第八章 RTX-51 实时操作系统

RTX51 是一个适用于 8051 家族的实时多任务操作系统。RTX51 使复杂的系统和软件设计以及有时间限制的工程开发变得简单。RTX51 是一个强大的工具，它可以在单个 CPU 上管理几个作业（任务）。RTX51 有两种不同的版本。

RTX51 Full 允许 4 个优先权任务的循环和切换，并且还能并行的利用中断功能。RTX51 支持信号传递，以及与系统邮箱和信号量进行消息传递。RTX51 的 os_wait 函数可以等待以下事件：中断、时间到、来自任务或中断的信号、来自任务或中断的消息、信号量。

RTX51 Tiny 是 RTX51 Full 的一个子集。RTX51 Tiny 可以很容易的运行在没有扩展外部存储器的单片机系统上。但是，使用 RTX51 Tiny 的程序可以访问外部存储器。RTX51 Tiny 允许循环任务切换，并且支持信号传递，还能并行的利用中断功能。RTX51 Tiny 的 os_wait 函数可以等待以下事件：时间到、时间间隔、来自任务或者中断的信号。

本章节以后的部分用 RTX-51 来指代 RTX-51 Full 和 RTX-51 Tiny。在两者之间不同的地方会加以说明。

导言

许多微处理器应用都需要同时执行多个作业或者任务。对于这种应用，一个实时的操作系统（RTOS）允许系统资源（CPU、内存等）被灵活的分配给几个任务。RTX-51 是一个强大的实时操作系统，并且易于应用。RTX-51 可以工作在 8051 系列的微处理器上。

你使用标准C语言编写RTX-51应用程序，并且用C51来编译它们。为了具体指明任务的标志和优先级，会与标准C存在一点差别。RTX-51应用程序要求你将RTX51.H或者RTX51TNY.H头文件包含进来。当你在µVision2集成环境里打开目标选项对话框，选择目标操作系统以后，链接器便会添加合适的RTX-51库文件。

单任务程序

一个标准 C 程序从主函数开始执行。在嵌入式应用里，主函数经常被编写为一个无穷循环，也可以被认为是一个连续执行的单个任务。例如：

```
int counter;

void main (void) {
  counter = 0;

  while (1) {                                    /* repeat forever */
    counter++;                                 /* increment counter */
  }
}
```

循环任务切换

RTX51 Tiny 允许“准并行”的同时执行几个任务。每一个任务在预先定义好的时间片内得以执行。时间到使正在执行的任务挂起，并使另一个任务开始执行。下面的例子使用了循环任务切换的技术。

使用RTX51的C程序例子

```
#include <rtx51tny.h>                      /* Definitions for RTX51 Tiny */
int counter0;
int counter1;

job0 () _task_ 0  {

  os_create_task (1);                   /* Mark task 1 as "ready"       */

  while (1)  {                          /* Endless loop                 */
    counter0++;                         /* Increment counter 0          */
  }
}

job1 ()  task  1  {
  while (1)  {                          /* Endless loop                 */
    counter1++;                         /* Increment counter 1          */
  }
}
```

RTX51 从任务 **0**（分配给作业 **0**）开始执行程序。**os_create_task** 函数标记任务 **1**（分配给作业 **1**）为准备执行。这两个任务是简单的计数循环。在一个时间片结束后，**RTX51** 中断作业 **0**，并且开始执行作业 **1**。作业 **1** 在一个时间片结束后,系统重新开始执行作业 **0**。

os_wait 函数

os_wait 函数提供了一种更为有效的方式来给几个任务分配可使用的处理器时间。**os_wait** 函数中断当前正在运行的任务，并且等待特定的事件。在一个任务等待事件的时间里，其他任务可以被执行。

等待时间到

RTX51 使用 **8051** 的一个定时器来产生一个循环的中断（时钟周期）。响应 **os_wait** 的最简单事件是时间到，当前正在执行的任务被指定的时钟周期所中断。下面的延时例子使用的是时间到。

使用 **os_wait** 函数编程

```
#include <rtx51tny.h>           /* Definitions for RTX51 Tiny */

int counter0;
int counter1;

job0 () _task_ 0  {

  os create task (1);

  while (1)  {
    counter0++;                 /* Increment counter 0         */
    os_wait (K_TMO, 3, 0);      /* Wait 3 timer ticks          */
  }
}

job1 () _task_ 1  {
  while (1)  {
    counter1++;                 /* Increment counter 1         */
    os_wait (K_TMO, 5, 0);      /* Wait 5 timer ticks          */
  }
}
```

这个程序与上一个程序相似，不同的是作业 **0** 是在计数器 **0** 完成计数后被 **os_wait** 函数所中断的。**RTX51** 等待 **3** 个时钟周期直到作业 **0** 准备好再次运行为止。在这期间，作业 **1** 得以执行。作业 **1** 也调用了 **os_wait** 函数，等待 **5** 个时钟周期。结果是：定时器 **0** 每三个时钟周期增加一次，计数器 **1** 则每 **5** 个时钟周期增加一次。

等待信号

os_wait 函数的另一个事件是信号。信号被用来协调任务。直到另一个任务发出信号，在 **os_wait** 函数控制下的任务才结束等待状态。如果信号预先就被发送出来，那么任务将立即继续执行。

使用等待信号的程序

```
#include <rtx51tny.h>           /* Definitions for RTX51 Tiny */

int counter0;
int counter1;

job0 () _task_ 0  {

  os create task (1);

  while (1)  {
    if (++counter0 == 0) {      /* On counter 0 overflow       */
      os_send_signal (1);       /* Send signal to task 1       */
    }
  }
}

job1 ()  task  1  {
  while (1)  {
    os_wait (K_SIG, 0, 0);      /* Wait for signal; no timeout */
    counter1++;                 /* Increment counter 1         */
  }
}
```

在这个例子当中，任务 **1** 等待着由任务 **0** 发出的信号，并且以此来处理计数器 **0** 产生的溢出。

抢先任务切换

RTX51 Full 提供了抢先的任务切换，**RTX51 Tiny** 不具备这个功能。为了对多任务的概念有一个完整的了解，在这里对抢先任务切换加以解释。

在上一个例子中，任务 **1** 收到一个信号后不会立即开始，只有当任务 **0** 发生了时间到事件后，任务 **1** 才会启动。如果任务 **1** 被赋予了比任务 **0** 高的优先级，通过抢先任务切换，如果任务 **1** 收到了信号，就会立即开始。优先级在任务定义中被指定（默认的优先级是 **0**）。

RTX51 的技术参数

描述	RTX-51 Full	RTX-51 Tiny
任务数量	最多 256 个；可同时激活 19 个	16 个
RAM 需求	40 到 46 字节 DATA 空间 20 到 200 字节 IDATA 空间(用户堆栈) 最小 650 字节 XDATA 空间	7 字节 DATA 空间 3 倍于任务数量的 IDATA 空间
代码要求	6KB 到 8KB	900 字节
硬件要求	定时器 0 或定时器 1	定时器 0
系统时钟	1000 到 40000 个周期	1000 到 65535 个周期
中断请求时间	小于 50 个周期	小于 20 个周期
任务切换时间	70 到 100 个周期（快速任务） 180 到 700 个周期（标准任务）取决于堆栈的负载	100 到 700 个周期 取决于堆栈的负载
邮箱系统	8 个分别带有整数入口的信箱	不提供
内存池	最多 16 个内存池	不提供
信号量	8＊1 位	不提供

RTX51 程序概览

下列表格里列出了 RTX51 的一些函数，并带有简要的说明和执行时间（针对 RTX51 Full）。

函数	描述	CPU 周期
isr_recv_message †	收到消息（来自中断调用）	71（具有消息）
isr_send_message †	发送消息（来自中断调用）	53
isr_send_signal	给任务发去信号（来自中断调用）	46
os_attach_interrupt †	分配中断资源给任务	119
os_clear_signal	删除一个以前发送的信号	57
os_create_task	将一个任务放入执行队列中	302
os_create_pool †	定义一个内存池	644（大小 20＊10 字节）
os_delete_task	从执行队列中移走一个任务	172
os_detach_interrupt †	移走一个分配的中断	96
os_disable_isr †	禁止 8051 硬件中断	81
os_enable_isr †	允许 8051 硬件中断	80
os_free_block †	归还一块存储空间给内存池	160
os_get_block †	从内存池获得一块存储空间	148
os_send_message †	发送一条消息（从任务中调用）	443（具有任务切换）
os_send_signal	向任务发送一个信号（从任务中调用）	408（具有任务切换） 316（具有快速任务切换） 71（没有任务切换）
os_send_token †	发送一个信号量（从任务中调用）	343（具有快速任务切换） 94（没有任务切换）
os_set_slice †	设置 RTX51 系统时钟时间片	67
os_wait	等待事件	68（用于等待信号） 160（用于等待消息）

† 标记的函数仅仅在RTX51 Full中具备

RTX51 Full里附加的调试和支持函数见下表：

函数	描述
oi_reset_int_mask	禁止 RTX-51 的外部中断资源
oi_set_int_mask	允许 RTX-51 的外部中断资源
os_check_mailbox	返回指定信箱的状态信息
os_check_mailboxes	返回所有的系统信箱的状态信息
os_check_pool	返回内存池中的块信息
os_check_semaphore	返回指定信号量的状态信息
os_check_semaphores	返回所有的系统信号量信息
os_check_task	返回指定任务的状态信息

函数	描述
os_check_tasks	返回所有的系统任务的状态信息

CAN 函数

CAN 函数仅在 RTX-51 Full 中提供。CAN 控制器支持非利浦 82C200 和 80C592 以及英特尔 82526。更多的 CAN 控制器正在准备中。

CAN 函数	描述
can_bind_obj	为一个任务绑定一个对象；当对象被接收的时候，任务启动
can_def_obj	定义通信对象
can_get_status	获取 CAN 控制器状态
can_hw_init	初始化 CAN 控制器硬件
can_read	直接读取一个对象的数据
can_receive	接收所有无界的对象
can_request	向一个指定的对象发送一个远程帧
can_send	通过 CAN 总线发送一个对象
can_start	开始 CAN 通信
can_stop	结束 CAN 通信
can_task_create	创建 CAN 通信任务
can_unbind_obj	断开任务和对象之间的绑定
can_wait	等待一个约束的对象被接收
can_write	向一个对象写入新数据，不用发送

TRAFFIC：使用 RTX-51 Tiny 的例程

TRAFFIC 例程是一个行人信号灯控制器，通过它来说明多任务实时操作系统 **RTX-51 Tiny** 的应用。在一个用户定义的时间段里，交通灯受到控制。在时间段以外，黄色灯闪烁。如果一个步行者按下了请求按钮，交通灯立即进入***行走***状态。否则，交通灯持续不断的工作。

交通灯控制器命令

下表列出了 **TRAFFIC** 所支持的一系列命令。这些命令由 **ASCII** 文本字符构成。所有的命令必须以回车符来终止。

命令	连续的文本	描述
Display	D	显示时钟，开始和结束时间
Time	T *hh:mm:ss*	设置当前时间为 **24** 小时格式
Start	S *hh:mm:ss*	设置开始时间为 **24** 小时格式。交通控制通常在开始和结束的时间段里操作，在此时间段以外，黄色灯闪烁。
End	E *hh:mm:ss*	设置结束时间为 **24** 小时格式

软件

TRAFFIC 应用程序由**3**个文件组成，这些文件可以在 \KEIL\C51\EXAMPLES\TRAFFIC 文件夹里找到。

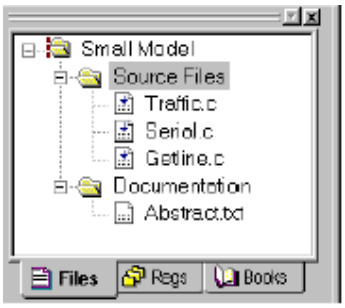
TRAFFIC.C 包含了交通灯控制程序，被分成了如下几个任务：

- **任务0 初始化**：初始化串口，并且启动所有其他的任务。由于初始化只需要进行一次，任务0将自己删除自己。
- **任务 1 命令**：任务 **1** 完成交通灯控制器的命令处理。这个任务负责控制和处理接收到的串行命令。
- **任务 2 时钟**：控制时钟。
- **任务 3 闪烁**：当时间落在活跃的时间段以外后，黄色灯闪烁。
- **任务 4 灯**：当时间落在活跃的时间段（在开始和结束时间之间）里以后，控制交通灯的相位。
- **任务 5 读按键**：读取行人按下的按钮，并且向任务 **4** 发送信号。
- **任务 6 退出**：如果在串行指令流里遇到了 **ESC** 字符，任务 **1** 获得一个信号，并且终止显示命令。

SERIAL.C 执行一个中断来驱动串行口。这个文件包含了函数 *putchar* 和 *getkey* 。高级的输入输出函数 *printf* 和 *getline* 调用这些基本的输入输出子函数。没有中断驱动串行输入输出，交通灯应用程序也会启动，但不会完成任务。

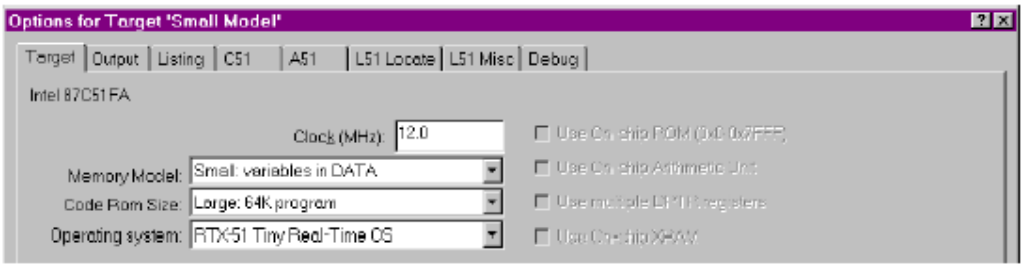
GETLINE.C 是一个命令行编辑器，用来编辑从串口接收到的字符。这个源文件也被**测量**应用程序所使用。

TRAFFIC 工程



在 µVision2 环境里，打开位于 \KEIL\C51\EXAMPLES\TRAFFIC 文件夹里的 TRAFFIC.UV2文件。在**工程窗口**的**文件**页里将会看到源文件。

在目标选项里选择 **RTX-51 Tiny Real-Time OS**。



选择**工程——编译**或者工具栏上的编译按钮，编译 **TRAFFIC** 程序。

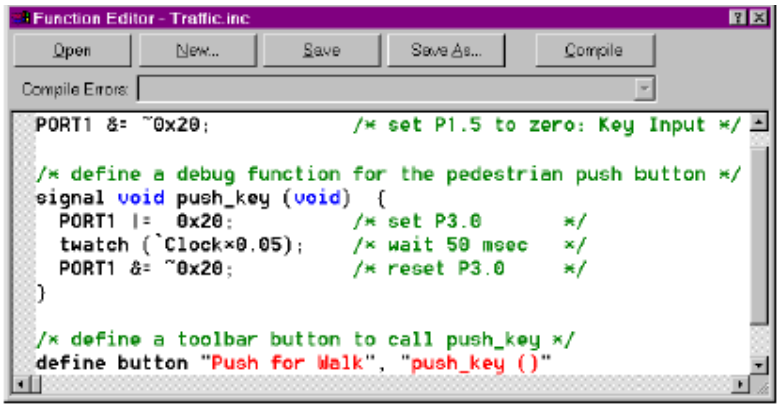
运行 TRAFFIC 程序

你可以使用µVision2 里的仿真来测试 **TRAFFIC** 应用程序。

右图的变量观察窗口允许你观察驱动交通灯的端口状态。

push_key 信号函数模拟行人按下按钮，切换交通灯进入*行走*状态。

这个函数被称为**按下以通过**工具按钮。



使用**调试——函数编辑器**，打开 **TRAFFIC.INC** 文件。在**目标选项**

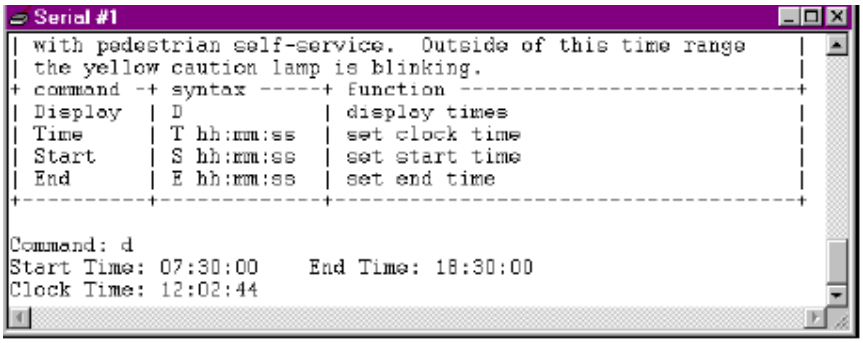
——**调试**——**初始化文件**里选择这个文件，并且定义信号函数 **push_key**，端口初始化和按钮工具栏。

注意：因为在 **TRAFFIC.C** 模块里有一个叫时钟的函数，所以 **VTREG** 符号 ***Clock*** 后面跟一个后引号（’）。参考 **121** 页的“字面符号”可以获得更多的信息。

 现在运行 **TRAFFIC** 应用程序。允许**查看—周期窗口更新**，在程序执行期间，通过观察窗口查看交通灯。

 **1 号串口**窗口显示了 ***printf*** 的输出，并且允许你输入下表里描述的交通灯控制器的命令。

设置时间在开始 / 结束时间段以外，让黄色灯闪烁。



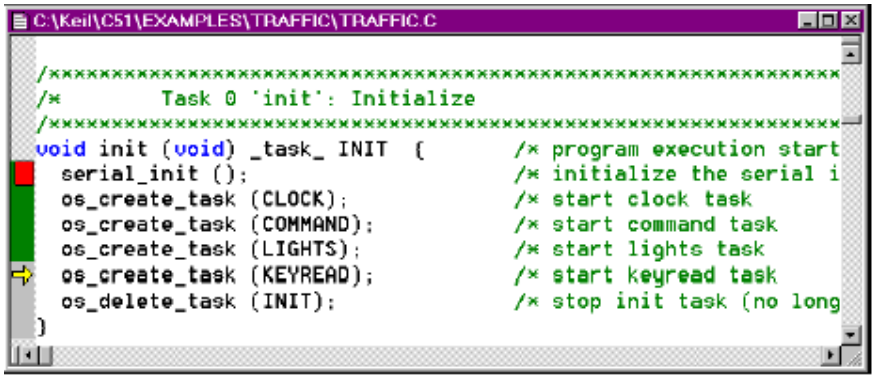
RTX 内核调试

一个 **RTX** 应用程序能够使用同标准 **8051** 应用程序相同的方法和命令进行测试。当你在**目标选项—目标**里选择了一种操作系统，µVision2 能够允许附加的调试特性：一个对话框列出了操作系统状态，并且当一个任务是活跃的时候，你可以应用***_TaskRunning_***调试函数来终止程序的运行。

下面的部分通过 **TRAFFIC** 例程来解释 **RTX** 的调试。

停止程序的运行，复位 **CPU**，取消所有断点。

一个 **RTX-51** 应用可以象标准应用一样使用相同的方式来测试。你可以打开源文件，设置断点，单步执行。**TRAFFIC** 应用从任务 **0** *初始化* 开始启动。



通过µVision2 可以完全清楚的了解内核。你可以使用菜单命令**外设—RTX Tiny 任务列表**来显示任务状态。

对话框 **RTX Tiny 任务列表**向你显示了如下信息：

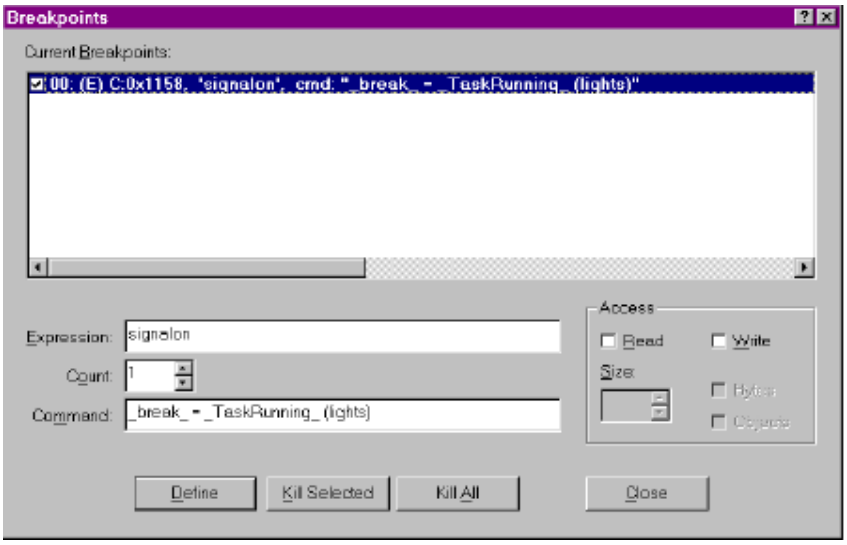
头	描述
TID	在任务函数定义里使用 ***task_id***
Task Name	任务函数名字
State	函数的任务状态；在下一个表里详细解释。
Wait for	任务等待的事件；下列事件是可能的（也可以是复合事件）
Event	**时间到**：任务 **计时器** 被设置为持续的时间，该时间在 ***os_wait*** 函数调用中被指定。当**计时器**减为 **0** 以后，任务便进入**准备**状态**.** **时间间隔**：任务 **计时器** 值被设置为时间间隔，该时间间隔在 ***os_wait*** 函数调用中被指定。当**计时器**减为 **0** 以后，任务便进入**准备**状态**.** **信号**：***os_wait*** 函数被 **K_SIG** 调用，并且任务等待 **SIG=1**。
Sig	分配给任务的信号位的状态
Timer	分配给任务的**计时器**的值。计时器值在每一个 **RTX** 系统时钟周期后减 **1**。如果**计时器**的值变为 **0**，并且任务等待**时间到**或**时间间隔**事件，那么任务进入**准备**状态。
Stack	当任务**运行**时，被使用的堆栈指针（**SP**）的值

RTX-51 Tiny 包含了一个有效地堆栈管理器，该管理器在 ***"RTX51 Tiny"*** *用户指南* 第五章：**RTX51 Tiny**，堆栈管理器中被解释。

该手册提供了详细的**堆栈**值的信息。

状态	描述
Deleted	尚未启动的任务处于**删除**状态。
Ready	等待执行的任务处于**准备**状态。当正在进行的任务结束后，**RTX** 启动下一个处在**准备**状态下的任务。
Running	当前正在运行的任务处于**运行**状态，在同一时刻，仅仅有一个任务处于**运行**状态。
Timeout	被循环任务切换时间到事件所中断的任务处于**时间到**状态，这个状态与**等待**状态等价；但是，循环任务切换是根据内部的操作过程被标记的。
Waiting	等待事件的任务处于**等待**状态。如果一个任务正在等待的事件发生了，该任务就进入**准备**状态。

只有当在***_TakRunning_***调试函数声明里指定的任务**正在运行**时，**调试—断点**对话框允许你定义停止程序运行的断点。参考 **134** 页“预定义函数”，可以找到关于***_TakRunning_***调试函数的详细描述。



仅仅当此刻**正在运行**的任务是**灯**的时候，在***signalon***函数里的断点才会停止程序的执行。