http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 10 (1): 158-163, 2011
ISSN 1812-5638

The Implementation of Semaphore Management in Hardware Real Time Operating System

Yan LI, Ping-Ping Gu and Xian-Shan Wang
The Computer and Control Institute, Harbin University of Science and Technology, China

Abstract: Currently, the study about hardware RTOS (Real Time Operating System) focus on implementation of the single module of RTOS by hardware, the overall hardware RTOS design and realization is seldom at home and abroad. So the hardware and software partitioning and implementing a hardware IP core of RTOS is worth further study. In this study, hardware-software partitioning of the hardware RTOS and the semaphore management are designed and implemented. The semaphore management is a program segment which runs frequently in operating system, the design solution that realize the semaphore management based on FPGA is put forward, in order to enhance the response capability of the RTOS. The storage structure, ECB (event control block) and mapping table are realized by on-chip registers; the semaphore management is realized by combinational logic circuit; both of them bring high speed to execute the semaphore system call function and P/V operations. The entire design is described by VHDL, simulated by the ISE 8.2 and realized on Xilinx's Virtex-II Pro Field Programmable Gate Array (FPGA) board. The result shows that realizing the semaphore management by hardware achieves good results in speeding up the RTOS.

Key words: RTOS, Semaphore, ECB, FPGA, hardware

INTRODUCTION

Operating system performance is not improving at the same rate as the speed of the execution hardware (Muneer and Rashid, 2006). As a RTOS are not keeping up with the demands placed on them. The RTOS is extensively used in the embedded system with the development of embedded technology, so there is an exigency to enhance the response capability of RTOS (Ramadass *et al*., 2007). However, improving data structure and scheduling algorithm of the RTOS which realized by software may improve the performance of system, but it is still unsatisfactory (Jianhua *et al*., 2008). At the same time, the word length of Microcomputer has reached 32, which is the upper limit size of the machine word (Yugui and Baohua, 1990).

A new approach implement the RTOS in hardware based on FPGA to improve the efficiency has been proposed by several institutes at home and abroad and some works have been done, in order to solve this problem (Yan *et al*., 2010). Currently, the software can be realized by hardware in two methods, one is micro-program mode with the characterizes of low cost, convenient and flexible; the other is realizing the software by combinational logical circuits, which characterizes are high speed and reliability, what's more important is the approach is gradually showing its superiority with the development of large scale integrated circuit (Yugui and Baohua, 1990). This thesis put forward the approach to realize the semaphore management in hardware in order to improve the efficiency of real-time system. The semaphore management and ECB management are designed reference to the RTOS μC/OS-II. The semaphore management system calls and P/V operations are realized by combinational logical circuits.

STRUCTURAL DESIGN OF HARDWARE RTOS

In this study, the system calls which used frequently in RTOS, such as task management, semaphore management, interrupt management and so on, are realized by hardware and others are realized by software. As it is shown in Fig. 1, there are two parts in the hardware RTOS, the software kernel and the hardware kernel. The software kernel can be subdivided into two parts, the interface transmit the information between application program and the hardware kernel; other parts of the software kernel implement other system calls which are not suitable to realize by hardware. These two parts cooperate with each other to form a complete Hardware RTOS.

In this system, a system call is completed by following steps. When a system call is invoked by the application program, the function code and the parameter of the system call are sent to the software kernel, the interface sends the function code and parameters to the hardware kernel. The function code and parameters are

Corresponding Author: Yan Li, Box 263, No. 52, Xuefu Road, Harbin, Heilongjiang Province, Postal Code 150080, China
Tel: 86+13664617642

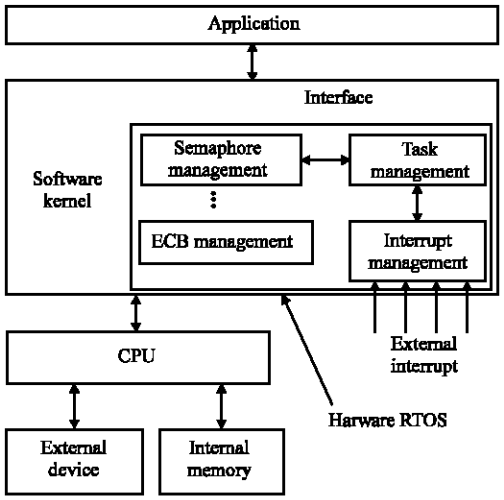

Fig. 1: The structure diagram of hardware RTOS

decoded and the hardware kernel executes the internal process of the system call and returns result to the interface, the software kernel returns result to the application program at the end (Nakano *et al*., 1997). When a task switching invoked by a system call or by external events, the hardware kernel sends an interrupt signal to the application program, saves the context of task in running state, then informs the task manager to executes a task scheduling, finds out the task with highest priority and load its context (Nakano *et al*., 1995).

THE HARDWARE DESIGN AND IMPLEMENTATION OF SEMAPHORE MANAGEMENT

The concept of semaphore management: The semaphore is actually a constraint mechanism, which is extensively used in multi-tasking kernel to manage the common pool resource (Hemalatha and Vivekanandan, 2008), mark the occurrence of an event and make the behaviors of two tasks synchronized. The semaphores work as key in RTOS, so a task can not goes on until it gets the semaphores needed (Labrosse, 2001).

The relationship among the tasks, interrupt service routines and the semaphores are shown in Fig. 2. In the figure, the key and the flag symbolize the semaphores and task can not goes on unless it obtains the key first, if the key have been occupied by other tasks, the current task has to be set in waiting state until the key is released.

Structural design of the hardware semaphore management: As it is shown in Fig. 3, the semaphore management module and ECB management are logical structures, both of them are independent of the CPU, can

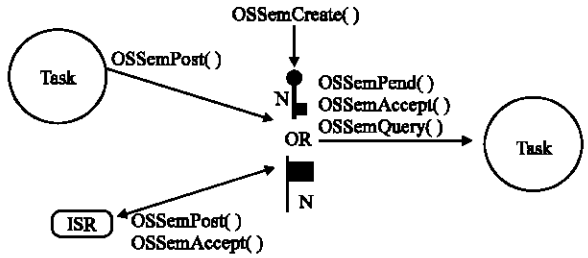

Fig. 2: The relationship of tasks, interrupt service routines and semaphores

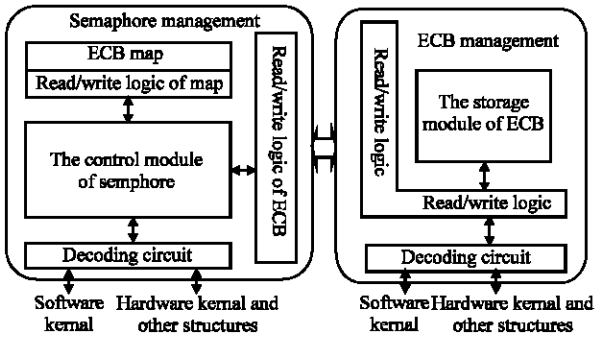

Fig. 3: Hardware structure of semaphore management

directly get information from the software kernel or other structures of the hardware kernel.

A data bus is established between the semaphore management module and ECB management module to speed up communication between them. The hardware RTOS works independent of the CPU and reduces the CPU's financial burden to improve the processing ability of system.

The ECB is the basic data structure to realize the semaphore management, message mailbox management and message queue management module in μC/OS-II. Therefore, the event control block management would be realized first.

The design and implementation of ECB: In the system, the ECB structure is designed reference to the structure of ECB in μC/OS-II. Each storage cell of ECB contains Event_Type unit, Task Waiting list unit, Cnt unit and so on. The Event_Type used to label the current ECB has been allocated by semaphore, mutex semaphore, mailbox or message queue; when an ECB is assigned to a semaphore, Cnt is the counter of semaphore; the Task Waiting list in ECB is used to store the tasks priorities which are waiting for this semaphore (Labrosse, 2001).

The structure diagram of the Task Waiting list implemented by hardware is shown in Fig. 4.

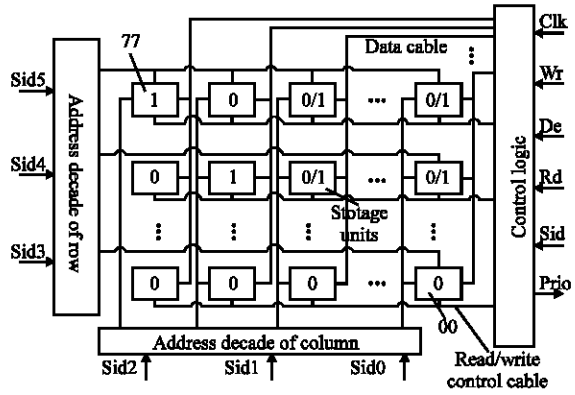

Fig. 4: Schematic figure of task waiting list

The structure of Task Waiting list is similar to a matrix with 8 rows and 8 columns, the storage units numbers are from 00 to 77. When a running task applies for a semaphore and does not get it, the task is set in waiting state. Make signal Wr effectively, use the priority of the task as address and decode the address to select the corresponding unit and write 1. When task A releases a semaphore and task B gets it, then task B should be deleted from Task Waiting list, make the signal Del effectively, use the priority of task B as address and decode the address, select the corresponding unit in Task Waiting list and write 0.

Other basic units of the ECB memory cell in hardware system are achieved by calling the IP core supplied by FPGA board; The ECB storage body are made up by ECB cells which are combined by basic memory units. An ECB cell can be selected by decoding the address, reading or writing operations is finished follow the control signals.

Design the system call function of semaphore management in hardware: The ECB is a public data structure, an ECB is needed firstly when a semaphore created in the traditional operating system, the P/V or other operations can not be done unless a semaphore has been initialized; When delete a semaphore, the ECB cell has been used by the semaphore should be released. Both operations above required many steps to complete. The first step is to look up the ECB storage body when a semaphore is created, to find if there are any spare ECB cells; if there is no spare ECB cell, the error message is returned; otherwise the ECB cell address is generated and returned to the task which applies to create the semaphore, the last step is initials the ECB cell. The whole process above will waste a lot of clocks in sending data back and forth among modules, which makes the system speed-down; A mapping table is set to record the usage of ECB in the semaphore management module to solve

	1st column	2nd column	3rd column	...	8th column
1st row	0	0	0	...	1
2nd row	0	1	0	...	0
⋮	⋮	⋮	⋮	⋮	⋮
8th row	0	0	0	...	0

Fig. 5: Schematic figure of mapping table

this problem, each position in the table corresponds to an ECB cell, when a position is 0 /1 means that the ECB cell in idle position is spare/occupied.

As it is shown in Fig. 5, value in the unit at 1st row, 8th column is 1, it means that ECB cell with offset address 000 111 is occupied; value of the unit at 2nd row, 2nd column is 1, it means that the ECB cell with the offset address 001,001 is occupied.

When a semaphore is deleted, the first step the Control Module is to look up the mapping table, to read out the value in corresponding position and judge whether it is 0, if it is 0, it means that the semaphore has been removed by other tasks, delete error signal is returned by the Control Module to the task which tries to remove the semaphore. Otherwise the Control Module clears the record of the semaphore in the mapping table, informs the ECB management module to set all tasks in ready state, which tasks are in Task Waiting list of this semaphore; the ECB management module clears the semaphore information in ECB cell and release the cell; the Task Manager triggers a task scheduling.

The hardware implement of P/V operations: The mainly part of the Semaphore management are P/V operations, the hardware implementation of P/V operations are shown in Fig. 6.

P operation, when a task applies for a semaphore, the input signal pend_sem is in high level, the Control Module judges whether the task is an interrupt service routine (in μC/OS-II, the interrupt service routine does not allow to apply for a semaphore), if it is return back error signal (pend_err in high level), does not carry out the following operations; Otherwise the Control Module enables the read_cnt signal, reads the value of Cnt from ECB management modules, judges Cnt value when it is read back, if the Cnt>0, the current task gets the semaphore and continue to run, the Control Module returns a successful signal (the signal pend_err in low level); if the Cnt ≤0, the Control Module sends the signal pend_err in high impedance, then according to application type the signal Pend_type (there are two kinds of application types in μC/OS-II , no wait application ,with wait application) to decides whether to modify the Cnt

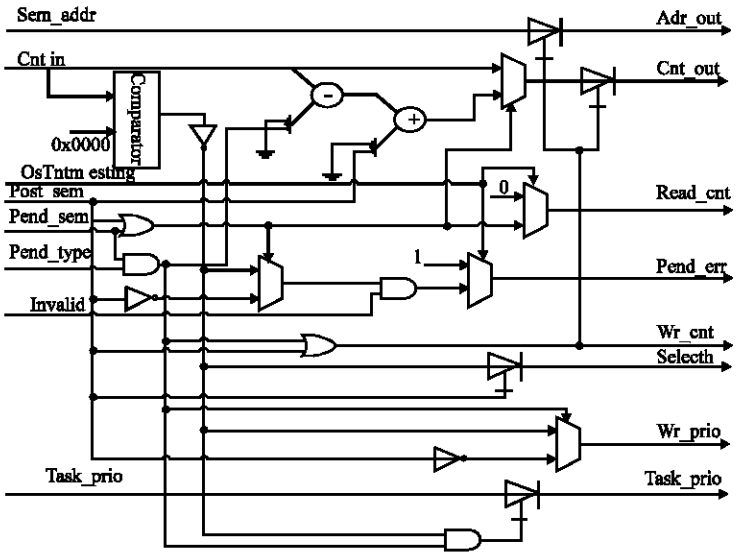

Fig. 6: Hardware implement of P/V operations

value, to set the task apply for the semaphore in waiting state.

V operation, release a semaphore, the Control Module gets the signal post_sem in high level, then the Control Module enables the signal read_cnt , sends out the ECB semaphore address at the same time, reads back and judges the value of Cnt at the next clock; if Cnt> =0, it means that there is no task waiting for the current semaphore, modifies the Cnt value; else if Cnt <0 ,it means that there are tasks waiting for the current semaphore, the Control Module modifies the Cnt value, enables the signal select_h; the ECB management module gets the signal select_h in high level, first it finds out the highest priority task from the task waiting list in the ECB and sends to the Task Manager, informs the Task Manager sets the task in ready state, then the Task Manager triggers a task scheduling.

THE RESULT OF SIMULATE AND TEST

In order to verify the accuracy and highly effective of the hardware realization, the entire design is described by the hardware language of VHDL and the ISE 8.2 software is used to carry on the succession simulation confirmation. The simulation results of the P / V operation is shown in Fig. 7.

As it is shown in the Fig. 7, the signal pend_sem or post_sem is still effective during the P or V operations.

- The pend_sem signal is in high level (P operation), the task with priority 0x01 applies for the semaphore with address 0x05. The Control Module reads semaphore value Cnt according to the address has given, the value returned from the ECB management module is 0x0002>0x0000; The task gets the semaphore and continues to run; the Control Module minus one Cnt value and writes it back to the ECB
- The post_sem signal in high level (V operation), the Control Module reads the Cnt value according to the address has given, the Cnt value is 0xFFFE <=0x0000 (the Cnt value store in the form of complement). At next clock the Control Module plus one Cnt value and write it back to ECB, enables the signal Select_h; the ECB management module selects the task with the highest priority in the task waiting list, sends to the Task Manager and informs it sets the task in ready state and triggers a task scheduling
- Apply for a semaphore, the task with priority 0x03 applies for the semaphore with the address 0x09. the Control Module reads back the Cnt value 0xFFFD <=0x0000 at next clock, the application type is in high level (it is with wait application), so modifies the Cnt value, writes the priority of current task to waiting list, notifies the Task Manager sets the task in waiting state and triggers a task scheduling
- Apply for a semaphore, the Control Module reads back the Cnt value 0xFFFA <=0x0000, but the current application type is in low level (no wait application), the Control Module does not carry out any operation but returns failure signal.

The simulation results of creating and deleting a semaphore are shown in the Fig. 8, the signal sem_create (del_sem) and data are maintain two system clocks when

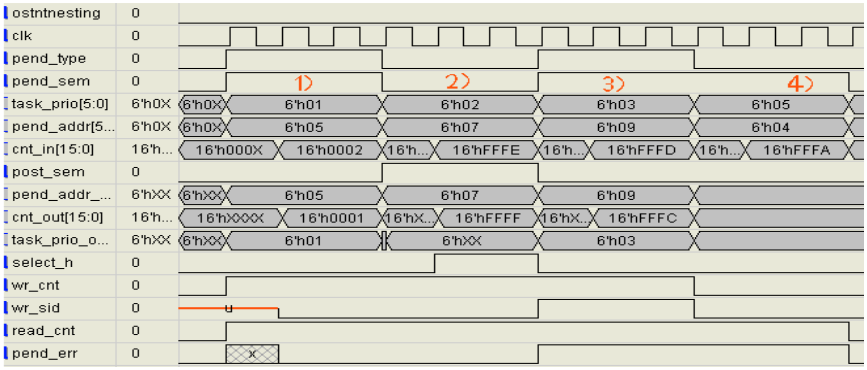

Fig. 7: Apply for or release a semaphore

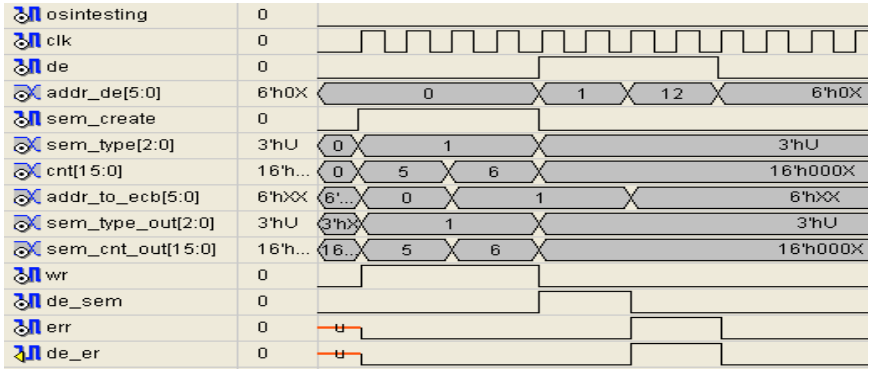

Fig. 8: Create/delete a semaphore

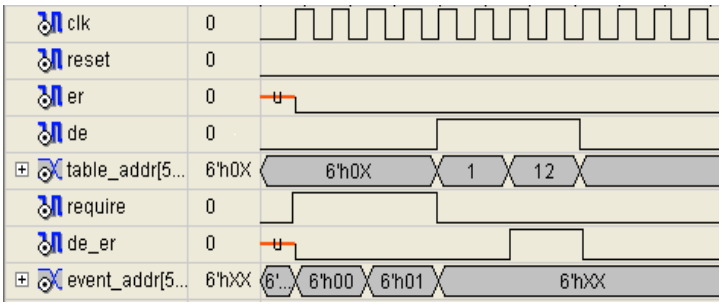

Fig. 9: The changes in the mapping table when create or delete a semaphore

a semaphore is creating (deleting), it is to make sure that data can be send to the ECB cell errorless.

In the Fig. 8, two semaphores with the initial values 0x05 and 0x06 are created from the beginning of first clock to the end of forth clock; the Control Module gets the sem_create signal in high level, it looks up the mapping table to gets address of an ECB unoccupied, the data changes in the mapping table are shown in the Fig. 9, the Control Module gets the ECB cell address and marks it; the Control Module sends the address and the value of the semaphore to the ECB management module at next clock.

At the fifth and sixth clocks, the Control Module gets the de signal in high level and gets the semaphore's address 0x01, the Control Module searches mapping table, makes sure the existence of the semaphore, then removes the semaphore, clears the record of semaphore in the mapping table and sends a signal to inform the ECB

Table 1: Resources required of hardware semaphore realization in XC2VP30FF896C

Logic utilization	Used	Available	Utilization (%)
No. of slices	1443	13696	10
No. of slice Flip Flops	1701	27392	6
No. of 4 input LUTs	2202	27392	8
No. of bonded IOBs	129	556	23
No. of BRAMs	2	136	1
No. of GCLKs8	16	50	

management module to set all the task which are waiting for the semaphore in ready state.

At the seventh clock, the semaphore with address 0x12 is being removed, the Control Module searches the mapping table, finds out that the semaphore is not exist and then it returns the signal remove error.

Table 1 shows the utilization of the resources to achieve the semaphore management modules which based on the component XC2VP30 of Xilinx Virtex II pro.

DISCUSSION

This study provides hardware-software partitioning of the hardware RTOS and the implementation semaphore management. In the traditional RTOS, user's application applies for a semaphore, the P/V operation will be executed as follow, at first the execute state would transform from user state to system state, the semaphore value is read out, calculated, compared, judged and then when the appropriate program is finished the execute state return to user state from the system state; However, in the hardware RTOS, one read or write command is needed which instruction is necessary in the software and other operations are implemented by hardware logic to simplify the operational process.

The simulation results in Fig. 7 indicate that three internal clocks of FPGA is needed to execute the P/V operation implemented by the hardware and the simulation results in Fig. 8 indicate that two internal clocks of FPGA is needed to execute the operation, creating or deleting a semaphore, implemented by the hardware. All of these operations needed much less steps and little time than the traditional operation implemented by the software. The Storage structures of semaphore management are realized by IP core of FPGA and resources required of hardware semaphore realization in XC2VP30FF896C is shown in Table 1.

The high speed and efficient hardware semaphore management reduces the time to execute a system call and increases the overall of RTOS. There are numbers of semaphores and the semaphore management runs frequently in common RTOS, so it speeds up the system a lot when realizes the semaphore management by hardware, particularly in the system with resources varieties and quantities. On the other hand, the reliability of the operating system is improved because of the reliability of the hardware far more than the software.

ACKNOWLEDGMENT

This study was supported by Natural Science Foundation of Heilongjiang Province of China (No. F200805)and the national innovation experiment program for university students (No. 081021413).

REFERENCES

Hemalatha, M. and K. Vivekanandan, 2008. A semaphore based multiprocessing k-mean algorithm for massive biological data. Asian J. Scientific Res., 1: 444-450.

Jianhua, C., S. Hongsheng and W. Baojin, 2008. The design and realization of hardware real-time operating system. Appl. Comput. Technol., 5: 34-37.

Labrosse, J.J., 2001. MicroC/OS-?The Real-Time Kernel. 2nd Edn., CMP Books, Gilroy, USA., pp: 178-185.

Muneer, H. and K. Rashid, 2006. SPE architecture for concurrent execution OS kernel and user code. Inform. Technol. J., 5: 192-197.

Nakano, T., U. Andy, M. Itabashi, A. Shiomi and M. Imai, 1995. Hardware implementation of a real-time operating system. Proceedings of the 12th TRON Project International Symposium, Nov. 28-Dec. 2, Tokyo, Japan, pp: 34-42.

Nakano, T., Y. Komatsudaira, A. Shiomi and M. Imai, 1997. VLSI implementation of a real-time operating system. Proceedings of the ASP-DAC '97 Asia and South Pacific Conference on Design Automation, Jan. 28-31, Chiba, pp: 679-680.

Ramadass, N., S. Natarajan and J.R.P. Perinbam, 2007. Dynamically reconfigurable (Self-modifiable) architecture for embedded system-on-chip applications. Inform. Technol. J., 6: 66-73.

Yan, L., L. Xian-Yao, G. Ping-Ping, Z. Hong-Jie and C. Ping, 2010. Hardware implementation of μC/OS-II based on FPGA. Proceedings of 2nd International Workshop on Education Technology and Computer Science, March 6-7, Wuhan, Hubei, China, pp: 852-858.

Yugui, Q. and Z. Baohua, 1990. Firmware of semaphore management in operating system. Comput. Applied Software, 6: 29-33.