RXT51 Real-time Multi-task Operating System and Applications

Chunmei WANG[*,1], Fei ZHOU [2]

[1]*Department of Internet Finance& Information Engineering*, Guangdong University of Finance, Guangzhou 510521, China.
[2] Guangzhou Yuefeng Communication Technology Co.Ltd, Guangzhou 510663, China

Abstract — **Embedded technology is one of the most promising technologies in the twenty-first Century, with high reliability, real-time operation, short system development cycle and convenient maintenance, and is wide applications. However, embedded systems often have some shortcomings, for example, limited CPU address space, small memory capacity, limited resources etc. Usually they serve a single function. In order to make embedded systems for real time multi task processing, we develop solutions to overcome the limitations of conventional methods of software for complex applications. This paper proposes the real-time multitasking operating system RXT51 for MCS-51 microcontroller, and further studies the characteristic of RXT51 real-time multitasking operating system, its structure and functions. Through specific examples of using RTX51 we cover: i) the development of control programs for parallel processing, ii) the improvement of the performance of multitasking systems, iii) suggestions to improve systems with poor resources, iv) parallel processing and v) real-time multitasking in embedded systems.**

Keywords - Embedded system; Real-time multitasking; RXT51; Parallel processing

I. INTRODUCTION

A. *Propose for the some problems*

The application software of the single chip microcomputer are conventionally composed of main program and interrupt handlers. The main program is generally a timing cycle program, and the interrupt service procedures is to deal with various random events and the real-time control of. But for the powerful system application of complex situations, such as a system that needs to control a plurality of objects, and the controlled object require real-time control[1], namely each control object information must be processed within the prescribed period of time and in response as soon as possible. For these multi object systems, still using the conventional method to write software, it is undoubtedly very difficult to achieve real-time processing of multi object. Methods of conventional design software of SCM has many limitations for complex applications, mainly for the following reasons：

a) Program structure complex

The more functions and the more controlled objects that the system has, the more complex the software should be. SCM is both limited in program storage space and data storage space. If a large number of interrupt occurs, it may make the program large and difficult to design and debug.

b)The real-time differential

The CPU does not allow a lower or the same level interrupt response in the treatment of an interrupt processing. Only after the completion of the current interruption, and there is no interrupt request other higher priority level, can give response. In order to further improve the real-time requirements, Requiring the interrupt service routine as short as possible. Similar to the situation, low-level interrupt or at the interruption is very difficult to give timely treatment.

c) It is difficult to communicate with each other between every control processes

The communication between the main program and interrupt service program can be achieved when used to protect the scene and to recover the scene in the interrupt service program stack operations to, but communication in between the interruption will be difficult to achieve.

The best way to solve the above problems is to become independent, the application can run in parallel with the completion of the task according to the function division. Such as the data acquisition program, data processing program, output control program, print program etc.. The whole software is composed of each task. This can have communication links between certain horizontal such that each task, including synchronization and mute operation[2]. Single processor can run multiple tasks at the same time, and on the macro, each task is in parallel operating, all the system resources are the shared task. The interrupt is an organic part of multi task mechanism, that is to say, the real-time response to external events are realized by the interrupt response, but operated by the control structure to inform the corresponding task to complete. The control mechanism is the real-time multi task operating system.

B. *Real Time Operating System*

Operating system is the software management system of computer resources. It is the interface between the application and the computer hardware. Computer real-time multi task operating system refers to the operating system supporting real-time control system work, and real-time refers to meet the time constraints and requirements, and multi mission is to allow multiple concurrent execution.

*1)*Task

In order to facilitate the processing, the application is divided into many program modules independently, which cooperate each other. The process of these independent processing function of the program for its data in computer processing operation called task or process. The task is a dynamic process, and it has a life cycle including create, act and die.

Each task is relatively independent. For example, a MCU control system including a keyboard and a display which has many tasks, for instance, data acquisition, date calculating, control output, keyboard scanning and output display. On the macro, These tasks are also running in parallel. At the micro level, each task has 3 states: running, ready and blocking state. A task once established, is always in one of the 3 states. The task of the single CPU system running state is always only one, and the exclusive CPU. Ready task representation has all the working conditions, but because of other tasks in the operation, so it can only wait in line operation; blocking state is the task needing to wait for a resource or an event, then it cannot run in a lockdown. Only when the condition is satisfied (after interruption or other tasks to incentive), it can be ready to work or to operate.

2)Functions of real-time multi task operating system

The tasks running in real-time systems is controlled by the real-time multi task operating system [3], which usually have the following basic functions:

*a)*The task management (multi task and priority based task scheduling);

*b)*Inter task synchronization and communication (semaphore and email etc.);

*c)*Memory optimization management (including ROM management);

*d)*Real time clock service;

*e)*The interrupt management service.

II. RTX51 REAL-TIME MULTI-TASK OPERATING SYSTEM

RTX51 is developed by Keil Software Company, designed specifically for MCS-51 real-time multi task operating system [4] application system. The kernel source code, completely open to the user, truly realize the management of multiple tasks in embedded system on single CPU (process) of concurrent operation. Single chip itself due to resource constraints, and its function is relatively simple, but its performance is fully able to meet the application requirement of embedded system with 51 series microcontroller as the core. It can simplify the engineering design system and software complexity and time limited engineering development.

RTX51 has the full version of RTX51 Full and the smallest model version of RTX51 Tiny two versions.

Task time slice the full version of RTX51 Full to allow the 4 priority round robin scheduling and preemptive task switch in parallel, using interrupt function. Each signal and information of each task can be transferred through a "mailbox" system, and it can allocate and free memory from one storage pool, can force a task execution stops, waiting for an interrupt, can force a task waiting timeout interrupt, and from another task or interrupt signals or information.

The minimal model version of the RTX51 Tiny is a subset of RTX51, very delicate, and it occupies only 800 bytes of program memory space. It can run in the absence of any external memory RAM (XDATA) 8051 system operation, but applications can still access the external memory. RTX51 Tiny only supports the time slice rotation task switching and signal task scheduling, parallel use the interrupt function, can force a task waiting timeout interrupt, and signals from another task or interrupt issued. But RTX51 Tiny does not support preemptive task scheduling, cannot be semaphore operation, also do not support memory allocation or release. The main function of RTX51 is task management, memory management.

A. Management tasks

1)RTX51 tasks

RTX51 has 2 types: quick task and standard task. Quick task has fast response, each quick task using 8051 a separate register group, and have their own regional stack. RTX51 supports 3 quick tasks at maximum at the same time. Standard tasks are stored in an external RAM (XDATA) by their own device context. Standard task use relatively less the internal RAM, and all of the standard task uses 1 registers and stack. When a task switch, it shifts register state and stack element of the current mandate to external memory. RTX51 supports a maximum of 16 standard tasks [5]. The RTX51 task state can be divided into 4 kinds of state:

*a)*running: in the running task, at the same time only one task is running;

b) ready : in a waiting operation task, quit running state in the currently running task, the ready queue, the highest priority task will enter operation state;

*c)*blocked: the task is in wait for an event, if the event occurs and has priority comparing with the running task, the event is to switched to run; but if the low priority than the running task, this task in the READY state;

*d)*deleted: the task is not in the loop in the execution of the rotary switch.

Task state transition diagram shown in figure 1.

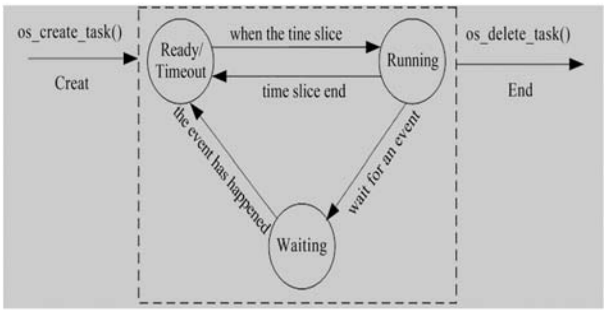


Fig. 1 state transition diagram

The RTX51 task has 4 priorities: 0, 1, 2 can be assigned to the standard task; priority 3 is reserved for the fast task.

Each task can occur to waiting for events, without increasing the burden of the system.

2) RTX51 events

The RTX51 event (event) can be timed out, waiting for news, signal, time interval, the interrupt event or their combination.

a) Timeout: from the beginning of the os_wait function of the time delay, the duration can be determined by the timing beat number.

b) Interval: from the beginning of the os_wait function and the time interval, the interval time can be determined by the timing beat number.

c) Signal: as the task of communication between a system function, can be used to set or clear.

d) Message: suitable for RTX51 Full, for the exchange of information.

e) Interrupt: suitable for RTX51 Full, the amount of signal system for management of shared resources.

3)Task control block

In order to describe and control tasks running kernel defines a data structure, called task control blocks for each task, including three:

*a)*ENTRY[task_id]:task_id task code entrance address in CODE space, 2 bytes.

*b)*STKP[taskid]: taskid task is using stack location of stack bottom, in the internal RAM (IDATA space), 1 bytes.

*c)*STATE[taskid].time and STATE[tasked].

state : the former said the timing beat counter task, at each time tick interrupt timing after reduction since a; the latter said task state register, with its various bits to represent the state of a task. In IDATA space, occupy 2 bytes.

4)Task scheduling

Task scheduling is according to a certain algorithm to dynamically assign CPU to the need to perform a task.

RTX51 TINY can only use non deprived preemptive mode, utilizing the 8051 internal timer T0 to generate timed rhythm, the task only can be worked in timing beat their assigned number (time slice). When a task is executed after the time slice, then the running the task is interrupted, and switch to the redeployment of the high priority task from the ready queue. If "several tasks ready" status is the same priority, then the first to enter the "ready" state of the first implementation.

RTX FULL also allows the use of deprivation of preemptive mode. When the task of the events occurs, once the priority level is higher than the running task high task, it is to force the running task to give up CPU, and CPU is used by a higher priority task.

B. Memory management

The kernel uses the static allocation of storage space strategy of KEIL C51 compiler to deal with the global variables and local variables, so the memory management is simplified as the stack management. The kernel for each task to retain a separate stack area, and all stacks are managed in the internal RAM (IDATA) space. In order to assign the largest area for the currently running task, each task with the stack position is dynamic, and the use of STKP[taskid] to record for each task stack location is dynamic, and to record and task stack bottom of stack location with the STKP[taskid]. When a stack of free space is less than FREESTACK (default 20) bytes, it will allocate STACK_ERROR, and make error stack management.

The following situations using stack management:

a) The task of creating, 2 bytes free stack space, assigned to the new task of innovation in task_id, and ENTRY [task_id], into the stack;

b) The task scheduling, tasks will be free of stack space allocation is running;

c) Delete tasks, task task_id stack space for recovery of deleted, and converted to the free stack space.

Figure 2 is to show the other not-working task released stack space for the currently running task.

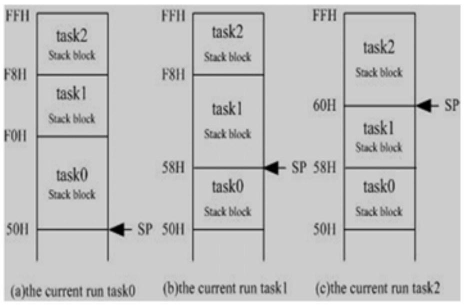
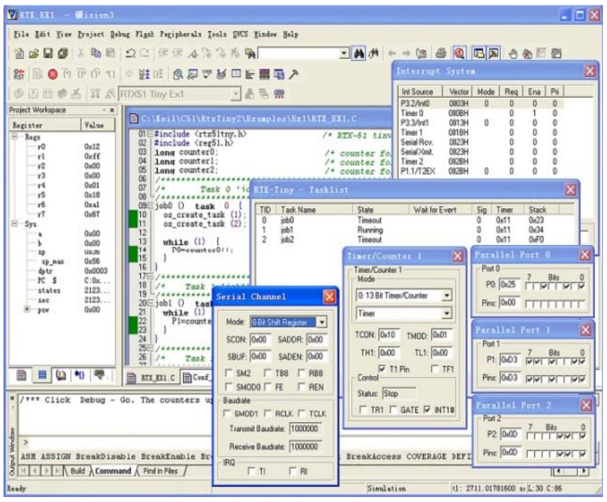


Fig. 2 Variation of the on-chip RAM stack

III. The use of RTX51

A. RTX51 system running environment


RTX_EX1 - µVision3

C:\Keil\C51\RtxTiny2\Examples\Ex1\RTX_EX1.C

```
#include <rtx51tny.h>                 /* RTX-51 tiny
#include <reg51.h>
long counter0;                         /* counter fo
long counter1;                         /* counter fo
long counter2;                         /* counter fo
/*********************
/*       Task 0 'jo
/*********************
job0 () _task_ 0 {
  os_create_task (1);
  os_create_task (2);

  while (1) {
    P0=counter0++;
  }
}
/*********************
/*       Task
/*********************
job1 () _tas
  while (1)
    P1=counte
  }
}
/*********************
/*       Task
```

Int Source	Vector	Mode	Req	Ena	Pri
P3.2/Int0	0003H	0	0	0	0
Timer 0	000BH		0	1	0
P3.3/Int1	0013H	0	0	0	0
Timer 1	001BH		0	0	0
Serial Rcv.	0023H		0	0	0
Serial Xmit.	0023H		0	0	0
Timer 2	002BH		0	0	0
P1.1/T2EX	002BH	0	0	0	0

TID	Task Name	State	Wait for Event	Sig	Timer	Stack
0	job0	Timeout		0	0x11	0x23
1	job1	Running		0	0x11	0x34
2	job2	Timeout		0	0x11	0xF0

Register	Value
r0	0x12
r1	0xff
r2	0x00
r3	0x00
r4	0x01
r5	0x18
r6	0xa1
r7	0x67
a	0x00
b	0x00
sp	0x36
sp_max	0x56
dptr	0x0003
PC $	C:0x...
states	2123...
sec	2123...
psw	0x00

Serial Channel: Mode: 8-Bit Shift Register; SCON: 0x00; SADDR: 0x00; SBUF: 0x00; SADEN: 0x00; Transmit Baudrate: 1000000; Receive Baudrate: 1000000

Timer/Counter 1: Mode: 0: 13 Bit Timer/Counter; Timer; TCON: 0x10; TMOD: 0x01; TH1: 0x00; TL1: 0x00; T1 Pin ✓; Status: Stop

Parallel Port 0: P0: 0x25; Pins: 0x00

Parallel Port 1: P1: 0xD3; Pins: 0xD3

Parallel Port 2: P2: 0x00; Pins: 0x00

```
/*** Click Debug - Go. The counters u
ASM ASSIGN BreakDisable BreakEnable Br ... BreakAccess COVERAGE DEFI
```

Ready   Simulation   t1: 2711.01781600 s   L:30 C:86

Fig. 3. Keil integrated development environment

With the continuous development of software development technology of SCM, programming voice to from the traditional assembly language into a high-level language. For the MCS-51 Series, MCU is currently popular C51 programming language, it can using C programming language in the Keilu Vision integrated development environment. Vision provides including macro assembler, compiler, C function library, connector and simulation debugger component perfect development plan. Keilu Vision also comes with RTX51 Tiny, and RTX51 FULL need to install. Figure 3 is a Keilu Vision simulation debugging interface.

RTX51 Tiny kernel code, written in assembly language, is short and pithy. It consists of two source files including conf_tny.a51 and rtx51tny.h. The former is a configuration file, used to need to be defined in the operation of the system of global variables and the stack error macro STACK_ERROR, the global variables and macros, the user can according to their own system of flexible configuration modification; the latter is the system kernel, all function contains the system call. Application to the rtx51tny.h header files included, together with the configuration file is compiled and linked together[6][7].

B. *Multi task programming*

1)Time slice round robin

RTX51 Tiny uses the time slice rotation to quasi parallel execution of multiple tasks, and each task at a predefined time slice can be implemented. Time is connected to the tasks being executed, and it makes another task start execution [8]. The following example using RTX51 Tiny time slice rotation task scheduling, task job0 using P1.0, P1.1 and P1.4 pins to simulate an OR gate, task job1 using P1.2, P1.3 and P1.5 pins the simulation of an and gate, as shown in figure 4.

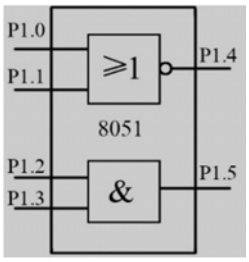


Fig. 4. Simulation of digital logic

The program included the header file rtx51tny.h, so there is no need to have the main function in the program, because the kernel of RTX51 Tiny operating system has its own main function main (). RTX51 Tiny will automatically start to run the task 0. If the application has the main function, so need to use the OS to create task function RTX51 in Tiny in the main function to create a task or to use the OS start system function in RTX51 to start RTX51.

RTX51 uses timer 0 overflow interrupt as a time slice rotation control, generated periodic interrupt is driving the RTX51's clock [9]. The configuration parameters of RTX51 Tiny (in conf_tny.a51) is the following two lines:

```
define Hardware-Timer Overflow in 8051 machine cycles.
INT_CLOCKEQU10000; default is 10000 cycles.
define Round-Robin Timeout in Hardware-Timer Ticks.
TIMESHARINGEQU5; default is 5 ticks.
```

INT_CLOCK is the cycle of clock interrupt , also is the basic time slice; the default value is 1000 machine cycles if the system clock frequency of 12MHz, a machine cycle is 1 * 12/12MHz = 1 x 10^{-6}S = 1μS. The basic time slice of the default is 10000μS = 10ms.

TIMESHARING is the number of time slices to each task every time, the default value is 5. When TIMESHARING is setted to 0, the tasks automatically are not switched in system, then the need to switch tasks with os_switch_task function.

Thus, the maximum time slice is determined both INT_CLOCK and TIMESHARING together to each task each time for using. For example, under INT_CLOCK= 10000, when TIMESHARING=1, the maximum time slice of a task is 10ms; while TIMESHARING=2, task using the largest time slice is 20ms; if TIMESHARING=5, task using the largest time slice is 50ms.

2)Time overflow events

RTX51 allows the use of time out events to control task scheduling. That is, time slice of waiting for a task is arriving, you can also use os_wait () function to notify RTX51 that it can make another task start execution. Os_wait () function suspends a task to wait for an event. It can synchronize 2 or several tasks. Its working process is as follows: when the events to wait for task did not occur, the system suspends this task; when the event occurs, the system switches tasks according to the task scheduling rules.

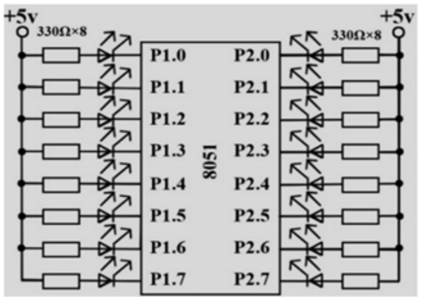


Fig.5 LED lights

Figure 5 is the LED lighting circuit, P1 and P2 port are used to drive the LED, task0 and task1 respectively make the P1 and the P2 port lighting loop shift, requirement shift speed of the two ports is different, tasks using waiting for occurrence of time overflow events is as delay of display toggle [10].

As previously described, firstly, creating the job1, then put the display data to P1 port, job0 call OS-wait function to suspend 50 clocks signal's time, then the RTX51 switches to the next task job1, after the job1 send display data to P2 port, it calls OS wait to suspend the 100 clocks the signal's time. If the current RTX51 has no executed other tasks, before execution job0 continually , it can it enters an empty loop to wait for the 50 clock signal time. The results of the example is that the P1 port's LEDs display are updated every 50 clocks cycle , while the P2 port's LEDs display are updated every 100 clocks cycle.

3)Signal

You can use OS-wait (K_SIG, 0, 0) function to suspend a task and waiting from another task execution os_send_signal ()function issued a wake-up signal to execution the task continually. Before the wake-up signal is sent , the task would have been suspended. From the executive order, the program before a task executes os_send_signal ( ) function prior to the procedure behind another task executes os_wait (K_SIG, 0, 0) function, the RTX51 signal can be used for synchronous operation between tasks.

IV. CONCLUSION

The above application examples can see efficiently support RTX51 TINY for multi task system, very suitable for single chip microcomputer application system running in of the scarce resource. If you hope to achieve more comprehensive support of RTX51 FULL version, RTX51 FULL has high requirement to the system hardware resources, often need to expand external data memory, and RTX51 FULL allows you to specify the priority of the task, a task with a higher priority becomes unavailable, would interrupt a low priority tasks or rob performing in front of it, this is called priority tasks or simply called preemptive mechanism. You can also by sending a token and waits for the token function to complete the semaphore operation, make the task of getting the continue. There are more advanced box (mail box) communication mode of operation etc.We have successfully used RTX51 OS in the Beidou satellite short message communication system.

REFERENCE

[1] Chun Mei WANG.“Application of frequency sweeper virtual digital frequency synthesis”[J], Computer applications and software, Vol 30,No 4,2013,pp.330-333.

[2] ChunMei WANG, WeiCai XIAO. Second-order IIR notch filter design and implementation of digital signal processing system”[J], Applied Mechanics and Materials,Vols.347-350,2013,pp.729-732.

[3] Chen Mingji, Zhou Ligong. “Embedded real-time operating system Small RTOS51 principle and application”[M]. Beijing: Beihang University press, 2004.

[4] XuAijun, PengXiuhua.“The microcontroller high-level language C51 Windows programming environment and application ”[M].Beijing: Publishing House of electronics industry, 2001.

[5] Zhang Hongbing, “Application of RTX-51Tiny real-time multi task operating system”[J]. Journal ofXianning University, vol 24 ,No. 6 , 2004,pp. 30-33.

[6] Zhou Hangci, “Programming technology of based on embedded real-time operating system”[M]. Beijing: Beihang University press, 2011

[7] DuanZhongxing, Zhang Deyuan,“Priority optimization and reversal and prevention ofthe real-time multi task operating system”[J], Computer engineering and science, No. 5,2003,pp.62-64.

[8] Feng Bin, Gong Zhuo, Yang Xuejun,“Memory management strategy of Real time operating system protected mode ”[J], Journal of Huazhong University of Science and Technology (NATURAL SCIENCE EDITION),No 8, 2002 ,pp.94-96.

[9] Yang Yan, Jiang Li, Yang Zhongling et al,“The embedded operating system RTX51 Tiny analysis and application of [J]”. Computer technology and development, No. 6,2006,pp89~91.

[10] Yang Wenlong. “Principle and application of system design of single chip microcomputer”[M],Tsinghua University press, 2011.