A NOVEL IMPLEMENTATION OF HARDWARE BASED HYBRID EMBEDDED RTOS

Qiang Huang[1],Yongbin Bai[2], QiRui Huang[3] and XiaoMeng Zhou[4]

College of Computer Science and Software Engineering,
ShenzhenUniversity, Shenzhen,518060
jameshq@szu.edu.cn

ABSTRACT

Reliable embedded systems play an increasing role in modern life, especially in modern automotive designs. Many studies have proved that it performs better in many situations. Firstly, reliable embedded systems provide the system reliability improvements. Secondly, reliable embedded systems also can improve the development efficiency and make the development cycle shorter.

However, in the high real-time required occasion, the software implementation of the RTOS can`t fully meet requirements. To have better real-time only through the algorithm improvement or just increase the processor speed. On the contrary, operating system based on a hardware implementation can make it more real-time and more reliable. The reason is due to that the hardware circuit is independent of the processor running and do not take up the processing time of the processor. Thereby it can save time to execute other tasks and improve real-time. In this paper, ARM+FPGA will be choose as the IP hardware development platform.

KEYWORDS

Time-triggered/event-triggered, jitter, hardware schedule.

1. INTRODUCTION

Since the 1980s, some international IT organizations have started to research the commercial embedded real-time operating system and specialized real-time operating system. Form that on, there appears many real-time operating systems, like VxWorks, LynxOS, embedded Linux, TRON and uC / OS-II.

In the 80s of the last century in the US, Jaehwan Lee and Vincent John MooenyIII[1] [2]compared the RTOS scheduler implementation from hardware to software, and make a RTOS scheduler accomplished by specialized hardware IP core which will greatly improve the work efficiency of the RTOS.

In Brazil, Mellissa Vetromilleand Luciano OST[3]compare and analyze the RTOS scheduler accomplished by the software and hardware, the results was that hardware scheduler has higher reliability.

In Japan, Professor Takumi Nakano[4][5]developed a silicon wafer named STRON-I（Silicon OS）in 1990s. It used VLSI to hardened the operating system TROS to a chip. Therefore, the operating system chips can work in parallel with the microprocessor, which can further ensure the real-time and high reliability of real-time operating systems.

The TTE32-HR2[6]microprocessor made by TTE Systems, which used the cooperative scheduling hardware implementation as the TTE32 kernel peripherals and use it to achieve the task scheduling.

Summary: We can find that the research is mainly concentrated in the local module of the hardening operating system, while little research literature based on the overall hardware design and implementation of real-time operating system. Therefore, we should have deeper research in how to carry out the optimal software partitioning for real-time operating system and accomplish a hardware real-time operating system.

Nowadays, in our country, real-time operating system based software mainly has two different types: one of that is China's independent research and development of real-time operating system, for example: the open source RT-Thread, Delta OS, Hopen OS, CASSPDA developed by Chinese Academy of Sciences, Beijing Software Engineering Centerand HBOS of Zhejiang university. Another one is completed by secondary development that based on foreign operating system. This kind of operating system is the exclusive use of the system such as the Chinese Academy of Sciences of the red flag Linux and Shenzhen blue Linux.

At present, the domestic research in literature hardware real-time operating system is nearly zero until some articles have been published recent years. For example, HouMi, from Shanghai Jiaotong University, proposes and designs a real-time task management device based on hardware. WangChuanfu and Zhou Xuehai, from the University of Science and Technology of China's, put forward a method to improve the performance of hardware multithread processor. Zhejiang University professor Chen Tianzhou[7] proposed a CPU FPGA hybrid architecture hardware thread execution mechanism method. Cui Jianhua, Sun Hongsheng and Wang Baojin, from PLA Information Engineering College design a simple hardware real-time operating system and realize the task scheduling, interrupt management and basic function of timer management RTOS with a FPGA development board.

Summary: The present study has focused on the hardware task scheduling and hardware interrupts processing. However, the communication and synchronization between tasks, memory management and implementation in hardware context switch are still a problem to be solved and research.

Our design is a hardware real-time operating system IP kernel which including task scheduling, interrupt processing, communication and synchronization between tasks, and time management. The kernel development use ARM+FPGA as the IP hardware development platform.

2. RELATED WORK

2.1 Hardware platform

There are two ways we can choose to realize hardware platform.

First：ARM+FPGA, the following picture is an overview which we realize on FPGA.

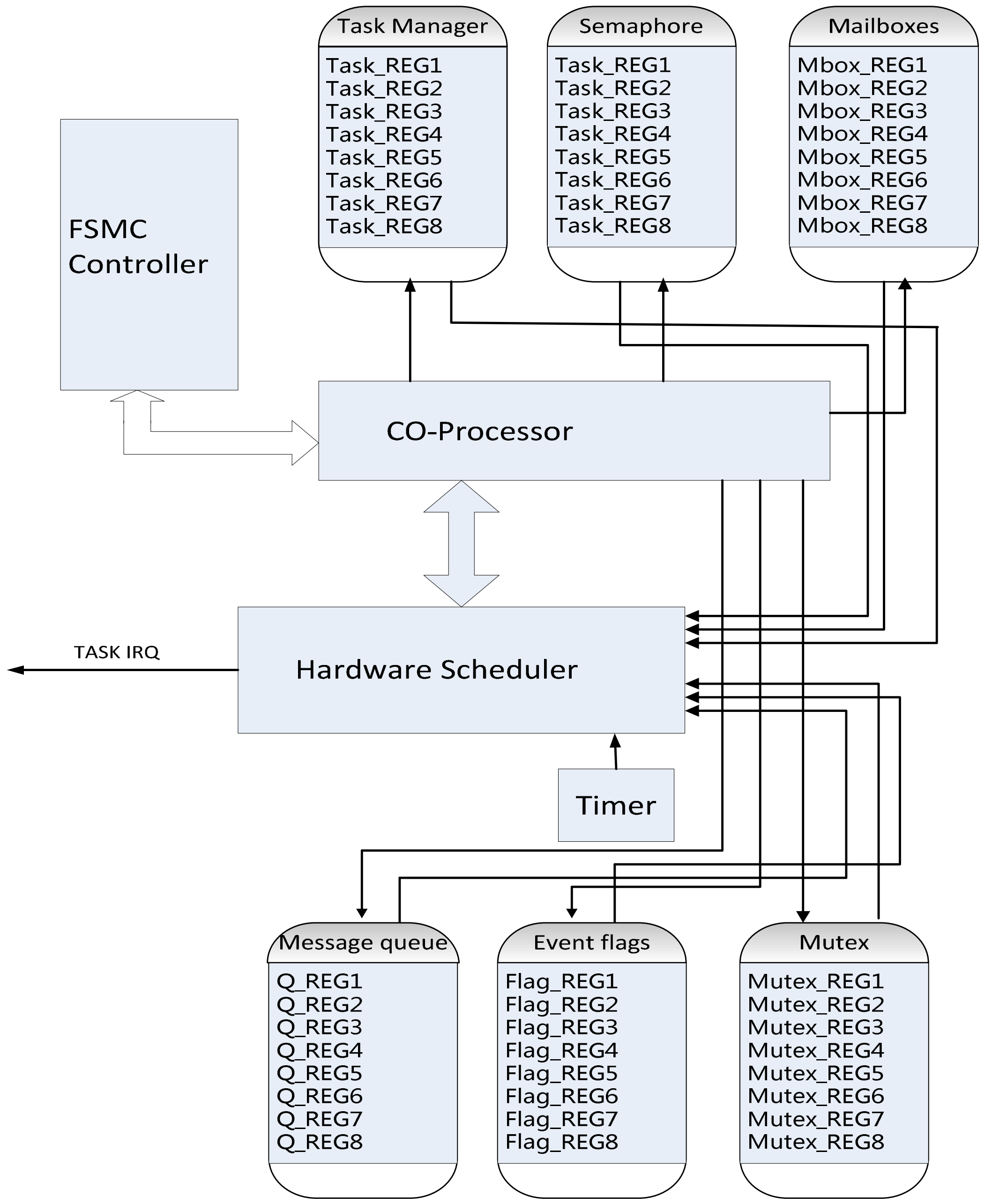

Figure 1. ARM+FPGA

Second：Only using FPGA, just like the following picture. This picture comes from TTE Systems

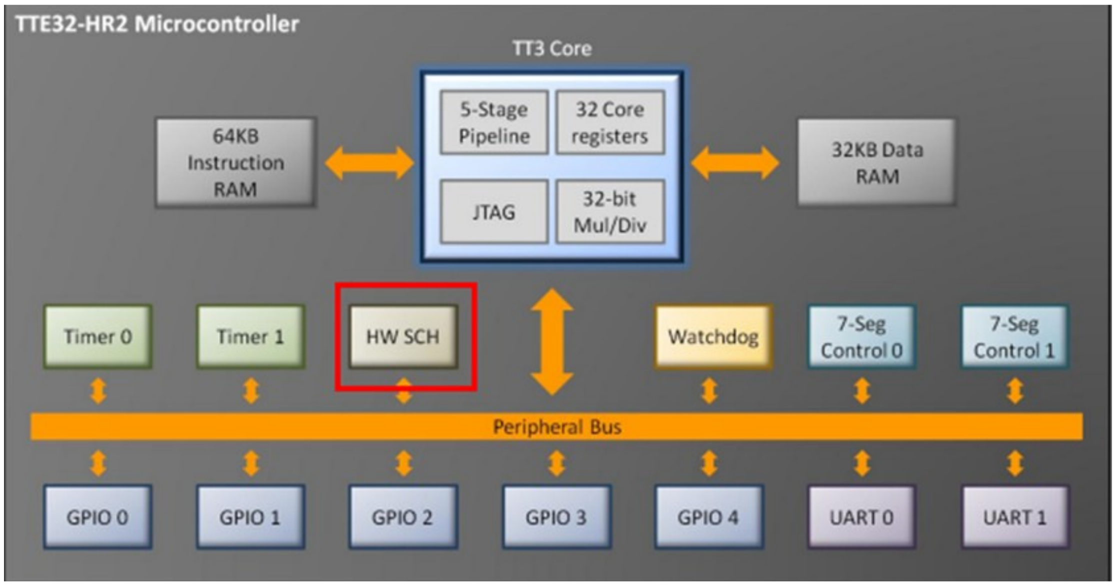

Figure 2. FPGA

Considering that the second method is inconvenient to debug. So we choose the first method as hardware platform.

In order to better test the hardware real-time system. We have made a total of 10 sets of development board.

Features of the platform include:

（1） MCU uses ST Company's STM32F103VET6
（2） FPGA uses Altera Company's EP4CE6E22C8
（3） Supply voltage acquisition circuit
（4） Supply voltage acquisition circuit
（5） Ethernet module
（6） Two Serial port modules
（7） ZigBee module
（8） CAN interface
（9） Segment module
（10）4.3-inch LCD module
（11）Analog signal acquisition circuit
（12） Buttons and LEDs

2.2 Hardware RTOS features

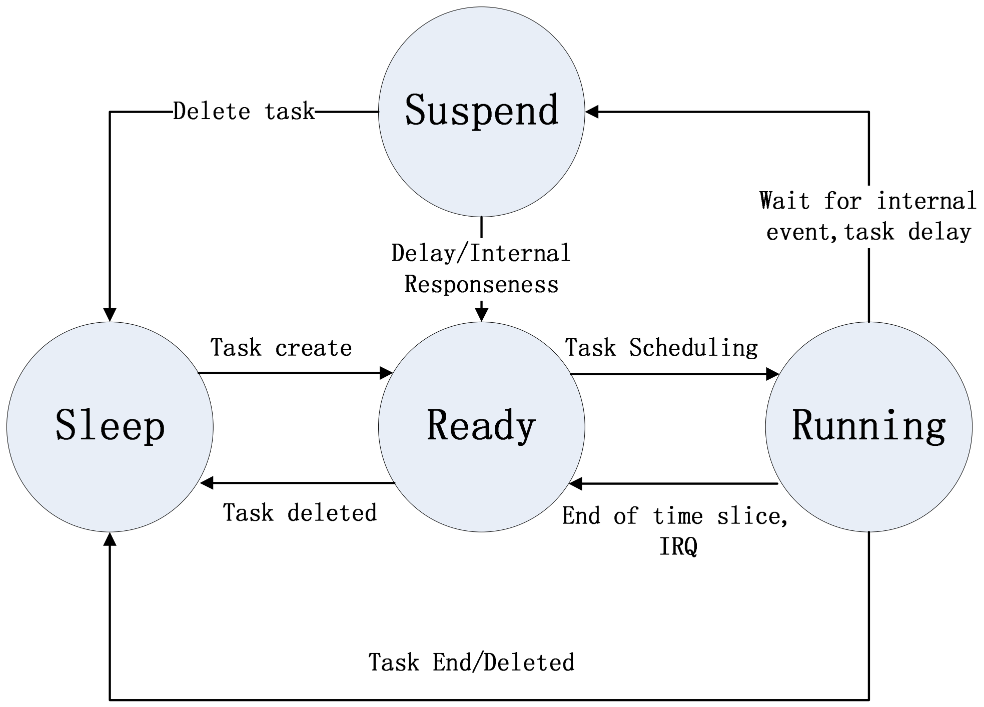

Figure3. State Switching

The Hardware RTOS mainly achieved functions as follows: preemptive scheduling, task management, semaphores, message mailboxes, message queues, mutexes and event flags group.

We have completed all the necessary components for small real-time embedded systems.

- Support the creation of 8 tasks
- Support the creation of 8 semaphores
- Support the creation of 8 message mailboxes
- Support the creation of 8 message queues
- Support the creation of 8 mutexes
- Support the creation of 8 event flags groups

The FPGA can easily extended to support more components and tasks.

2.3 The communication between ARM and FPGA

By using FSMC interface of STM32, we can realize the communication between ARM and FPGA. In order to make FPGA as one part of ARM kernel peripherals, we use Bus Interface instead of SPI or UART.

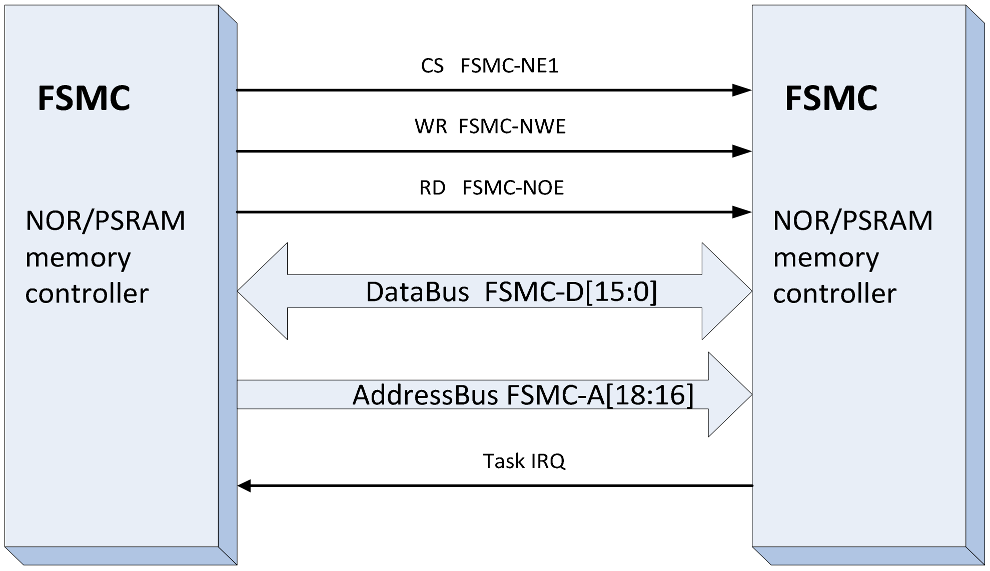

Figure4. ARM and FPGA interface

2.3.1 Writing data to FPGA

From the figure above, there are 16 data wires, 3 address wires. ARM can only access to eight 16-bit data on FPGA. In order to access to more data, We use a method similar to serial. By distinguishing different ID, we realize the access to more data, Each ID can access to eight 16-bit data on FPGA. For example. Writing data to FPGA is mainly used to initialize register.

When register HW_ID = 0, ARM can access to eight 16-bit data on FPGA.

```
HW_ID = 0;
HW_DELAY = 111;
HW_PERIOD = 0;
HW_GPT = 144;
HW_AOT = 155;
HW_BACKUP = 166;
HW_OVERRUN = 177;
HW_CONTROL = 1;
```

When register HW_ID = 1, ARM can access to eight 16-bit data on FPGA.

```
HW_ID = 1;
HW_DELAY = 211;
HW_PERIOD = 0xffff;
HW_GPT = 244;
HW_AOT = 255;
HW_BACKUP = 266;
HW_OVERRUN = 277;
HW_CONTROL = 1;
```

2.3.2 Reading data from FPGA

Reading data from FPGA is similar to writing data to FPGA.
Each ID can read eight 16-bit data from FPGA. Reading data is mainly used to read the current highest priority task which needs to execute from the FPGA.

2.3.3 Scheduler Tick Interrupt

Scheduler Tick Interrupt is generated every millisecond by FPGA.
After ARM receive external interrupt, the interrupt service routine read the current highest priority task which needs to execute from the FPGA.

3. THE TASK WE COMPLETED

We have mainly achieved functions as follows: preemptive scheduling, task management, semaphores, message mailboxes, message queues, mutexes and event flags group. We have completed all the necessary components for small real-time embedded systems.

Support the creation of 8 tasks
Support the creation of 8 semaphores
Support the creation of 8 message mailboxes
Support the creation of 8 message queues
Support the creation of 8 mutexes
Support the creation of 8 event flags groups

The following specific describes the various parts of the implementation.

3.1 Preemptive scheduler

There are three key points to realize preemptive scheduler:

When a task signals or sends a message to a higher-priority task, the current task suspended and the higher-priority task is given control of the CPU.

When each tick interrupt comes, if there is a high priority task is ready, high priority task will preemptive low priority task. When a task signals or sends a message to a higher-priority task, the massage has been sent, the interrupted task remains suspend and the newer higher priority task resumes.

3.1.1 FPGA

A core job to realize preemptive scheduler is to figure out how to find the highest priority task inside the task ready list. We use the priority encoder to realize it. The method is as follows:

```
function[15:0] code;
        input[7:0] din;
        case x (din)
                8'b1000_0000 : code = 16'h7;
                8'bx100_0000 : code = 16'h6;
                8'bxx10_0000 : code = 16'h5;
                8'bxxx1_0000 : code = 16'h4;
                8'bxxxx_1000 : code = 16'h3;
                8'bxxxx_x100 : code = 16'h2;
                8'bxxxx_xx10 : code = 16'h1;
                8'bxxxx_xxx1 : code = 16'h0;
                default: code = 16'h7;
        end case
end function
```

We must pay special attention to a point that the idle task is always ready. Idle task has the lowest priority, when there is no task running, idle task will be executed.

3.1.2 ARM

For this hardware real-time systems, we just need pay attention to two points:
Task-level task switching, which is mainly to realize a high priority task switch to a low-priority task.
Interrupt-level task switching, to determine whether there is a higher priority task is ready when interrupt quit and switch to the high priority task.

3.2 Task management

The task management mainly to achieve three functions: Delay time setting, Suspend the task, Task resume.

Each task has 8 configurable registers
Task_REG2 Delay time setting
        = 0 add task to ready list
        = 0xffff delete task for ready list
        = others the task delay time to be set
Task_REG3 Task ID
0-7 8 task ID, read the register can get the current highest priority ready task
Task_REG8 initialization task execution
        = 1 task can be executed
        = others task can not be executed

3.2.1 FPGA

The task will be start when Task_REG8 = 1, every single task running on the FPGA is a separate process but not put them all in one process. This can make full use of hardware real-time system.

```
always @ （posedge clk）
begin
task1 ;
task2 ;
End
```

```
always @（posedge  clk）begin
task1 ;
end
always @（posedge  clk）begin
task1 ;
end
```

3.2.2 ARM

ARM just need simply set the register to configure all tasks.

- Setting the task delay time

```
Task_REG3 = 0;    //set task 0
Task_REG3 = 100;  //set task delay time
```

- Task suspend, delete task form ready list.

```
Task_REG3 = 0;    //set task 0
Task_REG3 = 0xffff;  //to suspend task
```

- Task recovery, add the task to ready list

```
Task_REG3 = 0;    //set task 0
Task_REG3 = 0;   //to recovery task
```

3.3 Semaphore

Semaphore is to establish a flag for shared resources. The flag indicates that the shared resources occupancy. Hardware RTOS can support to create 8 semaphores, each semaphore has 8 registers. Register Description:

Task_REG3 semaphore ID
8 - 15 represents the semaphore ID can be created
Task_REG4 wait semaphore events list
write to this register, add task to this semaphore's wait list.
read from this register， find the highest priority task form wait list.
Task_REG5 Semaphore count, indicates the number of available resources

3.3.1 FPGA

In the implementation of the semaphore, Hardware RTOS not only provide the required scheduler function but also can find out highest priority task in semaphore wait list.

3.3.2 ARM

Mainly provides the following three functions, which is used for the semaphore register initialization and implementation.

- void OSSemCreate(uint16_t ucSemID, uint16_t uiSemCnt);

This function is used to initialize the semaphore
When uiSemCnt = 0 can use semaphore for task synchronization
When uiSemCnt> 0 indicates the number of available resources

- void OSSemPend(uint16_t ucSemID, uint16_t ucSemTime, uint16_t ucPendTaskID);

This function is used to request the semaphore
When ucSemTime = 0xffff indicates the task suspend until there are available resources.
When ucSemTime>0 indicates the task suspend times

- void OSSemPost(uint16_t ucSemID);

This function is used to release the semaphore

3.4 Message mailboxes

Message mailbox is mainly used for the transmission of messages between the two tasks. Hardware RTOS support to create 8 message mailboxes, each message mailbox have 8registers, Registers are described below:

Mbox _REG3 message mailbox ID
    16-23 indicates the semaphore ID can be created.
Mbox _REG7 wait message mailbox events list
write to this register, add task to this message mailbox's wait list.
read from this register, find the highest priority task form wait list.

3.4.1 FPGA

In the implementation of the message mail box, Hardware RTOS not only provide the required scheduler function but also can find out highest priority task in message mailbox wait list.

3.4.2 ARM

Mainly provides the following three functions, which is use for the message mailbox register initialization and Implementation.

- void OSMboxCreate(uint16_t ucMboxID);This function is used to initialize the semaphore
- Used to create the message mailboxes
- void *OSMboxPend(uint16_t uiMboxID, uint16_t uiMboxTime, uint16_t uiPendTaskID);

This function is used to request message mailbox
When uiMboxTime= 0xffff Indicates the task suspenduntil there are available resources.
When uiMboxTime>0 Indicates the task suspend times

- OSMboxPost(uint16_t uiMboxID, void *Pmsg);

This function is used to send a message

3.5 Message queue

The realization method of the message queue is similar to the message mailbox, but it is necessary to do a circular queue for the message queue used for message's FIFO or LIFO. Hardware RTOS support to create 8 message queues, each message queue have 8 registers. Registers are described below：

Q_REG3 messagequeueID
24-31 Indicates the message queue ID can be create.
Q_REG6 Wait message queue events list
write to this register, add task to this message queue's wait list.
read from this register, find the highest priority task form wait list.

3.5.1 FPGA

In the implementation of the message queue, Hardware RTOS not only provide the required scheduler function but also can find out highest priority task message in queue wait list.

3.5.2 ARM

Mainly provides the following three functions, which use for the message queue register initialization and Implementation.

- void OSQCreate (void **start, uint16_t uiSize, uint16_t uiQueueID);

Used to create the message queue.

- void *OSQPend(uint16_t uiQID, uint16_t uiQTime, uint16_t uiPendTaskID);

This function is used to request message queue
When uiQTime= 0xffff Indicates the task suspenduntil there are available messages.
When uiQTime>0 Indicates the task suspend times

- uint8_t OSQPost(uint16_t uiQID, void *Pmsg);

This function is used to send a message.

3.6 Event flag group

In the real applications practical, The task often need to determine the operation mode of the task according to the result of the amount of a composition of a plurality of semaphore. So we provide event flag group for this. Hardware RTOS support to create 8 event flag group, each event flag group have 8 registers, Registers are described below:

Flag_REG1 wait event flag group' list
write to this register, add task to this event flag group's wait list.
read from this register, find the highest priority task form wait list.
Flag _REG3 event flag group ID
32-39 Indicates the event flag group can be created.

3.6.1 FPGA

In the implementation of the event flag group, Hardware RTOS not only provide the required scheduler function but also can find out highest priority task in event flag group wait list.

3.6.2 ARM

Mainly provides the following three functions, which is used for the event flag group register initialization and Implementation.

- void OSFlagCreate(uint16_t ucFlagID);

- Used to create the event flag group.

- void OSFlagPend(uint16_t uiFlagID, uint16_t uiFlagTime, uint16_t uiPendTaskID, uint16_t uiFlag);

This function is used to request event flag group.
uiFlagID Indicates the flag need to be get .
whenuiFlagTime= 0xffff Indicates the task suspend until there are available resources.
When uiFlagTime>0 Indicates the task suspend times

- void OSFlagPost(uint16_t uiFlagID, uint16_t uiFlag);

This function is used to send event flag group
uiFlagID Indicates event flag group which is need to sent.

3.7 Mutual Exclusion Semaphore

Binary semaphore is so easy to cause priority inversion, so we use mutual exclusion semaphore to achieve exclusive use of shared resources. Hardware RTOS support to create 8 mutual exclusion semaphore, each mutual exclusion semaphore have 8 registers, Registers are described below:

Mutex _REG3 mutex ID
40-47 Indicates the mutex ID can be created
Mutex _REG8 Wait mutex events list
write to this register, add task to this mutex's wait list.
read from this register, find the highest priority task form wait list.

3.7.1 FPGA

In the implementation of the mutual exclusion semaphore, Hardware RTOS not only provide the required scheduler function but also can find out highest priority task in mutex wait list.

3.7.2 ARM

Mainly provides the following three functions, which is used for the mutex register initialization and Implementation.

- void OSMutexCreate(uint16_tuiMutexID, uint8_t uNewPrioty);

This function is used to initialize the mutex

- void OSMutexPend(uint16_tuiMutexID,uint16_tuiMutexTime, uint16_t ucPendTaskID);

This function is used to request the mutex

When uiMutexTime= 0xffff Indicates the task suspenduntil there are available resources.
When uiMutexTime>0 Indicates the task suspend times

- void OSMutexPost(uint16_t uiMutexID);

This function is used to release the mutex

4. Distributed Deployment

Many modern embedded systems contain more than one processor. For example, a modern passenger car might contain some forty such devices, controlling brakes, door windows and mirrors, steering, airbags, and so forth. Similarly, an industrial fire detection system might typically have200 or more processors, associated - for example - with a range of different sensors and actuators. Two main reasons:

· Additional CPU performance and hardware facilities
· Benefits of modular design

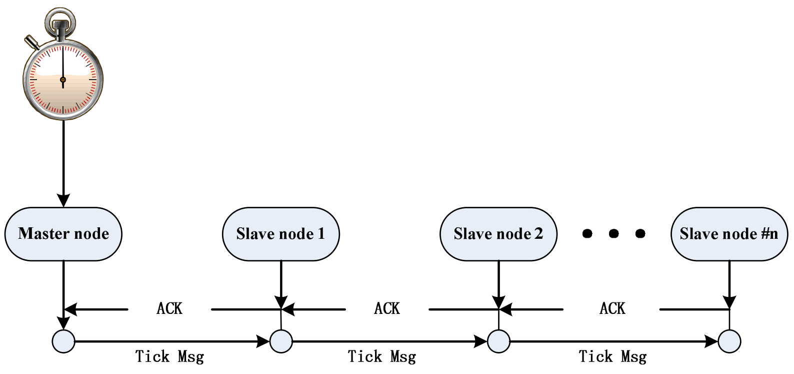

Figure5. S-C scheduler

By using a shared-clock (S-C) scheduler, we can link more than one processor. There are many ways to realize shared-clock scheduler. For examples, using external interrupts, using UART, can bus and so on. Here we will use ZigBee wireless to realize shared-clock scheduler.

5. Conclusions

The real-time operating system shows more real-time and reliability that based on the hardware implementation. Because the hardware implementation is running independent of the processor running, it does not consume the processing time and processor saves time to execute tasks, so that task scheduling and real-time is improved.

ACKNOWLEDGMENTS

This work was supported by Science & Technology Planning Project of Shenzhen City Grant No. JCYJ20120613112757342.

REFERENCES

[1] V.MOONEY III, J.LEE, A.DALEBY, et.al. A comparison of RTU hardware RTOS with a hardware/software RTOS[C]. Design Automation Conferece(ASP_DAC '03), 2003:683-688 .

[2] V.MOONEY 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

[3] MELISSA VETROMILLE , LUCIANO OST, CESAR A.M.MARCON, et.al RTOS Scheduler Implemention in Hardare and Software for Real time Application [C]. proceedings of the senventeenth IEEE International workshop on rapid system protoryping(RSP 06). 2006:163-168

[4] T.NAKANO, U.ANDY, M.ITABASHI, et.al. Hardware Implemention of a Real-time Operating System[J]. Proceedings of the Twelfth TRON Project International Symposium IEEE Computer Society Press, Nov,1995:34-32.

[5] T.NAKANO, U.ANDY , M.ITABSSHI, et.al. VLSI Implementation of a Real-time Operating System[J]. Proceedings of the ASP-DAC '97 Asia and South Pacific, January 1997:679-680

[6] TTE32-HR2 evaluation microcontroller programming guide. Datasheet and Programming Guide TTE32-HR2 Microcontroller (r1.2): March 2011. This document is copyright © TTE Systems Limited 2007-2011.

[7] CHEN TIANSHOU, WU XINGLIANG, HU WEI. Research on OS-AwareEmbedded Power-Saving Architectre[C]. The 2rd Joint Conference on Harmonious Human Machine Environment, HHME2006,PCC'06: 52-59

[8] ADOMAT J, FURUNAS J, INDH L, etal. Real – time Kernal Hardware RTU: A step torwards deterministic and high performance real-time systems[J]. Proceedings of eighth Euromicro Workshop on Real-time Sysrems, 1996:683-688.

AUTHOR

Qiang Huang

Professor of ShenZhen University, P.R. China. born on 1977. Graduated from the University of Liverpool in Electrical Engineering with Ph.D 2004.

He has published more than 30 papers in international journals and conference proceedings, 20 of which were indexed by SCI / EI / ISTP. His research work is supported by Chinese National Natural Science Foundation, Guangdong Province research foundation and Shenzhen Municipal Science-Technology foundation.