中图分类号:TP311.54　　文献标识码:B　　文章编号:1009 - 2552(2005)12 - 0085 - 03

RTX51 TINY中任务的建立

付　虹，牛国成，胡冬梅，苑广军

(长春工业大学电气与电子工程学院，长春 130012)

摘　要：简要介绍了 RTX51 TINY的基本特性。详细地分析了创建任务时堆栈的配置和状态字结构，并给出创建任务的主要代码流程图。

关键词：RTX51 Tiny；实时操作系统；堆栈配置

Creating tasks on RTX51 TINY

FU Hong，NIU Guo-cheng，HU Dong-mei，YUAN Guang-jun

(**Electric and Electronic Engineering College**，**Changchun University of technology**，**Changchun** 130012，**China**)

Abstract：Essentialities of RTX51 tiny are introduced simply，stack configuration and the construction of task-state are analysed in detail when tasks are created，and the flow chart of primary codes is given in this paper.

Key words：RTX51 tiny；real - time operating system；stack configuration

1　RTX51 TINY简介

RTX51是KEIL公司开发的用于8051系列单片机的多任务实时操作系统，利用它可以简化具有实时性要求的多任务复杂软件的设计。RTX51有两个版本：RTX51 Tiny和RTX51 Full[1]。

RTX51 Tiny是RTX51 Full的子集。RTX51 Tiny自身仅占用900字节左右的程序存储空间，可以很容易地运行在没有外部扩展存储器的8051单片机系统上。但是，使用RTX51 Tiny的程序可以访问外部存储器[1]。

通过仿真得出：当只有任务0，相当于只有初始化程序时，此时仅占2单元；当建立最简单的任务(无等待、等待时间到、中断等事件)，即只是简单的任务循环时，任务占用638 + 9(n - 1) (n = 1 …15)单元，当在任务0中用os_creat建立任务而无此任务时则只占用634 + 9(n - 1) (n = 1 …15)单元。

RTX51 Tiny最多可定义16个任务，所有任务可同时激活。它仅支持按时间片循环任务调度和用信号进行任务切换(非抢占式)，可以并行地利用中断。具有以下等待操作：超时、另一个任务或中断的信号。

2　创建任务的函数[1]

函数原型：char os_creat_task (unsigned char task_id)；

参数：task_id为要开始的任务数，必须使用与任务描述一致的数(0～15)。

返回值：0表示信号成功开始，- 1表示指定的任务不存在或不能开始指定的任务。

功能：os_creat_task()开始一个由task_id指定的任务，任务被标记为READY状态，并根据任务切换规则执行任务。

3　状态字结构

每个RTX51 TINY的任务都必须处在下列状态之一[1]：

RUNNING：任务正在运行中，每次只能有一个任务处于运行状态。

READY：任务正在等待运行，当前运行的任务完成后，RTX51 TINY开始运行处于等待中的任务。

WAITING：任务正在等待一个事件，如果发生的事件是任务所等待的，则任务进入“READY”状态。

DELETED：任务不开始执行。

TIMEOUT：任务被一个循环“时间到”所中断，该状态与“READY”状态相似，但由于内部操作过程使一个循环任务切换而被冠以标记。

收稿日期：2005 - 10 - 27

作者简介：付虹(1963 -)，女，1990年5月毕业于长春光机所，获硕士学位。现主要从事信号检测与处理及嵌入式系统的开发等方面的研究工作。

由于任务有以上几种状态,所以要标记任务状态必须设置状态字,任务状态字结构如下定义:

任务状态.0 = 等待信号

任务状态.1 = 等待时间到

任务状态.2 = 信号标志

任务状态.3 = 时间到标志

任务状态.4 = 任务就绪(等待运行)

任务状态.5 = 任务激活(os_create 使能位)

任务状态.6 = 轮转时间到

任务状态.7 = 运行标志

4 任务建立过程及流程

在 RTX51 Tiny 系统中没有主程序,但在编译时系统默认任务 0 为主程序。可以在任务 0 内建立全部任务,也可以在任务 0 中建立部分任务,而在已建立的任务中再建立其余任务,即任务可以在多任务调度开始前建立,也可以在其它任务执行过程中建立。在开始多任务调度前,用户必须建立至少一个任务,并将其放入任务表中。在任务建立的过程中,主要对任务的堆栈区、堆栈指针、任务所处的状态和任务的入口地址分配堆栈。任务创建过程主要是对堆栈的配置。

任务建立的形式有如下两种:

第一种形式:

```
void init (void) _task_ 0
{
    os_create_task (1);
    os_create_task (2);
        ……
    os_create_task (n);
}
```

第二种形式:

```
void init (void) _task_ 0
{
    os_create_task (1);
        ……
    os_create_task (3)
    os_create_task (n);
}
void scanKey (void)_task_ 1
{
    os_create_task (2);
}
```

4.1 任务建立的堆栈分配

RTX51 Tiny 用 8051 单片机的内部存储器为每个任务设定堆栈,并且为每个任务都保留一个单独的堆栈区,全部堆栈管理都在 IDATA 空间进行。为了给当前正运行的任务分配尽可能大的栈区,所以各任务所用的堆栈位置是动态的,并用 STKP[task - id]来记录各任务的堆栈栈底位置。

RTX51 Tiny 对内存分配大致分为如下三个部分:第一部分是当前任务寄存器(寄存器组 0),位于 00H 至 07H 单元。第二部分是当前任务存储区,位于 08H 至 0DH 单元。其中 08H 和 09H 单元是暂存单元;0AH 单元用来保护累加器 A;0BH 单元保护 PSW;0CH 是当前任务号(task ID);0DH 是时间片(time slice)。它们在中断程序里是以寄存器组 1 的方式来寻址的。第三部分是从 0EH 单元开始依任务数长度不同而建立的两张表:其一是任务堆栈表,其二是任务状态表。内存的分配主要是对第三部分的定义。

4.1.1 任务堆栈表的建立

任务堆栈表是对任务建立时任务的堆栈区和任务入口进行定义。假设任务的最大栈顶为 FFH,则初始化时,任务 1.2.3 …n 的栈顶分别为 FF。对于第一种形式,n 个任务创建后,任务栈顶 FF 开始向下建立任务入口,此入口占自由堆栈空间的 2 个字节。栈顶顺序为 FF - 2n,FF - 2n + 2,……FB,FD(n = 1,2 …15)。对于第二种形式,任务建立过程中也满足相邻两个任务栈顶之间相差 2 个单元的规则。但不同的是,建立任务 1 时栈顶为 FD,其余任务栈顶仍为 FF,当建立任务 3 时,应满足未建立的任务的栈顶与与它相邻的未建立的任务的栈顶相同的规则,所以,未建立的任务 2 的栈顶应与任务 3 的栈顶相同,任务 3 与任务 1 栈顶相差 2 单元,即栈顶分别为 FF - 2n + 2,FF - 2n + 4,FF - 2n + 4、FF - 2n + 6 …FD(n = 1,2 …15),当建立任务 2 后则遵循第一种形式时的规则,任务栈顶依次为 XX,FF - 2n,FF - 2n - 2 …FB,FD(n = 2,3 …15),由于任务 1 已执行,所以任务 1 的栈顶应据具体情况而定。

4.1.2 任务状态表

任务状态表由任务定时节拍计数器(timer)和任务状态寄存器(task state)组成,以 2 字节为一单位,前者在每一次定时中断后自减一次,后者用其各个

位表示任务所处的状态,20 单元的 D0 位作为禁止任务切换位。

随着建立任务数的不同,存储器分配也不同,具体分配情况可以分为以下几个方面:

(1) 建立任务数为 $1 \leq n_1 \leq 6$ 时,当前任务堆栈表和任务状态表均按照上述规律进行管理;

(2) 建立任务数为 $7 \leq n_2 \leq 9$ 时,以上存储单元分配将有所变化。因为 20 单元的 20.0 位被 RTX51 Tiny 定义为禁止任务切换位,故当前任务堆栈区将存放在从 21H 开始的各单元中,任务状态表仍然从 0EH 开始存放。

(3) 建立任务数为 $10 \leq n_3 \leq 16$ 时,由于 20 之前的单元已不够分配,为了给任务状态表留有足够的空间,所以,就将任务状态表存放在原来的任务堆栈表处,即从 21H 开始,而原来任务堆栈区则存放在从 0EH 开始的单元了。

4.2 任务创建主要代码流程图

任务建立的具体流程图过程如图 1 所示。

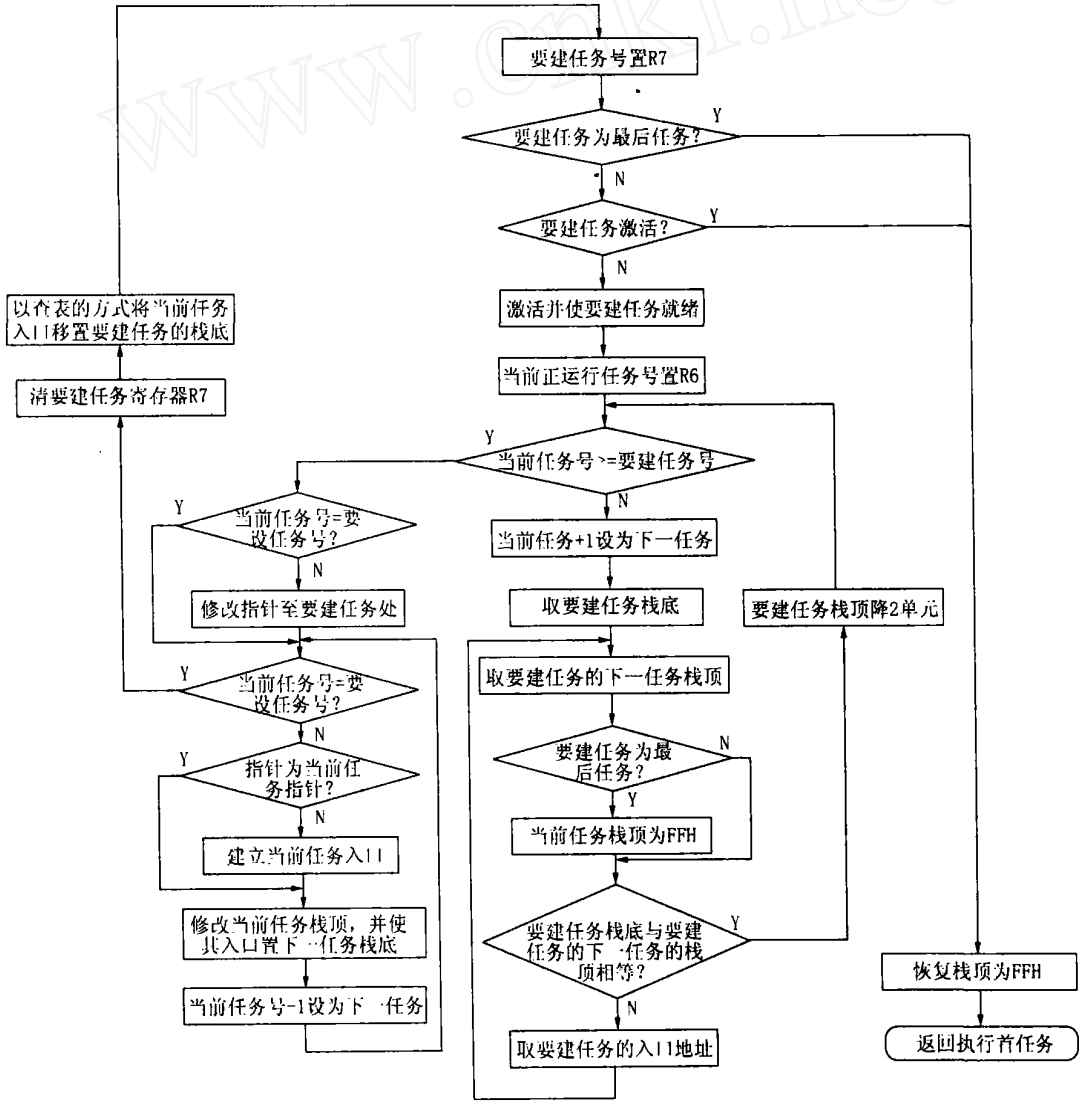

图 1 任务创建代码流程图

5 结束语

以上分析可以看出,RTX51 Tiny 任务建立的过程中,堆栈的分配是动态的,这样节约了存储器资源,非常适用于没有外部扩展器的单片机上。而且,设计思想简单,易于理解和移植。

参 考 文 献:

[1] 徐爱钧,彭秀华. 单片机高级语言 C51 Windows 环境编成与应用[M]. 北京:电子工业出版社,2001.

[2] KEIL SOFTWARE INC. Real - time multitasking executives for the 8051 and MCS 251 Microcontrollers user's guide[M]. 1997.

责任编辑:张荣香