FPGA-based implementation of task management in μC/OS-II operating system

Shihai Zhu[*]

Institute of Information Engineering and Art Design, Zhejiang University of Water Resources and Electric Power, Hangzhou, China

Received 23 November 2014, www.cmnt.lv

Abstract

Task management is one of the most basic functions of an operating system. We took μC/OS-II real-time operating system as an example in this paper, put forward hardware design scheme of task management based on FPGA, and carried out the simulation and verification by means of Xilinx ISE software. We mainly designed and implemented hardware logical circuits of task management module so that the potential parallelism of multitasking was greatly improved; In the meantime hardware logical circuits of interrupt task management module was also designed and implemented. Specifically speaking, as interrupt tasks, external interrupt requests enjoyed higher priorities than those of ordinary tasks. If external interrupt arrived, then corresponding task was set to ready state, thus task scheduling was triggered, and then interrupt task was given higher priority for processing in order that the response time was improved. The simulation results showed that task management implemented by hardware could obviously reduce the executing time of a task, thus greatly expanded the application ranges of μC/OS-II operating system.

Keywords: RTOS, FPGA, task management, task scheduling, interrupt

1 Introduction

Jaehwan Lee et al. put forward the concept of hardware real-time operating system (HRTOS) from the 1980s, and proposed that if specific hardware IP core was used to realize RTOS scheduler then the efficiency of RTOS would be greatly improved [1]. We all know that task scheduling is the bottleneck of affecting the performance of RTOS. If task scheduling is implemented by hardware, there is no doubt that its performance can be improved greatly, thus the performance of the whole RTOS is improved accordingly [2-5]. Takumi Nakano [6] 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 ensuring high reliability of real-time operating system. Prof. 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 [7]. Moonvin Song, Sang Hong, Yunmo Chung at Kyunghee 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 switching operations by hardware among the most time-consuming task switching process and also interrupt handling [8]. Next they made some experiments on the realized operating system in the multi-channel speaker system, with the result that its performance was improved by 60% compared with traditional software real-time operating system. Mellissa Vetromille and Luciano Ost [9] in Brazil realized RTOS scheduler by means of software, co-processor and hardware. Next, they compared and analysed the performance of RTOS scheduler under the above three cases. Finally they drew a conclusion that hardware scheduling model has higher performance. Specifically speaking, hardware scheduler model consisted of scheduler core, task management and communication interface. Paul Kohout, Brinda Ganesh and Bruce Jacob [10] at Maryland University in the U.S. have realized real-time task manager (RTM) by hardware. RTM fully tapped the potential parallelism of multi-tasking, so as to minimize the overhead of real-time operating system, the processing time of the whole system caused by using RTOS was reduced up to 90%, while the response time was reduced up to 81%. Lounis Kessal et al. at French national higher electronic power application engineering school put forward that the optimized reconfigurable logical core was embedded in on-chip system, using hardware instead of original software to realize the scheduling algorithm of real-time operating system. The experimental results showed that using dynamic reconfigurable logical core instead of software to perform corresponding functions would greatly improve the performance of RTOS.

Embedded real-time operating system (ERTOS) has been widely used in many fields and has become more and more important especially in those applications of complex functions and huge system. However, Traditional

[*] ***Corresponding author's*** e-mail: zhushh@zjweu.edu.cn

software real-time operating system (RTOS) compiles and executes its core with application programs together, which will occupy storage space and processing time of applications, and influence executing efficiency of applications. As to such occasions as higher requirements for real-time performance, it is harder to make its real-time performance better simply by relying on improved scheduling algorithm. Besides, increasing the speed of CPU also cannot achieve ideal effect because its speed has reached a certain height [11, 12]. Seen from the development trend of computer industry, the scale of integrated circuits has become larger and larger, at the same time the cost has been greatly reduced, therefore, software hardening has become more and more popular. With the development of large scale integrated circuits and software hardening technology, the boundaries between software and hardware of a computer system has become blurred [13-15]. Logically speaking, hardware is equivalent to software, that is to say, any operation can be performed by software or hardware. Obviously, software and hardware mixed operating system provides us with a better solution. Specifically speaking, a part of functions of an operating system are implemented by hardware, and those functions not suitable for hardware implementation are still implemented by software. As is known to all, task management is one of the most basic functions of an operating system. In this paper, we took µC/OS-II as an example to implement hardware design based on FPGA of task management, especially hardware design of interrupt task management.

2 The design of software and hardware mixed operating system

2.1 SOFTWARE AND HARDWARE CODESIGN

Usually serial instruction streams are always adopted during software programming model for the description of given problems, which is based on the thinking process of problem solving for human beings that any problem can be solved within limited time through multiple operations, but this model can't simulate parallel behaviors of a system. The biggest advantage of hardware system lies in its stronger parallel processing ability, but it is difficult for human beings to simulate this parallel characteristics, therefore hardware design needs to be performed through combining and building. The huge difference between software and hardware design leads to the separation of software and hardware design for a long time. In order to satisfy the constraints and design requirements of different operating system, we must design the mixed operating system by adopting the method of software and hardware co-design to obtain higher efficiency, lower energy consumption and greater flexibility. As the two bases of computer system, hardware and software can both restrict and rely on each other, at the same time can be transformed mutually. That is to say, some software operations can be implemented by hardware, and also part of hardware operations can be performed by software [16, 17], which is known as software hardening and hardware softening.

2.2 THE DESIGN OF SOFTWARE AND HARDWARE MIXED OPERATING SYSTEM

The software of mixed operating system is run by the microprocessor, meanwhile hardware IP core works in parallel with the microprocessor. The software part also includes the interactions between it and the hardware IP core in addition to some necessary functions, whose main functions contain two parts: one is to provide the interface functions to access the hardware for the software part; the other is to provide interrupt handle functions and return handle results for the hardware logic. For example, if the current system has tasks A and B, then the simulation diagram of system calls between software and hardware modules is shown as Figure 1.

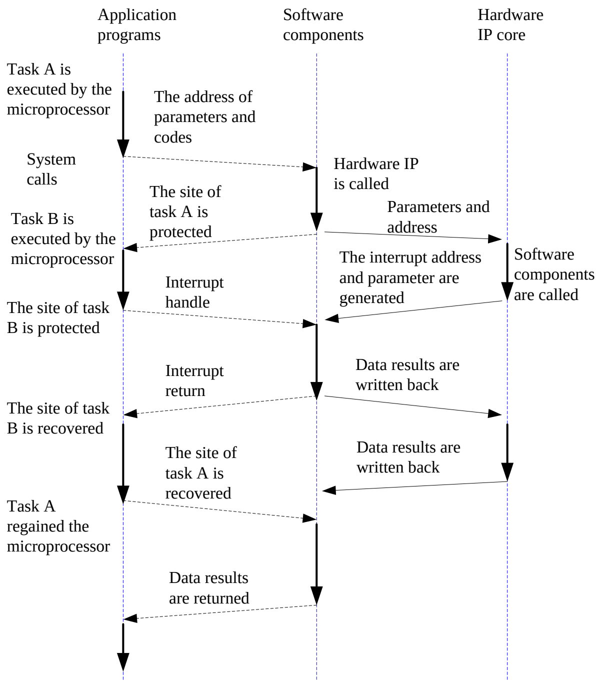
Application programs
Software components
Hardware IP core
Task A is executed by the microprocessor
The address of parameters and codes
System calls
Hardware IP is called
The site of task A is protected
Parameters and address
Task B is executed by the microprocessor
Software components are called
The interrupt address and parameter are generated
Interrupt handle
The site of task B is protected
Interrupt return
Data results are written back
The site of task B is recovered
The site of task A is recovered
Data results are written back
Task A regained the microprocessor
Data results are returned

FIGURE 1 The simulation diagram of system calls between software and hardware modules

First, task A makes a system call, and gives the parameter and code address of called function module. If all the called function modules are implemented by software, then the system jumps straight to corresponding code address to execute, and returns the results after the completion. If the called function modules need to call hardware IP core during the software implementation process, then it will finish the following operations: First of all, it will query corresponding status registers of IP core to judge whether the IP core is busy or not. If the IP core is busy then the program will be suspended a certain number of clock cycles; On the contrary, if the IP core is not busy then control signal is sent by rewriting the control register of hardware IP core, meantime the parameters are written into data registers of hardware IP core, and triggers executing, rewrites the status registers, write the data results into data registers after executing. After task A regains the microprocessor, it will read back the results by confirming the results have been written into data registers. Task A can be suspended a few clock cycles during hardware IP core is executing, thus the microprocessor can execute other tasks, such as task B in this example. As soon as the suspended time arrives, task A is entered back into the ready list for scheduling. The operations of hardware IP core here are similar to the critical resources of an operating system.

The software part of hardware IP core calls can be implemented by interrupt. When an interrupt occurs, the microprocessor will stop executing the current task, save the sites, and enter into corresponding interrupt handle program according to the interrupt number triggered by hardware IP core. At the same time hardware IP core will not wait until the software part completes writing data into control or data registers. The microprocessor will return to original task to execute after it finishes corresponding interrupt process.

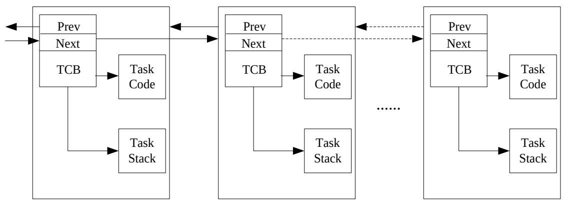

FIGURE 2 The task structure of μC/OS-II in memory

3 Hardware implementation of task management in μC/OS-II operating system

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

3.1 TASK STRUCTURE AND SYSTEM CALLS OF MC/OS-II

Each task of μC/OS-II is composed of three parts, they are task program code, task stack and TCB (Task Control Block). Among them, TCB is used to store the task properties; Task stack is used to store the working environment of a task. The task code is the executed part of a task. TCB is the basic attribute of a task managed by the operating system. When the right of a task to use CPU is deprived, Its TCB is used to store the task state in μC/OS-II operating system. When the task regains the right to use CPU, Its TCB will confirm that the task can be executed faithfully from the breakpoint. In order to facilitate management, μC/OS-II treats each task as a node, and then links into a task list, as shown in Figure 2. The system call functions of task management of μC/OS-II are shown as Table 1, which are listed below: create, delete, suspend, restore and query a task, etc.

TABLE 1 The system call functions of task management module

System call functions	Performed functions
OSTaskCreate (void(* task) (void* pd), void * pdata, INT8U prio)	Create a task
OSTaskDel (INT8U prio)	Delete a task
OSTaskPend (INT8U prio)	Suspend a task
OSTaskPost (INT8U prio)	Restore a task
OSTaskQuery (INT8U prio)	Query a task

3.2 THE BASIC OPERATIONS OF TASK MANAGEMENT

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 its TCB.

1) Create a task.

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

2) Delete a task.

Ready list and waiting list are inquired to judge whether the current task priority has already existed, if not then a deletion error will be returned, otherwise the records will be deleted in the relevant lists. Next corresponding Task_Del_Hook module will be called to clean up the task stack and its 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 0 after the creation or deletion of a task is performed.

3) Suspend and Restore a task.

Specifically speaking, the current state of the task is changed, and the values in the ready list and waiting list are also modified. We all know that we can find corresponding TCB according to the ID of a task, thus the state of the task can be changed by modifying the value of corresponding state register. In the following, the present value in the ready list and the data in the waiting list corresponding to resource request are changed. A task scheduling will be triggered after all the above operations have been performed.

We give simplified logic in Figure 3. 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: specific part of task creation, specific part of task deletion and their public part.

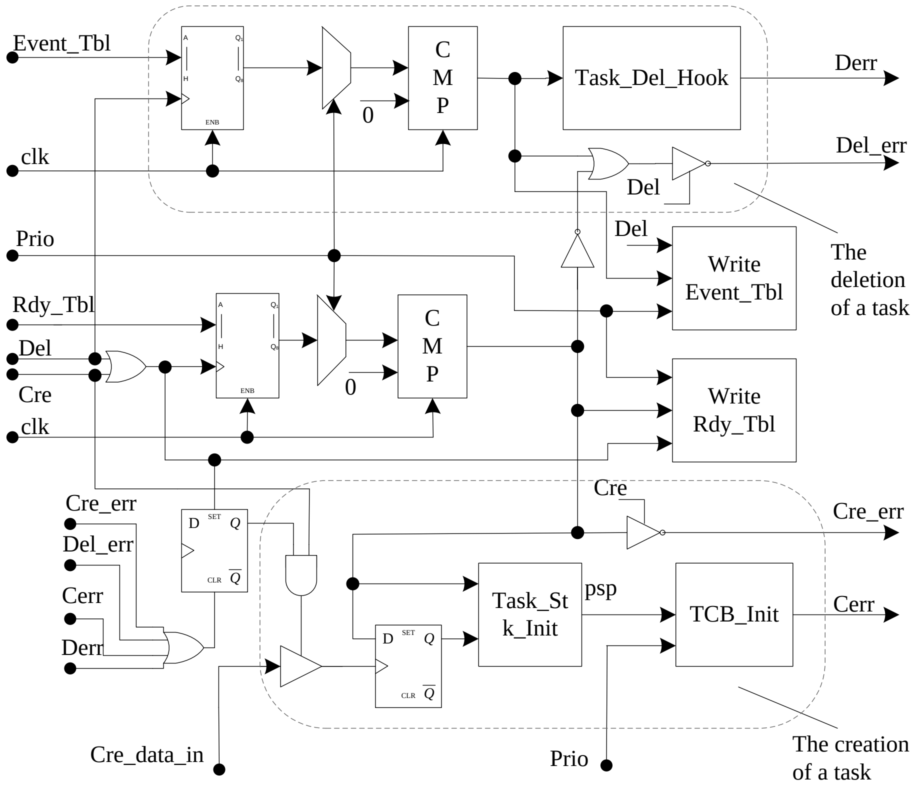

FIGURE 3 The creation and deletion of a task

3.3 READY LIST

Like waiting list, ready list is one of the most frequently operated data structures in task management. We put forward a more superior hardware implementation method in this paper and logical diagram is shown as Figure 4. The ready list adopts clock synchronization signal, all the storage units are set 0 by using Clr signal when they are initialized. The main operations of ready list can be divided into two kinds of reading and writing. Reading ready list can be further divided into two kinds: inquiring the current highest priority task to find if a task is in ready state. The control circuit sends reading signal to all of data storage units for this purpose, then the system can get the highest priority task according to two binary eight digits outputted to the data lines, and its priority is outputted to Prio signal. On the other hand, in order to inquire if a task is in ready state, the priority of current task is inputted as Sid signal, and is compared with the output of the current data line. If 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 the stored data can be flipped according to writing signal.

3.4 TASK SCHEDULING

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 change of a task state will trigger a task scheduling, but not necessarily produces switching. Figure 5 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 state of current module 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 switching function.

3.5 HARDWARE DESIGN OF INTERRUPT TASK MANAGEMENT

1) The overall design of interrupt task management module.

The internal structure of interrupt management module is shown as Figure 6. Interrupt management module is composed of interrupt request module and interrupt task control block (INT_Task_TCB). Interrupt request module receives external interrupt request signals, and finds out such interrupt request which has the highest priority and has not been masked, then it responds to IRi request, sends interrupt request signal INT, and outputs the approved interrupt vector number after the arbitration as the basis for CPU to find the entry point of interrupt service routine. INT_Task_TCB will assign the INT signal to the state bit of corresponding interrupt task according to interrupt vector number, and set corresponding interrupt task to ready state. Both interrupt tasks and ordinary tasks are together scheduled by the scheduler. Besides, the priority of interrupt task is higher than that of ordinary task, therefore, interrupt task will gain the right to use CPU and be performed immediately.

2) Interrupt request module.

As shown in Figure 7, interrupt request logic is composed of five modules, which are listed respectively below: interrupt request register, the priority encoder, interrupt service register, interrupt mask register and comparator. The function of interrupt request module is to queue the inputted interrupt requests, judge their priorities, and store the priority of interrupt request which is being served [18,19].

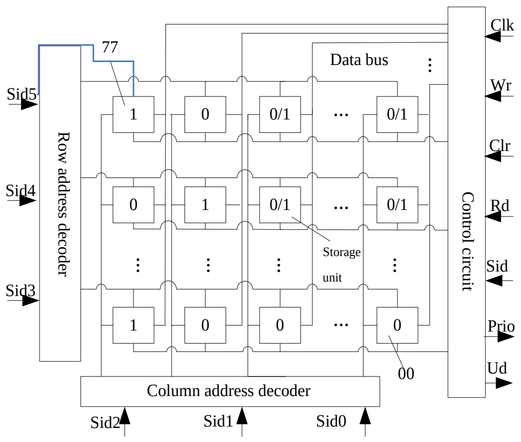

FIGURE 4 Logic diagram of ready list

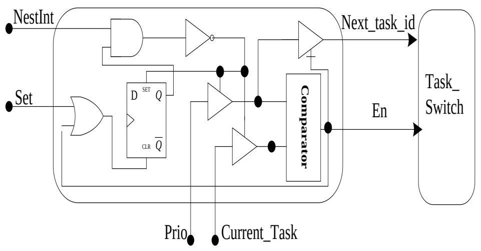

FIGURE 5 Simplified logical diagram of task scheduler

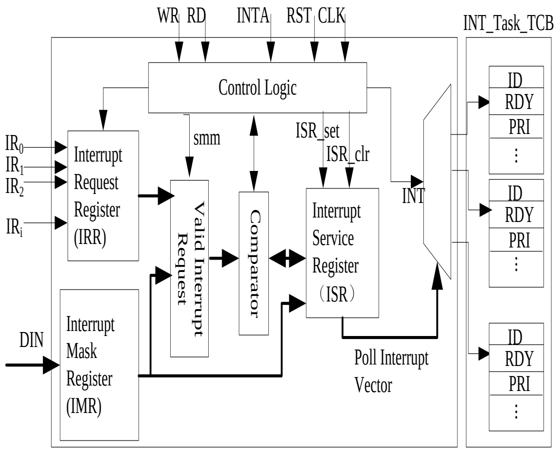

FIGURE 6 The internal structure of interrupt management module

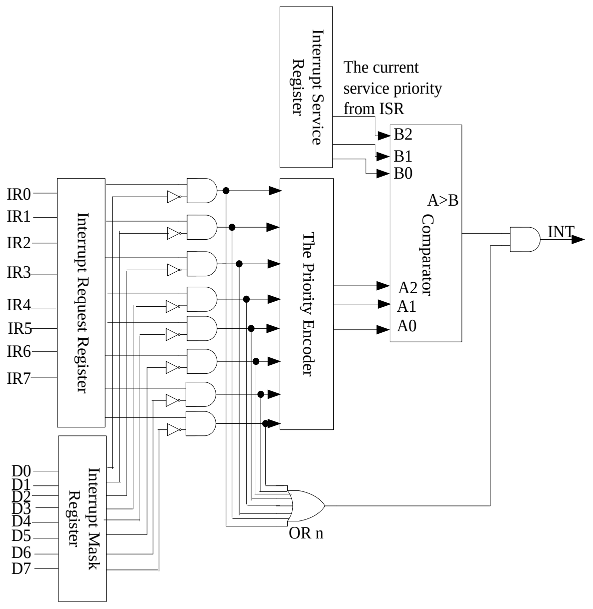

FIGURE 7 Logical circuit of interrupt request processing

3.6 SIMULATION AND EXPERIMENTAL RESULTS

1) The simulation results of interrupt task management. First, we programmed the codes for interrupt task management by VHDL language; then synthesized, debugged and simulated by Xilinx ISE software. Finally the simulation results of interrupt request register, interrupt priority comparator and interrupt service register are shown as Figure 8, Figure 9, and Figure 10. The simulation waveform diagram of interrupt request register is shown as Figure 8. First, mrst_n=0, the system is reset and all of the system registers are cleared to zero. If IR1, IR3 and IR4 of IR register are set to high voltage, then D1, D3, and D4 of IRR are all set to 1 to latch the three interrupt requests. At the same time D3 of IMR is set to 1, that is to say, the system masks the interrupt request offered by IR3. Then D1 and D4 of masked_irr [7:0] are set to 1, indicating that the system can receive effective interrupt requests offered by IR1 and IR4.

The simulation waveform diagram of interrupt priority comparison is shown as Figure 9. D1 and D4 of IRR are set to 1, indicating that IR1 and IR4 offer interrupt requests. At the same time, D6 of the current service register is set to 1, indicating that the system is executing the interrupt request offered by IR6. Because the priority of IR1 is higher than that of IR6, the system responds the interrupt request offered by IR1 and sets INT=1, at the same time set_isr [7:0] saves the current highest priority interrupt request.

The simulation waveform diagram of interrupt service register is shown as Figure 10. When isr_set_stb=1, the value saved by set_isr [7:0] is rewritten into ISR, and the current interrupt vector number is outputted by poll_vector [7:0]. If poll_vector [7] = 1, then it indicates that there is an interrupt request. On the contrary, If poll_vector [7] = 0, then it indicates that there isn't an interrupt request.

poll_vector [2:0] is the number of interrupt which is being served. If responded interrupt request currently is IR1, then poll_vector [2:0] is set to 001. Next the system finds the location of an interrupt task control block by taking the interrupt vector number as selecting criteria, and sets corresponding state bit to 1, indicating that it is ready.

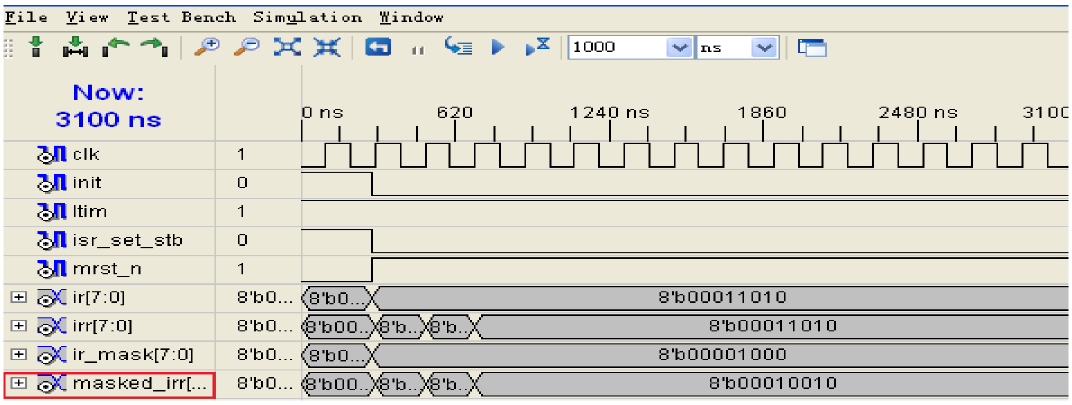
File View Test Bench Simulation Window
1000 ns
Now: 3100 ns
0 ns 620 1240 ns 1860 2480 ns 3100
clk 1
init 0
ltim 1
isr_set_stb 0
mrst_n 1
ir[7:0] 8'b0... 8'b0... 8'b00011010
irr[7:0] 8'b0... 8'b00... 8'b.. 8'b.. 8'b00011010
ir_mask[7:0] 8'b0... 8'b0... 8'b00001000
masked_irr[... 8'b0... 8'b00... 8'b.. 8'b.. 8'b00010010

FIGURE 8 Simulation waveform diagram of interrupt request register

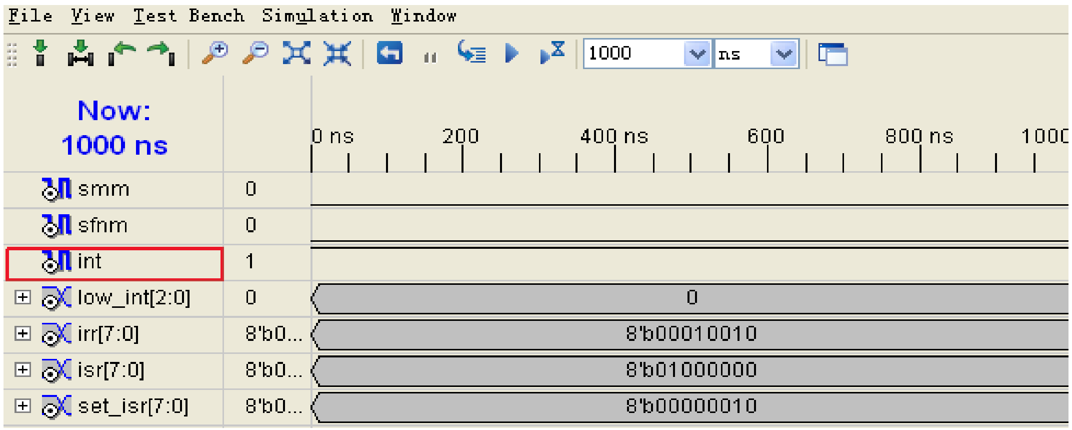
File View Test Bench Simulation Window
1000 ns
Now: 1000 ns
0 ns 200 400 ns 600 800 ns 1000
smm 0
sfnm 0
int 1
low_int[2:0] 0 0
irr[7:0] 8'b0... 8'b00010010
isr[7:0] 8'b0... 8'b01000000
set_isr[7:0] 8'b0... 8'b00000010

FIGURE 9 The comparison of interrupt priority

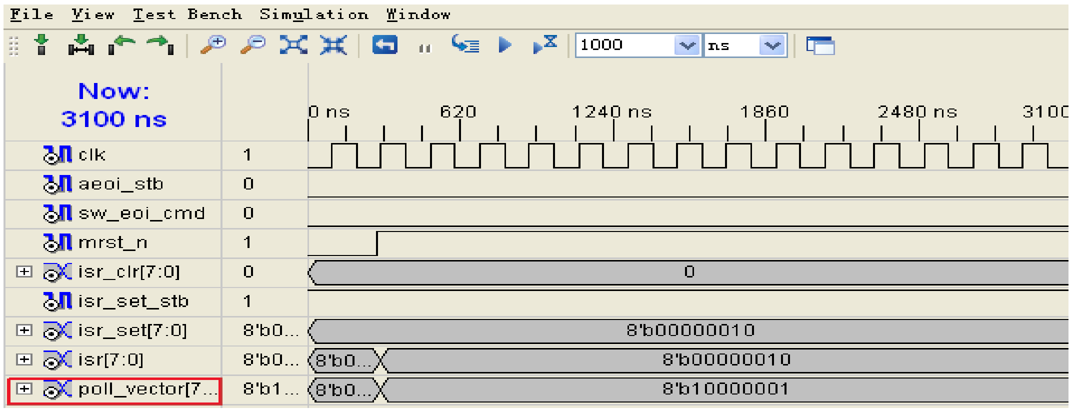
File View Test Bench Simulation Window
1000 ns
Now: 3100 ns
0 ns 620 1240 ns 1860 2480 ns 3100
clk 1
aeoi_stb 0
sw_eoi_cmd 0
mrst_n 1
isr_clr[7:0] 0 0
isr_set_stb 1
isr_set[7:0] 8'b0... 8'b00000010
isr[7:0] 8'b0... 8'b0... 8'b00000010
poll_vector[7... 8'b1... 8'b0... 8'b10000001

FIGURE 10 Simulation waveform diagram of interrupt service register

2) The simulation result of hardware implementation of task management.

The simulation result of hardware implementation of task management is shown as Figure 11.

(1) *Create a task.* 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 perform 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 perform 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 perform 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 perform task switching.

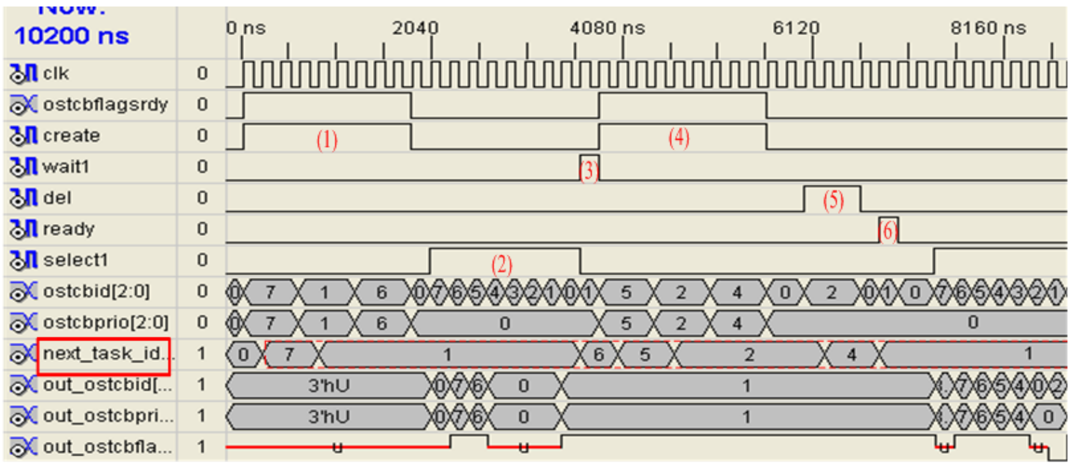

FIGURE 11 Simulation waveform diagram of task management implemented by hardware

(2) *Inquire a task.* We can complete 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 a task.* We can suspend currently running task whose priority is 1, which will automatically evoke scheduling module and perform 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 restored.

(4) *Recreate a task.* Based on the above description, three tasks whose priority is decimal 5, 2 and 4 respectively are created. System scheduling will be triggered after the task whose priority is 5 is created, with the result that the task whose priority is 6 is deprived of the right to use CPU, and the task whose priority is 5 will gain the right to use CPU. In the following, another system scheduling will be triggered after the task whose priority is 2 is created with the similar result that the task whose priority is 5 is deprived of the right to use CPU, and the task whose priority is 2 will gain the right to use CPU and start executing. Similarly, the creation of a task whose priority is 4 will also trigger additional task scheduling. Because its priority is lower than that of the task whose priority is 2, it is set to ready state and it can't preempt the right to use CPU.

(5) *Delete a task.* Suppose we have created tasks whose priority is decimal 5, 2 and 4 respectively as shown in Figure 11. If we delete task 2, then we can trigger a task scheduling. Because task 4 has highest priority and is in ready state in the current system, task 4 will gain the right to use CPU to run.

(6) *Restore a task.* If we restore 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 Figure 11. The creation and deletion of a task will need three clock beats respectively; By contrast, the suspension, restoration and inquiry of a task will need only one clock beat respectively.

4 Discussion

On [20], a new three-level resource management that is based on two methods is presented. That is to say, a complete analytic method and an approximate iterative method. For both methods, the placement quality is measured by the rate of resource efficiency and by the amount of configuration overhead. But the dependency between tasks should be investigated. Inter-task communication might be an important criterion in deciding on the most optimal RZ fitting.

In [21], a complete model and implementation of the lightweight and portable OS4RS supporting preemptible and clock-scalable HW tasks was presented. While DFS was discussed in the context of FPGAs by the previous works, none of them proposed a complete model and implementation of the OS4RS architecture supporting this concept. The DFS allows improving performance of the SW-HW code signed applications and avoid some of the restrictions imposed by the underlying DPR technology. But the overall performance results could be further improved if better HLS tools were used.

In [22], a new approach for scheduling and placement of task on a dynamic reconfigurable device based on reflected binary gray space filling curve method is being presented with the goal of minimizing task rejection ratio and increasing FPGA utilization. The free space is managed as one dimensional run-length based representation. Also, a new method to find the fragmentation is used. But the algorithm does not consider routability, I/O communication, and heterogeneous FPGA. The algorithm can be improved to reduce the total reconfiguration overhead by reusing some of the task locations.

In [23], a novel scheduling algorithm and two novel allocation heuristics have been presented in the scope of R3TOS project. The effectiveness of the proposed algorithms is tested by means of a wide range of synthetic simulations, but the evaluation of these approaches in a real-world application is still under way.

5 Conclusions

We fully understood the present situations and development trend of real-time operating system at home and abroad, combined with the principle of μC/OS-II and characteristics of FPGA technology, then put forward the design scheme of hardware RTOS based on FPGA. Using FPGA to realize the process of task management module was emphasized. Specifically speaking, we implemented the creation and deletion of a task, designed the logical circuits of ready list and interrupt task management by hardware. We also completed system debugging and functional verification by means of Xilinx ISE software. The simulation results showed that the implementation of task management by hardware kept the correctness of system calls, at the same time reduced the execution time of system calls and the overhead of CPU.

Acknowledgments

We would like to acknowledge the partial support given by the national natural science foundation project of China (No. 60972127).

References

[1] Mooney V III, Lee J, Daleby A, Ingstrom K, Klevin T, Lindth L 2003 A comparison of the RTU hardware RTOS with a hardware/software RTOS *Proceedings of Asia South Pacific Design Automation Conference (ASPDAC'2003)*-12
[2] Mooney V III, Blough D M 2002 *IEEE Design and Test of Computers* **19**(6) 44-52
[3] Apostolos P Fournaris, Nicolas Sklavos 2014 Secure embedded system hardware design – A flexible security and trust enhanced approach *Computers and Electrical Engineering* **40** 121-33
[4] Swarnalatha A, Shanthi A P 2014 Complete hardware evolution based SoPC for evolvable hardware *Applied Soft Computing* **18** 314-22
[5] So HK Borph 2010 An operating system for FPGA-based reconfigurable computers *PhD dissertation, Berkeley: University of California*
[6] Nakano T, Andy U, Itabashi M, Shiomi A, Imai M 1995 Hardware Implementation of a Real-time Operating System *Proceedings of the Twelfth TRON Project International Symposium IEEE Computer Society Press* 34-42
[7] Waldeck P, Bergmann N 2003 Dynamic hardware-software partitioning on reconfigurable system-on-chip *Journal of System-on-Chip for Real-Time Applications* **30**(2) 102-5
[8] Song M, Hong S, Chung Y 2009 Reducing the overhead of real-time operating system through reconfigurable hardware *Proceedings of Digital System Design Architectures, Methods and Tools (DSD 2009)* 311-4
[9] Vetromille M, Ost L, Maroon C A M, et al. 2006 RTOS Scheduler Implementation in Hardware and Software for Real time Applications *Proceedings of the seventeenth IEEE international workshop on rapid system prototyping (RSP'06)* 163-8
[10] Kohout P, Ganesh B, Jacob B 2003 Hardware Support for Real-time Operating Systems *Proceedings of Automation and Test in Europe Conference (DATE'03)* 45-51
[11] Ycho, Syoo, Kchoi 2005 Scheduler implementation in MPSoC Design *Proceedings of Asia South Pacific Design Automation Conference (ASPDAC'05)* 151-6
[12] Akgul BS, Vmooney 2001 System-on-a-Chip Processor Support in Hardware, Proceedings of Design *Automation, and Test in Europe (DATE 01), IEEE CS Press, Los Alamitos, Calif* 633-9
[13] Fons F, Fons M, Canto E, Lo´pez M 2013 Real-time embedded systems powered by FPGA dynamic partial self-reconfiguration: a case study oriented to biometric recognition applications *J Real-Time Image Proc.* **2013**(8) 229-51
[14] Vyas S, Kumar C NG, Zambreno J, Gill C, Cytron R, Jones P 2014 *IEEE Embedded Systems Letters* **6**(1) 4-8
[15] Hajduk Z 2014 An FPGA embedded microcontroller *Microprocessors and Microsystems* **38** 1-8
[16] Hao K,Xie F 2009 Componentizing hardware/software interface design *Proceedings of Design, Automation, and Test in Europe (DATE 2009), IEEE Computer Society* 232-7
[17] Shahbazi M, Poure P, Saadate S, Zolghadri M R 2013 *IEEE Transactions On Industrial Electronics* **60**(8) 3360-71
[18] Wang Y 2011 The key technologies of embedded real-time operating system *PhD dissertation, University of Electronic Science and Technology, China*
[19] Iturbe X, Benkrid K, Hong C, Ebrahim A, Torrego R, Martinez I, Arslan T, Perez J 2013 *IEEE Transactions On Computers* **62**(8) 1542-56
[20] Belaid I, Ouni B, Muller F 2013 Complete and Approximate Methods for Off-line Placement of Hardware Tasks on Reconfigurable Devices *Journal of Circuits, Systems, and Computers* **22**(2) 1-30
[21] Jozwik K, Honda S, Edahiro M, Tomiyama H, Takada H 2013 Rainbow: An Operating System for Software-Hardware Multitasking on Dynamically Partially Reconfigurable FPGAs *International Journal of Reconfigurable Computing* **2013** 1-40
[22] Senoj J, Olakkenghil K Baskaran 2014 An FPGA Task Placement Algorithm Using Reflected Binary Gray Space Filling Curve *International Journal of Reconfigurable Computing* **2014** 1-7
[23] Iturbe X, Benkrid K, Hong C, Ebrahim A, Arslan T, Martinez I 2013 Runtime Scheduling, Allocation, and Execution of Real-Time Hardware Tasks onto Xilinx FPGAs Subject to Fault Occurrence *International Journal of Reconfigurable Computing* **2013** 1-32

Author

Shihai Zhu, China.

Current position: associate professor.
Scientific interest: embedded system, software and hardware co-design, new generation of computing system, the system chip SoC design method and IP reuse technology.