation of Harbin (No. 2013RFXXJ034) and Innovative and experimental program for college students of china.

References

[1] Y. Zhenyu, Z. Hai, W. Jinying, X. Jiuqiang and L. Kai, “HOS Design on Embedded Processor”, Computer Engineering, (**2008**), pp. 268-270.

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

[3] M. Vetromille, L. Ost, C. A. M. Marcon, C. Reif and F. Hessel, “RTOS Scheduler Implementation in Hardware and Software for Real Time Application”, Rapid System Prototyping, (**2006**), Seventeenth IEEE International Workshop, 163 - 168 Digital Object Identifier 10.1109/RSP.2006.34.(2006)

[4] L. Yan, C. Xiaoying, L. Xianyao, Z. Hongjie, C. Ping and Z. Liyong, “Hardware implementation of task management of μC/OS-II based on FPGA”, Application of Electronic Technique, vol. 02, (2010), pp.25-29.

[5] G. Fuqiang, Q. Changshuo, Y. Jiyuan and Z. Heng, “Extension of time slice circular scheduling in UC/OS-II kernel”, Journal of Computer Applications, (**2009**), pp. 1128-1130+1142.

[6] J. J. Labrosse, “The Real-Time Kernel”, Translated by Shao Beibei and so on, Beijing: Beijing University Press, (**2001**), pp. 178-185.

[7] L. Yan, J. Xiaoli and C. Huanhuan, “The research and hardware design of interrupt manager based on FPGA”, Application of Electronic Technique, vol. 09, (**2011**), pp.49-52.

[8] C. Jianhua, S. Hongsheng and W. Baojin, “The design and realization of hardware real-time operating system”, The of application computer technology, (**2008**), pp. 34-37.

[9] S. Moon, J. Rexford and K. Shin, “Scalable hardware priority queue architectures for high-speed packet switch, IEEE Transactions on Computer”, vol. 49, no. 11, (**2000**) , pp. 1215-1227.

[10] S. Mori, “The Present and Future of the TRON-specification CHIP-Promoting Open Architecture and Standardization”, Journal of formation Processing Society of Japan, vol. 35, no. 10, (**1994**).