RTX51 Tiny(第 2 版)用户手册

目录

第一章 概述

RTX51 Tiny 是一种实时操作系统（RTOS），可以用它来建立多个任务（函数）同时执行的应用。嵌入式应用系统经常有这种需求。RTOS 可以提供调度、维护、同步等功能。

实时操作系统能灵活的调度系统资源，像 CPU 和存储器，并且提供任务间的通信。RTX51 Tiny 是一个功能强大的 RTOS，且易于使用，它用于 8051 系列的微控制器。

RTX51 Tiny 的程序用标准的 C 语言构造，由 Keil C51 C 编译器编译。用户可以很容易的定义任务函数，而不需要进行复杂的栈和变量结构配置，只需包含一个指定的头文件。

一、What's New

RTX51 Tiny 第二版增加了许多新特性,使得实时软件的开发更加简单,如:

- 支持 Code Banking

  该选项必须在 CONF_TNY.A51 配置文件中允许，还要在 L51_BANK.A51 文件中定义 Code Banking 硬件配置。

- 直接任务切换

  新增加的函数（os_swich_task）允许一个任务立即切换到另一个处于就绪态的任务。

- 任务就绪标志

  新的库函数 isr_set_ready 和 os_set_ready 允许用户给一个任务设置就绪标志。就绪标志可以用于将一个正在等待时间间隔、超时或信号（参见 os_wait）的任务置为就绪态，该任务在下一个运行时机恢复。

- CPU 空闲模式支持

- 支持用户在定时器中断的代码

  现在可以在定时器滴答中断中加入自己的代码。

  该选项必须在 CONF_TNY.A51 中被允许

- 支持时间间隔调整

当在 os_wait 中混合使用时间间隔和信号时，可用 os_reset_interval 函数调整时间间隔超时值。

此外，RTX51 Tiny 进行了完全重构，以增加灵活性，加快执行速度，减少代码和数据空间需求。

当满足以下条件时，RTX51 Tiny 第二版在代码大小上的缩小尤为显著。

1、禁止任务的时间轮转

2、尽量少的 RTX51 Tiny 系统函数调用

3、禁止栈检查

禁止任务时间轮转同时也降低了数据空间的需求。

二、已解决的问题

以下是在 1.06 版中已知的问题,已在第二版中得到了修正.

1、在 RTX51 Tiny1.06 中当在 os_wait 期间产生一个中断时，isr_send_signal 数可能会破坏就绪状态，导致任务挂起，等待从中断发来的信号，该问题在 RTX Tiny2 中已解决。

2、在 RTX51 Tiny1.06 中，由于信号产生时时间间隔定时器的值不能被调整，因而 K_IVL 和 K_SIG 事件不能在 os_wait 中合并为一个调用。在 RTX Tiny2 中，提供的 os_reset_interval 函数允许调整间隔定时器。

3、在 RTX51 Tiny1.06 中，TIMESHARING 不能被设为 1，如果设为 1，并且在时间片轮转前产生了中断，时间轮转周期可能被破坏，成为延迟 256 个滴答数，而不是 1 个。该问题在第 2 版中解决。

4、在 RTX51 Tiny1.06 中，当用户中断执行的时间比系统时钟滴答时间长时，RTX51 Tiny 系统时钟定时器就会递归调用，这导致 SAVEPSW 和 SAVEACC 的覆盖，引起系统崩溃。该问题在 RTX 51 Tiny 第 2 版中解决。如果在应用中包含一个执行时间大于 RTX51 Tiny 系统时钟定时间隔的中断程序，可以将 LONG_USR_INTR 设为 1。如果应用程序在高优先级中断程序中消耗大量时间，很可能会用到这个选项。

三、产品规约（产品规格说明）

参　数	范　围
最大任务数	16
最大活动任务	16
代码空间需求	900 字节最大
数据空间需求	7 字节
栈空间需求	3 字节/任务
外部 RAM 需求	0 字节
定时器	0
系统时钟因子	1000～65535
中断等待	20 个周期或更少
上下文切换时间	100～700 个周期

四、工具需求

以下为使用 RTX51 Tiny 需要的应用软件：

C51 编译器

A51 宏汇编器

BL51 连接器或 LX51 连接器

RTX51TNY.LIB 和 RTX51BT.LIB 库文件必须保存于库路径下,通常,该路径是"KEIL"C51"LIB 文件夹。RTX51TNY.H 必须保存在包含路径下,通常是"KEIL"C51"INC 文件夹。

五、目标需求

RTX51 Tiny 运行于大多数 8051 兼容的器件及其变种上。RTX51 Tiny 应用程序可以访问外部数据存储器，但内核无此需求。

RTX51 Tiny 支持 Keil C51 编译器全部的存储模式。存储模式的选择只影响应用程序对象的位置，RTX51 Tiny 系统变量和应用程序栈空间总是位于 8051 的内部存储区（DATA 或 IDATA 区），一般情况下，应用程序应使用小（SMALL）模式。

RTX51 Tiny 执行协作式任务切换（每个任务调用一个操作系统例程）

和时间片轮转任务切换（每个任务在操作系统切换到下一个任务前运行一个固定的时间段），不支持抢先式任务切换以及任务优先级。RTX51 Full 支持抢先式任务切换。

1、中断

RTX51 Tiny 与中断函数并行运作，中断服务程序可以通过发送信号（用 isr_send_signal 函数）或设置任务的就序标志（用 isr_set_redy 函数）与 RTX51 Tiny 的任务进行通信。

如同在一个标准的，没有 RTX51 Tiny 的应用中一样，中断例程必须在 RTX51Tiny 应用中实现并允许，RTX51 Tinyim 没有中断服务程序的管理。

RTX51 Tiny 使用定时器 0、定时器 0 中断，和寄存器组 1。如果在程序中使用了定时器 0，则 RTX51 Tiny 将不能正常运转。你可以在 RTX51 Tiny 定时器 0 的中断服务程序后追加自己的定时器 0 中断服务程序代码（参见硬件定时器）。

RTX51 Tiny 假设总中断总是允许（EA=1）。RTX51 Tiny 库例程在需要时改变中断系统（EA）的状态，以确保 RTX51 Tiny 的内部结构不被中断破坏。当允许或禁止总中断时，RTX51 Tiny 只是简单的改变 EA 的状态，不保存并重装 EA，EA 只是简单的被置位或清除。因此，如果你的程序在调用 RTX51 例程前某止了中断，RTX51 可能会失去响应。

在程序的临界区，可能需要在短时间内禁止中断。但是，在中断禁止后，不能调用任何 RTX51 Tiny 的例程。如果程序确实需要禁止中断，应该持续很短的时间。

2、再入函数

C51 编译器提供对再入函数的支持，再入函数在再入堆栈中存储参数和局部变量，从而保护递归调用或并行调用。RTX51 Tiny 不支持对 C51 再入栈的任何管理。因此，如果在程序中使用再入函数，必须确保这此函数不调用任何 RTX51 Tiny 系统函数，且不被循环任务切掉所打断。

仅用寄存器传递参数和保存自动变量的 C 函数具有内在的再入性，可以无限制的调用 RTX51 Tiny。

非可再入 C 函数不能被超过一个以上的任务或中断过程调用。非再入 C51 函数在静态存储区段保存参数和自动变量（局部数据），该区域在函数被多个任务同时调用或递归调用时可能会被修改。

如果确定多个任务不会递归（或同时）调用，则多个任务可以调用非再入函数。通常，这意味着必须禁止循环任务调度，且该非再入函数不能调用任何 RTX51 Tiny 系统函数。

附注：

- 如果希望在多个任务或中断中调用再入或非再入函数，应当禁止循环任务调度。

3、C 库例程

可再入 C51 库函数可在任何任务中无限制的使用。对于非再入的 C51 库函数，同样有非可再入 C 函数的限制。

4、多数据指针

Keil C51 编译器允许使用多数据指针(存在于许多 8051 的派生芯片中), RTX51 Tiny 不提供对它们的支持.因此,在 RTX51 Tiny 的应用程序中应小心使用多数据指针。

从本质上说，必须确保循环任务切换不会在执行改变数据指针选择器的代码时发生。

附注：

- 如果要使用多数据指针，应该禁止循环任务切换。

5、运算单元

Keil C51 编译器允许使用运算单元（存在于许多 8051 的派生芯片中）。RTX51 Tiny 不提供对它们的支持。

因此，在 RTX51 Tiny 的应用程序中须小心使用运算单元。

从本质上说，必须确保循环任务切换不会在执行用运算单元的代码时发生。

附注：

- 如果希望使用运算单元，应禁止循环任务切换。

6、寄存器组

RTX51 Tiny 分配所有的任务到寄存器 0，因此，所有的函数必须用 C51 的默认设置进行编译，REGISTERBANK（0）。

中断函数可以使用剩余的寄存器组。然而，RTX51 Tiny 需要寄存器组区域中的 6 个永久性的字节，用于这些字节的寄存器组在配置文件中指定。

第二章 实时程序

实时程序必须对实时发生的事件快速响应。事件很少的程序不用实时操作系统也很容易实现。随着事件的增加，编程的复杂程度和难度也随之增大，这正是 RTOS 的用武之地。

一、单任务程序

嵌入式程序和标准 C 程序都是从 main 函数开始执行的，在嵌入式应用中，main 通常是一个无限循环，可以认为是一个持续执行的单个任务，例如：

```
void main (void)
{ while(1)          /*永远重复*/
   {
   do_something(); /*执行 do_something “任务” */
   }
}
```

在这个例子里，do_something 函数可以认为是一个单任务，由于仅有一个任务在执行，所以没有必要进行多任务处理或使用多任务操作系统。

二、多任务程序

许多 C 程序通过在一个循环里调用服务函数（或任务）来实现伪多任务调度。如：

```
      void main(void)
      {
   int counter="0";
   while(1)                    /*一直重复执行*/
  {
   check_serial_io();          /*检查串行输入*/
   process_serial_cmds() ;     /*处理串行输入*/
```

```
    check_kbd_io();            /*检查键盘输入*/
    process_kbd_cmds();        /*处理键盘输入*/
    adjust|ctrlr_parms();      /*调整控制器*/
    counter++;                 /*增加计数器*/
  }
}
```

该例中，每个函数执行一个单独的操作或任务，函数（或任务）按次序依次执行。

当任务越来越多，调度问题就被自然而然的提出来了。例如，如果 process_kbd_cmds 函数执行时间较长，主循环就可能需要较长的时间才能返回来执行 check_sericd_io 函数，导致串行数据可能被丢失。当然，可以在主循环中更频繁的调用 check_serial_io 函数以纠正这个问题，但最终这个方法还是会失效

三、RTX51 Tiny 程序

当使用 Rtx51Tiny 时，为每个任务建立独立的任务函数，例如：

```
void check_serial_io_task(void) _task_ 1
{ /*该任务检测串行 I/0*/ }
void process_serial_cmds_task(void) _task_ 2
{ /*该任务处理串行命令*/ }
void check_kbd_io_task(void) _task_ 3
{ /*该任务检测键盘 I/O*/ }
void process_kbd_cmds_task(void) _task_ 4
{ /*处理键盘命令*/ }
void startup-_task(void) _task_ 0
  {
        os_create_task(1);          /*建立串行 I/O 任务*/
        os_create_task(2);          /*建立串行命令任务*/
```

```
    os_create_task(3);          /*建立键盘 I/O 任务*/
    os_create_task(4);          /*建立键盘命令任务*/
    os_delete_task(0);          /*删除启动任务*/
}
```

该例中，每个函数定义为一个 RTX51 Tiny 任务。RTX51 Tiny 程序不需要 main 函数，取而代之，RTX51 Tiny 从任务 0 开始执行。在典型的应用中，任务 0 简单的建立所有其他的任务。

第三章 原理

RTX51 Tiny 用于管理目标系统的资源，本章讨论 RTX51 Tiny 如何使用这些资源。

一、定时器滴答中断

RTX51 Tiny 用标准 8051 的定时器 0（模式 1）生产一个周期性的中断。该中断就是 RTX51 Tiny 的定时滴答（Timer Tick）。 库函数中的超时和时间间隔就是基于该定时滴答来测量的。

默认情况下，RTX51 每 10000 个机器周期产生一个滴答中断，因此，对于运行于 12MHZ 的标准 8051 来说，滴答的周期是 0.01 秒，也即频率是 100HZ(12MHz/12/10000)。该值可以在 CONF_TNY.A51 配置文件中修改。

附注：

- 可以在 RTX51 的定时滴答中断里追加自己的代码。参见 CONF_TNY.A51 配置文件。
- 关于 RTX51 Tiny 如何使用中断可以参考概述中中断一节的叙述。

二、任务

RTX51 Tiny 本质上是一个任务切换器，建立一个 RTX51 Tiny 程序，就是建立一个或多个任务函数的应用程序。下面的信息可以帮助你快速的理解 RTX51 。

- 任务用新的关键字由 C 语言定义，该关键字是 Keic C51 所支持的。
- RTX51 Tiny 维护每个任务的正确状态（运行、就绪、等待、删除、超时）。
- 某个时刻只有一个任务处于运行态。
- 任务可能处于就绪态、等待态、删除态或超时态。
- 空闲任务（Idle_Task）总是处于就绪态，当定义的所有任务处于阻 塞状态时，运行该任务。

三、任务管理

每个 RTX51 Tiny 任务总是处于下述状态中的一种状态中。

状 态	描　述
运 行	正在运行的任务处于运行态。某个时刻只能有一个任务处于该状态。os_running_task_id 函数返回当前正在运行的任务编号。
就 绪	准备运行的任务处于就绪态。一旦运行的任务完成了处理，RTX51　Tiny 选择一个就绪的任务执行。一个任务可以通过用 os_set_ready 或 os_set_ready 函数设置就绪标志来使其立即就绪（即便该任务正在等待超时或信号）。
等 待	正在等待一个事件的任务处于等待态。一旦事件发生，任务切换到就绪态。Os_wait 函数用于将一个任务置为等待态。
删 除	没有被启动或已被删除的任务处于删除态。Os-delete-task 函数将一个已经启动（用 os_create_task）的任务置为删除态。
超 时	被超时循环中断的任务处于超时态，在循环任务程序中，该状态相当于就绪态。

四、事件

在实时操作系统中，事件可用于控制任务的执行，一个任务可能等待一个事件，也可能向其他任务发送任务标志。

os_wait 函数可以使一个任务等待一个或多个事件。

- 超时是一个任务可以等待的公共事件。超时就是一些时钟滴答数，　当一个任务等待超时时，其他任务可以执行。一旦到达指定数量的滴答数，任务就可以继续执行。
- 时间间隔（Interval）是一个超时（Timeout）的变种。时间间隔与超时类似，不同的是时间间隔是相对于任务上次调用 os_wait 函数的指定数量的时钟滴答数。
- 信号是任务间通信的方式。一个任务可以等待其他任务给它发信号 （用 os_send _signal 和 isr_send_signal 函数）。

•每个任务都有一个可被其它任务设置的就绪标志（用 os_set_ready 和 isr_set_ready 函数)。一个个等待超时、时间间隔或信号的任务可以通过设置它的就绪标志来启动。

•isr_set_ready 函数）。一个等待超时、时间间隔或信号的任务可以通 过设置它的就绪标志来启动。

下表是 os_wait 函数等待的事件：

K_IVL	等待制定的时间
隔 K_SIG	等待一个信号
K_TMO	等待指定的超时

os-wait 返回时，返回值表明发生了的事件：

返 回 值	**意 义**
RDY_EVENT	任务的就绪标志被置位
SIG_EVENT	收到一个信号
TMO_EVENT	超时完成或时间间隔到达。

os_wait 可以等待下面的事件组合：

•K_SIG | K_TMO:任务延迟直到有信号发给它或者指定数量的时钟滴答到达。

•K_SIG | K_IVL:任务延迟直到有信号到来或者指定的时间间隔到达。

附 注：

•K_IVL 和 K_TMO 事件不能组合

五、任务调度程序：

任务调度程序给任务分配处理器，RTX51 Tiny 调度程序用下列规则确定哪个任务要被运行：

当前任务被中断如果：

1、任务调用了 os_switch_task 且另一个任务正准备运行。

2、任务调用了 os_wait 且指定的事件没有发生。

3、任务执行了比轮转时间片更长的时间。

另一个任务启动如果：

1、无其它任务运行。

2、要启动的任务处于就绪态或超时态。

六、循环任务切换

RTX51 Tiny 可以配置为用循环法进行多任务处理（任务切换）。循环法允许并行的执行若干任务。任务并非真的同时执行，而是分时间片执行的（CPU 时间分成时间片，RTX51 Tiny 给每个任务分配一个时间片）。由于时间片很短（几毫秒），看起来好象任务在同时执行。

任务在它的时间片内持续执行（除非任务的时间片用完）。然后，RTX51 Tiny 切换到下一个就绪的任务运运行。时间片的持续时间可以通过 RTX51 Ting 配置

定义。

下面是一个 RTX51 Tiny 程序的例子，用循环法多任务处理，程序中的两个任务是计数器循环。RTX51 Tiny 在启动时执行函数名为 job0 的任务 0，该函数建立了另一个任务 job1，在 job0 执行完它的时间片后， RTX51 Tiny 切换到 job1。在 job1 执行完它的时间片后，RTX51 Ting 又切换到 job0，该过程无限重复。

```
  #include
int counter0;
int counter1;
void job0(void) _task_ 0
{
    os_create(1);          /*标记任务 1 为就绪*/
    while(1)
    {                      /*无限循环*/
        counter0++;        /*更新记数器*/
```

```
    }
  }
  void job1(void) _task_1
  {
while(1)
 {                    /*无限循环*/
  counter++;              /*更新记数器*/
    }
  }
```

附 注:

- 可以用 os_wait 或 os_switch_task 让 RTX51 Tiny 切换到另一个任务而不是等待任务的时间片用完。 os_wait 函数挂起当前的任务（使之变为等待态）直到指定的事件发生（接着任务变为就绪态）。在此期间，任意数量的其他任务可以运行。

七、协作任务切换

如果禁止了循环任务处理，就必须让任务以协作的方式运作，在每个任务里调用 os_wait 或 os_switch_task，以通知 RTX51 Tingy 切换到另一个任务。

os_wait 与 os_switch_task 的不同是，os_wait 是让任务等待一个事件，而 os_switch_task 是立即切换到另一个就绪的任务。

八、空闲任务

没有任务准备运行时，RTX51 Ting 执行一个空闲任务。空闲任务就是一个无限循环。如：

```
SJMP $
```

有些 8051 兼容的芯片提供一种降低功耗的空闲模式，该模式停止程序的执行，直到有中断产生。在该模式下，所有的外设包括中断系统仍在运行。

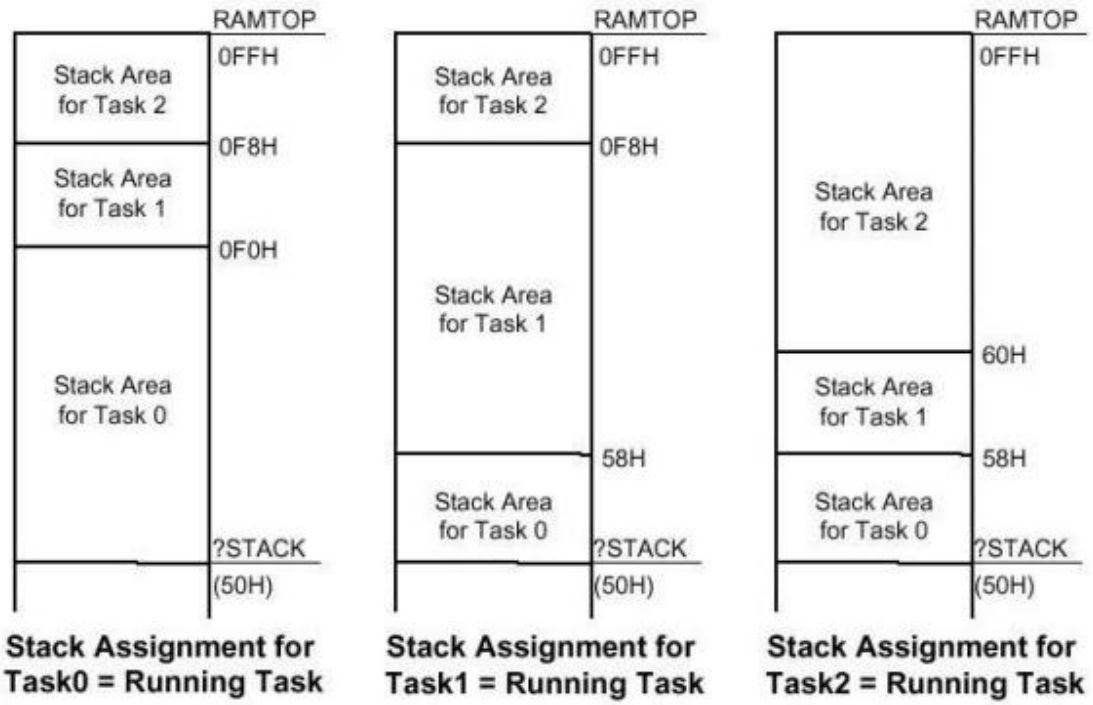
RTX51 Tiny 允许在空闲任务中启动空闲模式（在没有任务准备执行时）。当 RTX51 Tiny 的定时滴答中断（或其它中断）产生时，微控制器恢复程序的执行。

空闲任务执行的代码在 CONF_TNY.A51 配置文件中允许和配置。

九、栈管理

RTX51 Tiny 为每个任务在 8051 的内部 RAM 区（IDATA）维护一个栈。任务运行时，将得到可能得到的最大数量的栈空间。任务切换时，先前的任务栈被压缩并重置，当前任务的栈被扩展和重置。

下图表明一个三任务应用的内部存储器的布局。



？STACK 表示栈的起始地址。该例中，位于栈下方的对象包括全局变量、寄存器和位寻址存储器，剩余的存储器用于任务栈。存储器的顶部可在配置中指定。

第四章 RTX51 Tiny 配置

RTX51 Tiny 可根据应用的不同定制。

一、配置

建立了嵌入式应用后，RTX51 Tiny 必须要配置。所有的配置设置都在 CONF_TNY.A51 文件中，该文件位于"KEIL"CS1"RTXTINY2"目录下。在 CONF_TNY.A51 中的配置选项允许:

- 指定滴答中断寄存器组。
- 指定滴答间隔（以 8051 机器周期为单位）。
- 指定在滴答中断中执行的代理。
- 指定循环超时。
- 允许或禁止循环任务切换。
- 指定应用程序占用长时间的中断。
- 指定是否使用 code banking。
- 定义 RTX51 Tiny 的栈项。
- 指定最小的栈空间需求。
- 指定栈错误发生时要执行的代码。
- 定义栈错误发生时要执行的代码。
- 定义空闲任务操作。

CONF_TNY.A51 的默认配置包含在 RTX51 Tiny 库中。但是，为了保证配置的有效和正确，须得将 CONF_TNY.A51 文件拷贝到工程目录下并将其加入列工程中。

通过改变 CONF_TNY.A51 中的设置来定制 RTX51 Tiny 的配置。

附注：

- 如果在工程中没有包含配置文件（CONF_TNY.A51），库中的默认配置将自动加载，后续的改变将存储在库中，这样可能会对以后的应用起到不良影响。

1、硬件定时器

下面的常数指定 RTX51 Tiny 的硬件定时器如何配置。

- INT_REGBANK 指定用于定时器中断的寄存器组，默认为 1（寄存器组 1）。
- INT_CLOCK 指定定时器产生中断前的指令周期数。该值用于计算定时器的重装值（65536_INT_CLOCK）。默认该值为 10000。

- HW_TIMER_CODE 是一个宏，它指出在 RTX51 Tiny 定时器中断结尾处要执行的代码。该宏默认是中断返回，如：

```
HW_TIMER_CODE  MACRO
RETI
ENDM
```

2、循环

默认情况下，循环任务切换是使能的。下面的常数允许你配置循环任务切换的时间或完全禁止循环切换。

- TIMESHARING 指定每个任务在循环任务切换前运行的滴答数。设为 0 时禁止循环任务切换。默认值为 5 个滴答数。

3、长中断

一般情况下，中断服务程序设计为快速执行的程序，在某些情况下，中断服务程序可能执行较长的时间。如果一个高优先级的中断服务程序执行的时间比 RTX51 Tiny 滴答的时间间隔长，RTX51 Tiny 定时器中断可能被中断并可能重入（被后继的 RTX51 定时器中断）。

如果要使用执行时间较长的高优先级中断，应该考虑减少 ISR 中执行的作业的数量，改变 RTX51 定时器的滴答率使其低一些，或者使用下面的配置选项。

- LONG_USR_ISR 指示器是否有执行时间长于滴答时间间隔的中断（滴答中断除外）。当该选项设为 1，RTX51 Tiny 就会包括保护再入滴答中断的代码。该值默认为 0，即认为中断是快速的。

4、Code Banking

以下配置选项允许你指定 RTX51 Tiny 应用是否使用 code banking。

CODE_BANKING 指定是否使用 code banking。使用 code banking 时该选项必须设为 1，未使用 code banking 时，该选项须设为 0，默认的值为 0。

附注

- L51_BANK.A51 2.12 及其以上的需要 RTX51 Tiny 程序使用 code banking。

5、栈

一些选项用于栈配置。下面的常数定义用于栈区域的内部 RAM 的大小和栈的最小自由空间。一个宏允许指定当没有足够的自由栈时执行的代码。

- RAM TOP 指定片上栈顶部的地址。除所有位于栈之上的 IDATA 变量，否则不应修改该值。该值默认为 OXFF。
- FREE_STACK 指定栈允许的最小字节数。切换任务时，如果 RTX51 Tiny 检测到低于该值时，STACK_ERROR 宏将被执行。设为 0 禁止栈检查，默认设置是 20 字节。
- STACK_ERROR 是一个指定发生栈错误（少于 FREE_STACK 字节数）时要执行的指令的宏。该宏默认是禁止中断并进入无限循环：

```
STACK_ERROR MACRO
CLR    EA
SJMP $
ENDM
```

6、空闲任务

当没有任务准备运行时，RTX51 Tiny 执行一个空闲任务。空闲任务只是一个循环，不做任何事——只是等待滴答中断切换到一个就绪的任务。下列常数允许配置空闲任务。

- CPU_IDLE 宏指定空闲任务中执行的代码。默认的指令是置位 PCON 寄存器的空闲模式位（大多数 8051 设备适用）。这将停止执行程序，降低功耗，直到有中断产生：

```
CPU_IDLE  MACRO
ORL PCON， # 1
ENDM
```

- CPU_IDLE MACRO 指定在空闲任务中是否执行 CPU_IDLE 宏。

  默认为 0，CPU_IDLE 宏不包括在空闲任务中。

二、库文件

RTX51 Tiny 包括两个库文件：

- RTX51TNY.LIB 用于无代码分组(non_banking)的 RTX51 Tiny 程序。
- RTX51BT.LIB 用于代码分组（code_ banking）的 RTX51 Tiny 程序。

在"KEIL"C51"RTXTINYZ"SOURCECODE"下的 RTXTINZ.PRJ 工程用来建立这两个库。

附注：

- 应用时并不需要显式的包含一个 RTX51 Tiny 库。当使用 μVision 集成环境或命令行连接器时会自动执行。
- 建立 RTX51 Tiny 库时，默认配置文件（CONF_TNY.A51）包括在库中。如果在工程中未显示包含配置文件（CONF_TNY.A51），将从库中包含一个默认的，后续对配置文件的修改将存储到库中，这可能对你的应用产生负面影响。

三、优化

下面的事情是为了优化 RTX51 Tiny 程序应该做的。

- 如果可能，禁止循环任务切换。循环切换需要 13 个字节的栈空间存储任务环境和所有的寄存器。当任务切换通过调用 RTX51 Tiny 库函数（像 os_wait 或 os_switch_task）触发时，不需要这些空间。
- 用 os_wait 替代依靠循环超时切换任务。这将是提高系统反应时间和任务响应时间。
- 避免将滴答中断率设置的太快。

为了最小化存储器需求，从 0 开始对任务编号。

第五章 使用 RTX51 Tiny

一般地，下面三步是使用 RTX51 Tiny 要实现的

- 编写 RTX51 程序
- 编译并连接程序
- 测试和调试程序

一、编写程序

写 RTX51 Tiny 程序时，必须用关键字对任务进行定义，并使用在 RTX51TNY.H 中声明的 RTX51 Tiny 核心例程。

1、包含文件

RTX51 Tiny 仅需要包含一个文件：RTX51TNY.H。所有的库函数和常数都在该头文件中定义。你可以在你的源文件中包含它：

```
#include<rtx51tny.h>
```

2、编程原则

以下是建立 RTX51 Tiny 程序时必须遵守的原则：

①、确保包含了 RTX51TNY.H 头文件。

②、不要建立 main 函数，RTX51 Tiny 有自己的 mian 函数。

③、程序必须至少包含一个任务函数。

④、中断必须有效（EA=1），在临界区如果要禁止中断时一定要小心。参见概述中的中断一节。

⑤、程序必须至少调用一个 RTX51 Tiny 库函数（象 os_wait）。否则，连接起将不包含 RTX51 Tiny 库。

⑥、Task 0 是程序中首先要执行的函数，必须在任务 0 中调用 os_create_task 函数以运行其余任务。

⑦、任务函数必须是从不退出或返回的。任务必须用一个 while(1)或类似的结构重复。用 os_delete_task 函数停止运行的任务。

⑧、必须在 uvison 中指定 RTX51 Tiny，或者在连接器命令行中指定。更多技术文档参见 keil 软件知识库。

3、定义任务

实时或多任务应用是由一个或多个执行具体操作的任务组成的，RTX51 Tiny 支持最多 16 个任务。

任务就是一个简单的 C 函数，返回类型为 void，参数列表为 void，并且用_task_声明函数属性。例如：

```
void func (void)_task_task_id
```

这里，func 是任务函数的名字，task_id 是从 0 到 15 的一个任务 ID 号。

下面的例子定义函数 job0 编号为 0 的任务。该任务使一个计数器递增并不断重复。

```
void job0(void)_task_0
{
   while(1)
    {
    Counter0++;
   }
}
```

附注：

- 所有的任务都应该是无限循环，任务一定不能返回。
- 任务不能返回一个函数值，它们的返回类型必须是 void。
- 不能对一个任务传递参数，任务的形参必须是 void。
- 每个任务必须赋予一个唯一的，不重复的 ID。
- 为了最小化 RTX51 Tiny 的存储器需求，从 0 开始对任务进行顺序编号。

二、编译和连接

有两种方法编译和连接 RTX51 Tiny 应用程序。

- 用 uvison 集成开发环境
- 用命令行工具

1、命令行工具

RTX51 Tiny 已经完全集成到了 C51 编译语言中，这使得生成 RTX51 Tiny 应用非常容易。建立 RTX51 Tiny 程序只需编写 C 函数，无需使用汇编。

从命令行编译 RTX51 Tiny 程序…

按常规方式调用编译器，无需特别的编译指示。例如：

C51 RTXPROG.C DEBUG OBJECTEXTEND

产生的 RTXPROG.OBJ 文件中包含 C 代码和定义的 RTX51 Tiny 任务。

从命令行连接 RTX51 Tiny 程序：

- 在连接器命令行内指定 RTX51TNY 指示
- 在目标文件列表中包含 RTX_CONF.OBJ 文件（如果改变了配置）

例如：BL51 RTPROG.OBJ, RTX_CONF.OBJ RTX51TNY

RTX51TNY 指示命令连接器连接 RTPROG.OBJ 和 TX_CONF.OBJ 并且包含 RTX51 Tiny 库。这样就建立了 RTX51 Tiny 程序。

附注：

- 不要在 RTX51 Tiny 程序中建立 mian 函数，只建立任务函数就可以。main 函数包含在 RTX51 Tiny 库中，它启动操作系统和任务 0。如果在程序中包含了 main 函数，将产生一个连接错误指示有多个 main 被定义。
- 程序中至少建立一个任务函数。
- 必须至少调用一个 RTX51 Tiny 函数（象 os_wait 或 os_create_task），这样，连接器才会包含 RTX51 Tiny 库。

2、uvison 集成开发环境

用 uvison 建立 RTX51 Tiny 程序。

1) 打开目标对话框选项（从 project 菜单选择 Options for Target）。
2) 选择目标标签。
3) 从操作系统选项列表选择 RTX51 Tiny。

三、调试

uvison 模拟器允许运行和测试 RTX51 Tiny 应用程序。RTX51 Tiny 程序的载入和无 RTX51 Tiny 程序的载入是一样的。无需指定特别的命令和选项。

一个核心的对话框显示 RTX51 Tiny 核心和程序中任务的所有特征。从 Peripherals 菜单选择 RTX51 Tiny Tasklist 显示该对话框。

该对话框中：

- TID 是在任务定义中指定的任务 ID。
- Task Name 是任务函数的名字。
- State 是任务当前的状态。

- Wait for Event 指出任务正在等待什么事件。
- Sig 显示任务信号标志的状态（1 为置位）。
- Timer 指示任务距超时的滴答数，这是一个自由运行的定时器，仅在任务等待超时和时间间隔时使用。
- Stack 指示任务栈的起始地址。

第六章 函数参考

以下部分描述 RTX51 Tiny 的系统函数。函数依字母顺序排列，分为以下部分：

概要（Summary） 简述程序作用，列出包含的文件，包括它的声明和原型，语法举例，和参数描述。

描述（Description）程序的详细描述，如何使用。

返回值 程序返回值说明。

参阅（see also） 相关程序。

例子 如何正确使用该函数的程序例子中断。

附注：

- 以 os_开头的函数可以由任务调用，但不能由中断服务程序调用。
- 以 isr_开头的函数可以由中断服务程序调用，但不能由任务调用。

一、irs_send_signal

概要 #include<rtx51tny.h>

char isr_send_signal(unsigned char task_id); /*信号发往的任务*/

描述 isr_send_signal 函数给任务 task_id 发送一个信号。如果指定的任务正在等待一个信号，则该函数使该任务就绪，但不启动它，信号存储在任务的信号标志中。

附注

- 该函数是 RTX51 Tiny 实时操作系统的一部分，仅包含于 PK51 中。
- 该函数仅被中断函数调用。

返回值 成功调用后返回 0，如果指定任务不存在，则返回-1。

参阅 os_clear_signal,os_send_signal,os_wait

例子

```
#include<rtx51tny.h>
void tst_isr_send_signal(void) interrupt 2
{
  isr_send_signal(8); /*给任务 8 发信号*/
}
```

二、irs_set_ready

概要　　#include< rtx51tny.h>

char isr_set_ready{ unsigned char task_id};/*使就绪的任务*/

描述　将由 task_id 指定的任务置为就绪态。

附注

- 该函数是 RTX51 Tiny 的一部分，包含在 PK51 中。
- 该函数仅用于中断函数。

返回值　　无

例子　　#include< rtx51tny.h>

void tst_isr_set_ready(void)interrupt 2

{ isr_set_ready(1);/*置位任务 1 的就绪标志*/

}

三、os_clear_signal

概要　#include< rtx51tny.h>

char os_clesr_signal(unsigned cahr task_id);/*清除信号的任务*/

描述　清除由 task_id 指定的任务信号标志。

附注　该函数是 RTX51 Tiny 的一部分，包含在 PK51 中。

返回值　信号成功清除后返回 0，指定的任务不存在时返回-1。

参阅　isr_send_signal,os_send_signal,os_wait

例子　#include< rtx51tny.h>

void tst_os_clsar_siganl(void)_task_8

{

…

os_clear_signal(5);　　/*清除任务 5 的信号标志*/

…

}

四、os_create_task

概要 #include<rtx51tny.h>

char os_create_task(unsigned char task_id);/*要启动的任务 ID*/

描述 启动任务 task_id，该任务被标记为就绪，并在下一个时间点开始执行。

附注： 该函数是包含在 PK51 中的 RTX51 Tiny 的组成部分。

返回值 任务成功启动后返回 0，如果任务不能启动或任务已在运行，或没有以 task_id 定义的任务，返回-1。

参阅 os_delete_task

例子 #include< rtx51tny.h>

#include<stdio.h> /*用于 printf*/

```
void new_task(void)_task_2
 {…}
 void tst_os_create_task(void)_task_0
 {
       …
  if(os_create_task(2))
{
    printf("couldn't start task2"n");
         }
            …
         }
```

五、os_delete_task

概要 #include<rtx51tny.h>

char os_delete_task(unsigned char task_id);/*要删除的任务*/

描述 函数将以 task_id 指定的任务停止，并从任务列表中将其删除。

附注 该函数是包含在 PK51 中的 RTX51 Tiny 的组成部分。

返回值 任务成功停止并删除后返回 0。指定任务不存在或未启动时返回-1。

附注 如果任务删除自己，将立即发生任务切换。

参阅 os_create_task

例子

```
#include<rtx51tny.h>
#include<stdio.h>
 void tst_os_delete_task(void)_task_0
 {
    …
  if(os_delete_task(2))
  {
    printf("couldn't stop task2"n");
  }
    …
 }
```

六、os_reset_interval

概要 #include<rtx51tny.h>

void os_reset_interval(unsigned char ticks); /*滴答数*/

描述 用于纠正由于 os_wait 函数同时等待 K_IVL 和 K_SIG 事件而产生的时间问题，在这种情况下，如果一个信号事件（K_SIG）引起 os_wait 退出，时间间隔定时器并不调整，这样，会导致后续的 os_wait 调用（等待一个时间间隔）延迟的不是预期的时间周期。允许你将时间间隔定时器复位，这样，后续对 os_wait 的调用就会按预期的操作进行。

附注 该函数是包含在 PK51 中的 RTX51 Tiny 的组成部分。

返回值 无

例子

```
#include<rtx51tny.h>
 void task_func(void)_task_4
                {
                  …
                  switch(os_wait2(KSIG|K_IVL,100))
                    {
                       case   TMO_EVENT:
                  /*发生了超时，不需要 Os_reset_interval*/
                              break;
                       case    SIG_EVCENT:
```

```
        /*收到信号，需要 Os_reset_interval*/
            os_reset_interval(100);
        /*依信号执行的其它操作*/
            break;
       }
   …
  }
```

七、os_running_task_id

概要 #include<rtx51tny.h>

char os_running_task_id(void);

描述 函数确认当前正在执行的任务的任务 ID。

附注： 该函数是包含在 PK51 中的 RTX51 Tiny 的组成部分。

返回值 返回当前正在执行的任务的任务号，该值为 0~15 之间的一个数。

例子 #include<rtx51tny.h>

```
void tst_os_running_task(void)_task_3
{
  unsigned char tid;
  tid=os_running_task_id( ); /*tid=3*/
 }
```

八、os_send_signal

概要 #include<rtx51tny.h>

char os_send_signal(char task_id);/*信号发往的任务*/

描述 函数向任务 task_id 发送一个信号。如果指定的任务已经在等待一个信号，则该函数使任务准备执行但不启动它。信号存储在任务的信号标志中。

附注 该函数是包含在 PK51 中的 RTX51 Tiny 的组成部分。

返回值 成功调用后返回 0，指定任务不存在时返回-1。

参阅 isr_send_signal,os_clear_signal,os_wait

```
#include<rtx51tny.h>
void signal_func(void)_task_2
{
  …
  os_send_signal(8);     /*向 8 号任务发信号*/
  …
 }
void tst_os_send_signal(void)_task_8
{
  …
  os_send_signal(2);    /*向 2 号任务发信号*/
  …
 }
```

九、 os_set_ready

概要 #include<rtx51tny.h>

char os_set_ready(unsigned char task_id);/*使就绪的任务*/

描述 将以 task_id 指定的任务置为就绪状态。

附注 该函数是包含在 PK51 中的 RTX51 Tiny 的组成部分。

返回值 无

例子 #include<rtx51tny.h>

```
void ready_func(void)_task_2
{
   …
   os_set_ready(1);     /*置位任务 1 的就绪标志*/
   …
}
```

十、os_switch_task

概要　#include<rtx51tny.h>

char os_switch_task(void);

描述　该函数允许一个任务停止执行，并运行另一个任务。如果调用 os_switch_task 的任务是唯一的就绪任务，它将立即恢复运行。

附注　该函数是包含在 PK51 中的 RTX51 Tiny 的组成部分。

返回值 无

例子　#include<rtx51tny.h>

```
#include<stdio>
void long_job(void)_task_1
{
   float f1,f2;
   f1=0.0;
   while(1)
    {
      f2=log(f1);
      f1+=0.0001;
      os_switch_task();  /*运行其它任务*/
     }
 }
```

十一、os_wait

概要　#include<rtx51tny.h>

```
char os_wait(
 unsigned char event_sel,   /*要等待的事件*/
 unsigned char ticks,     /*要等待的滴答数*/
 unsigned int   dammy);  /*无用参数*/
```

描述　该函数挂起当前任务，并等待一个或几个事件，如时间间隔，超时，或从其它任务和中断发来的信号。参数 event_set 指定要等待的事件，可以是下表中常数的一些组合。

事 件	描　　述
K_IVL	等待滴答值为单位的时间间隔
K_SIG	等待一个信号
K_TMO	等待一个以滴答值为单位的超时

事件可以用竖线符（“|”）进行逻辑或。例如，K_TMO1K_SIG 指定任务等待一个超时或者一个信号。

ticks 参数指定要等待的时间间隔事件(K_IVL)或超时事件(K_TMO)的定时器滴答数。参数是为了提供与兼容性而设置的，在中并不使用。

附注

- 该函数是包含在 PK 中的 RTX51 Tiny 的组成部分。
- 请参阅事件一节获得关于 K_IVL,K_SIG,K_TMO 的更多信息。

返回值 当有一个指定的事件发生时，任务进入就绪态。任务恢复执行时，下表列出的由返回的常数指出使任务重新启动的事件。可能的返回值有：

返 回 值	描　　述
RDY_EVENT	表示任务的就绪标志是被或函数置位的。
SIG_EVENT	收到一个信号
TMO_EVENT	超时完成，或时间间隔到
NOT_OK	参数的值无效

参阅

isr_send_signal,isr_set_ready,os_clear_signal,os_reset_interval,

os_send_signal,os_set_ready,os_wait1,os_wait2

例子 #include<rtx51tny.h>

```
#include<stdio.h>
void tst_os_wait(void)_task_9
{
  while(1)
   {
    char event;
```

```
event=os_wait(K_SIG|K_TMO,50.0);
switch(event)
{
  default:          /*从不发生，该情况*/
  break;
  case   TMO_EVENT;/*超时*/
  break;            /*50 次滴答超时*/
  case   SIG_EVENT; /*收到信号*/
  break;
 }
}
}
```

十二、os_wait1

概要　#include<rtx51tny.h>

char os_wait1(unsigned char event_sel);/*要等待的事件*/

描述　该函数挂起当前的任务等待一个事件发生。os_wait1 是 os_wait 的一个子集，它不支持 os_wait 提供的全部事件。参数 event_sel 指定要等待的事件，该函数只能是 K_SIG。

附注

- 该函数是包含于 PK51 中的 RTX51Tiny 的组成部分。
- 参见事件一节获得 K_IVL，K_SIG 和 K_TMO 的更多信息。

返回值　当指定的事件发生，任务进入就绪态。任务恢复运行时，os_wait1

返回的值表明启动任务的事件，返回值见下面的常数列表：

返　回　值	
RDY_EVENT	任务的就绪标志位是被 os_set_ready 或 isr_set_ready 置位的
SIG_EVENT	收到一个信号
NOT_OK	Event_sel 参数的值无效

例子　见 os_wait

十三、os_wait2

概要　　#include<rtx51tny.h>

char os_wait2(unsigned char event_sel,　　/*要等待的事件*/

unsigned char ticks);　　/*要等待的滴答数*/

描述　　函数挂起当前任务等待一个或几个事件发生，如时间间隔，超时或一个从其它任务或中断来的信号。参数 event_sel 指定的事件可以是下列常数的组合：

Event	描　　述
K_IVL	等待以滴答数为单位的时间间隔
K_SIG	等待一个信号
K_TMO	等待以滴答数为单位的超时

事件可以用“|”进行逻辑或。如 K_TMO|K_SIG 表示任务等待一个超时或一个信号。

参数 ticks 指定等待时间间隔(K_IVL)或超时(K_TMO)事件时的滴答数。

附注

- 该函数是包含于 PK51 中的 RTX51Tiny 的组成部分。
- 参见事件一节获得更多关于 K_IVL, K_TMO,和 K_SIG 的信息。

返回值 当一个或几个事件产生时,任务进入就绪态.任务恢复执行时, os_wait2 的返回值见下面的常数列表：

返 回 值	描　　述
RDY_EVENT	任务的就绪标志是被 os_set_ready 或 isr_set_ready 函数置位的
SIG_EVENT	收到一个信号
TMO_EVENT	返回时完成,或时间间隔到达
NOT_OK	参数 event_sel 的值无效

例子　　见 os_wait。