



# 第17章 RTX51操作系统原理及实现

何宾  
2018.03



# 本章主要内容

- 操作系统的必要性
- 操作系统基本知识
- RTX51操作系统的任务
- RTX51操作系统内核函数
- RTX51操作系统实现

# 操作系统的必要性

## --单任务程序

在不使用操作系统的传统的单片机中，常使用单任务的程序或者轮询的程序。

- 一个标准的C程序用main函数启动执行。在嵌入式应用中，main通常作为一个无限循环，被认为是一个单任务，这个任务连续的运行。

# 操作系统的必要性

## --单任务程序

### 【例】 单任务程序C语言描述的例子

```
int counter;
```

```
void main (void)
```

```
{
```

```
    counter = 0;
```

```
    while (1)
```

```
        counter++;
```

```
}
```

```
//无限循环
```

```
//递增计数器
```

# 操作系统的必要性

## --轮询程序

**不用实时操作系统RTOS，解决单任务程序的一个方法就是将需要CPU执行的一些程序编写称为子程序，然后用一个轮询预安排的多任务机制，实现一个更复杂的C程序。**

■ **在这个机制中，任务或者函数在一个无限循环中被重复的调用。**

# 操作系统的必要性

## --轮询程序

### 【例】 轮询程序 C语言描述的例子

```
int counter;  
void main (void)  
{  
    counter = 0;  
    while (1)                                //无限循环  
    {  
        check_serial_io ();  
        process_serial_cmds ();              //处理串行输入  
        check_kbd_io ();  
        process_kbd_cmds ();                 //处理键盘输入  
        adjust_ctrlr_parms ();               //调整控制器  
        counter++;  
    }  
}
```

//递增计数器

# 操作系统基本知识

**操作系统是管理和控制计算机硬件与软件资源的计算机程序，是直接运行在计算机硬件上的最基本的系统软件，任何其他软件都必须在操作系统的支持下才能运行。**

■ **操作系统在硬件系统上运行，它常驻内存内，并提供给上层两种接口：操作接口和编程接口。**

- **操作接口由一系列操作命令组成，用户通过操作接口可以方便地使用计算机。**
- **编程接口由一系列的系统调用组成各种程序可以使用这些系统调用让操作系统为其服务，并通过操作系统来使用硬件和软件资源。**

# 操作系统基本知识

## --操作系统的作用

**操作系统的作用主要体现在以下两方面：**

- **屏蔽硬件物理特性和操作细节，为用户使用计算机提供了便利。**
- **有效管理系统资源，提高系统资源使用效率。**



# 操作系统基本知识

## --操作系统的功能

**操作系统位于底层硬件与用户之间，是两者沟通的桥梁。**

- 用户可以通过操作系统的用户界面，输入命令。
- 操作系统则对命令进行解释，驱动硬件设备，实现用户要求。

# 操作系统基本知识

## --操作系统的功能

■ 以现代观点而言，一个完整的OS应该提供以下的功能：

- 资源管理
- 内存管理
- 程序控制
- 虚拟内存
- 人机交互
- 用户接口
- 进程管理
- 用户界面

# 操作系统基本知识

## --资源管理

- 系统的设备资源和信息资源都是操作系统根据用户需求按一定的策略来进行分配和调度的。
- 处理器管理或称处理器调度，是操作系统资源管理功能的另一个重要内容。操作系统的设备管理功能主要是分配和回收外部设备以及控制外部设备按用户程序的要求进行操作等。
- 信息管理是操作系统的一个重要的功能，主要是向用户提供一个文件系统。

# 操作系统基本知识

## --程序控制

**操作系统控制用户程序的执行主要有以下一些内容：**

- 调入相应的编译程序，将用某种程序设计语言编写的源程序编译成计算机可执行的目标程序；
- 分配内存存储等资源将程序调入内存并启动；
- 按用户指定的要求处理执行中出现的各种事件以及与操作员联系请示有关意外事件的处理等。

# 操作系统基本知识

## --人机交互

操作系统的人机交互功能是决定计算机系统友好性的一个重要因素。

- 人机交互功能主要靠可输入输出的外部设备和相应的软件来完成。
  - 可供人机交互使用的设备主要有键盘显示、鼠标、各种模式识别设备等。
  - 与这些设备相应的软件就是操作系统提供人机交互功能的部分。
- 人机交互部分的主要作用是控制有关设备的运行和理解并执行通过人机交互设备传来的有关的各种命令和要求。

# 操作系统基本知识

## --进程管理

**不管是常驻程序或者应用程序，他们都是以进程为标准的执行单位。**

- **进程就是当前正在运行的程序。**
- **进程管理指的是操作系统管理多个进程的准备、运行、挂起和退出。**
- **进程管理通常使用分时复用的调度机制，大部分的操作系统可以通过为不同进程指定不同的优先级，从而改变为这些进程所分配的时间片。**
- **在进程管理中，优先调度优先级高的进程。**

# 操作系统基本知识

## --内存管理

**操作系统的存储器管理功能提供：**

- 查找可用的存储空间；
- 配置与释放存储空间；
- 交换内存和外存的内容；
- 提供存储器访问的权限等功能。

# 操作系统基本知识

## --虚拟内存

**虚拟内存是计算机系统内存管理的一种技术。**

- 它使得应用程序认为它拥有连续的可用的内存（一个连续完整的地址空间）。
- 而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要进行数据交换。



# 操作系统基本知识

## --用户接口

**用户接口包括作业一级接口和程序一级接口。**

- **作业一级接口为了便于用户直接或间接地控制自己的作业而设置。**
- **它通常包括联机用户接口与脱机用户接口。**
- **程序一级接口是为用户程序在执行中访问系统资源而设置的，通常由一组系统调用组成。**

# 操作系统基本知识

## --用户界面

**用户界面（User Interface, UI）是系统和用户之间进行交互和信息交换的媒介，它实现信息的内部形式与人类可以接受形式之间的转换。**

- **目的在使得用户能够方便有效率地去操作硬件以达成双向之交互，完成所希望借助硬件完成之工作。**
- **用户界面定义广泛，包含了人机交互与图形用户接口，凡参与人类与机械的信息交流的领域都存在着用户界面。**

# RTX51操作系统的任务

## --定义任务

实时或者多任务应用是由一个或多个执行指定操作的任务所构成的。RTX51 Tiny允许最多16个任务。

- 任务是简单的C函数，返回类型为void，有一个void参数列表。使用\_task\_函数属性声明。格式如下：

```
void func(void) _task_ num
```

- func为任务的函数名；
- \_task\_是定义任务关键字；
- num是任务ID号，取值从0~15，每一个任务必须有一个唯一的任务号。

# RTX51操作系统的任务

## --定义任务

### 【例】 定义任务 C语言描述的例子

```
void job0 (void) _task_ 0
{
    while(1)
    {
        counter0++;           //计数器递增
    }
}
```

# RTX51操作系统的任务

## --管理任务

**每个在RTX51 Tiny中定义的任务，都应该在下面状态中的其中某个状态。**

不同状态的描述

状态	描述
RUNNING	在RUNING状态时，正在执行当前任务。在一个时刻，只允许执行一个任务
READY	在READY状态，等待执行任务。当处理完当前运行的任务后，RTX51 Tiny启动下一个准备好的任务
WAITING	在WAITING状态，任务等待一个事件。如果发生一个事件，任务则进入READY状态
DELETED	在DELETED状态，没有启动任务。
TIME-OUT	在TIME-OUT状态，任务被轮询超时打断。这个状态等同于READY状态。

# RTX51操作系统的任务 --切换任务

**RTX51 Tiny执行轮询多任务调度，这样允许模拟并行执行多个无限循环或者任务。**

- **任务不是并发执行的，而是按时间片执行的。**
- **可用的CPU时间被分成时间片，RTX51 Tiny为每个任务分配一个时间片。**
- **每个任务允许执行预先确定的时间长度。然后，RTX51 Tiny切换到其他准备运行的任务，然后这个任务执行一段时间。**
- **时间片的长度使用变量TIMESHARING定义。**

# RTX51操作系统的任务

## --切换任务

RTX51 Tiny中，负责分配处理器给一个任务的那部分称为调度器。根据下面的规则，RTX51 Tiny调度器定义所运行的任务：

- 如果发生下面的情况，则打断当前正在运行的任务：
  - 任务调用os\_wait函数，并且没有产生指定的事件；
  - 执行任务的时间大于定义的轮询超时时间；
- 如果发生下面的情况，则启动其他任务：
  - 没有运行其他任务；
  - 将要启动的任务处于READY或者TIME\_OUT状态；

# RTX51操作系统内核函数

**注：在调用这些内核函数时，必须包含头文件rtx51tny.h。**

## ■ `char isr_send_signal(unsigned char task_id)`

**功能：**该函数发送信号到task\_id所确定的任务。

- 如果指定的任务已经在等待信号，则该函数调用将准备用于执行的任  
务。
- 否则，保存信号到任务的信号标志内。该函数只能被中断函数所调用。

**返回值：**0，表示成功；-1，表示任务不存在；



# RTX51操作系统内核函数

■ **char os\_clear\_signal(unsigned char task\_id)**

功能：清除task\_id指定任务的信号标志；

返回：0，表示成功清除信号标志； - 1，表示任务不存在

# RTX51操作系统内核函数

## 【例】 调用isr\_send\_signal()函数的例子

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

```
void tst_isr_send_signal (void) interrupt 2
```

```
{
```

```
    isr_send_signal (8);           //向任务8发送信号
```

```
}
```

# RTX51操作系统内核函数

## 【例】 调用os\_clear\_signal()函数的例子

```
#include <rtx51tny.h>

void tst_os_clear_signal (void) _task_ 8
{
    ....
    os_clear_signal (5);    //清除任务5中的信号标志
    ....
}
```

# RTX51操作系统内核函数

## ■ `char os_create_task(unsigned char task_id)`

功能：启动由`task_id`指定的任务。将该任务标记为准备状态，并且根据RTX51 Tiny指定的规则执行该任务。

返回：0，表示成功启动任务；-1表示没有启动任务，或者不存在`task_id`定义的任务。

# RTX51操作系统内核函数

## ■ `char os_delete_task(unsigned char task_id)`

功能：停止`task_id`指定的任务，将由`task_id`指定的任务从任务列表中删除。

返回：0，表示任务成功停止和删除； - 1，表示指定的任务不存在或者`task_id`定义的任务没有启动。

# RTX51操作系统内核函数

## 【例】调用os\_create\_task()函数的例子

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

```
#include <stdio.h>
```

```
void new_task (void) _task_ 2
```

```
{
```

```
....
```

```
}
```

```
void tst_os_create_task (void) _task_ 0
```

```
{
```

```
....
```

```
if (os_create_task (2))
```

```
    printf ("Couldn't start task 2\n");
```

```
.....
```

```
}
```

# RTX51操作系统内核函数

## 【例】 调用os\_delete\_task()函数的例子

```
#include <rtx51tny.h>
#include <stdio.h>
void tst_os_delete_task (void) _task_ 0
{
    ....
    if (os_delete_task (2))
        printf ("Couldn't start task 2\n");
    .....
}
```

# RTX51操作系统内核函数

## ■ `char os_running_task_id(void)`

功能：确定当前运行任务的id。

返回：当前运行任务的任务ID号，值的范围为0~15。



# RTX51操作系统内核函数

■ **char os\_send\_signal(unsigned char task\_id)**

功能：发送信号到task\_id任务。

- 如果指定的任务已经在等待信号，则该函数调用将准备任务用于执行。
- 否则，保存信号到任务的信号标志内。

返回：0，表示成功； - 1，表示任务不存在。

# RTX51操作系统内核函数

【例】 调用os\_running\_task\_id()函数的例子

```
#include <rtx51tny.h>

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

# RTX51操作系统内核函数

【例】调用os\_send\_signal ()函数的例子

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

```
void signal_func (void) _task_ 2
```

```
{  
    .....
```

```
    os_send_signal (8);           //向任务8发送信号
```

```
    .....
```

```
}
```

```
void tst_os_send_signal (void) _task_ 8
```

```
{
```

```
    .....
```

```
    os_send_signal (2);           //向任务2发送信号
```

```
    .....
```

```
}
```

# RTX51操作系统内核函数

■ **char os\_wait(unsigned char event\_sel, unsigned char ticks, unsigned int dummy)**

功能：os\_wait函数停止当前的任务，等待一个或多个事件，比如：来自时间间隔，来自一个超时，或者来自其他任务或者中断。

■ **event\_sel**

该参数指定了时间或者等待事件，可以是下面常数的任何的组合：

- **K\_IVL**：等待一个定时器滴答间隔；
- **K\_SIG**：等待一个信号；
- **K\_TMO**：等待一个超时；

# RTX51操作系统内核函数

## ■ tick

- 该参数指定用于等待一个间隔事件（K\_IVL）或者超时事件（K\_TMO）的定时器滴答的数目。

## ■ dummy

提供和RTX51的兼容性，RTX51 Tiny不使用。

返回：指定事件发生时，使能任务用于执行。恢复执行。可能的返回值：

- SIG\_EVENT：接收到一个信号；
- TMO\_EVENT：完成超时，或者间隔过期；
- NOT\_OK：event\_sel参数无效；

# RTX51操作系统内核函数

## ■ char os\_wait1(unsigned char event\_sel)

功能：os\_wait1停止当前的任务，等待发生一个事件。os\_wait1函数是os\_wait函数的子集，不允许os\_wait所提供的的所有的事件。

其中：

□ event\_sel 指定等待的事件，只能有K\_SIG，即等待信号。

返回：当信号事件发生时，使能任务用于执行。恢复执行。可能的返回值：SIG\_EVENT或者NOT\_OK。

# RTX51操作系统内核函数

## 【例】调用os\_wait()函数的例子

```
#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: break; //空操作
            case TMO_EVENT: break; //超时
            case SIG_EVENT: os_reset_interval (100); //收到信号, 必须使用
            break; // os_reset_interval调整延迟
        }
    }
}
```

# RTX51操作系统内核函数

## ■ `char os_wait2(unsigned char event_sel, unsigned char ticks)`

功能：os\_wait函数停止当前的任务，等待一个或多个事件，比如：来自时间间隔，来自一个超时，或者来自其他任务或者中断。

其中：

- event\_sel参数指定了时间或者等待事件，能是下面常数的任何的组合：
  - K\_IVL：等待一个定时器滴答间隔；
  - K\_SIG：等待一个信号；
  - K\_TMO：等待一个超时；



# RTX51操作系统内核函数

- tick: 参数指定用于等待一个间隔事件 (K\_IVL) 或者超时事件 (K\_TMO) 的定时器滴答的数目。

返回: 指定事件发生时, 使能任务用于执行。恢复执行。可能的返回值:

- SIG\_EVENT: 接收到一个信号;
- TMO\_EVENT: 完成超时, 或者间隔过期;
- NOT\_OK: event\_sel参数无效;

# RTX51操作系统实现1

在这个例子中所创建的任务是实现简单的计数器循环。

- RTX51启动执行名字为job0的任务0。

- 这个函数添加另一个名字为job1的任务。

- 当job0执行一段时间后，RTX51切换到job1。

- 在job1执行一段时间，RTX51切换回job0。这个过程无限重复。

# RTX51操作系统实现1

## 【例】使用RTX51 Tiny内核函数调用实现轮询调度C语言描述

```
#include <rtx51tny.h>

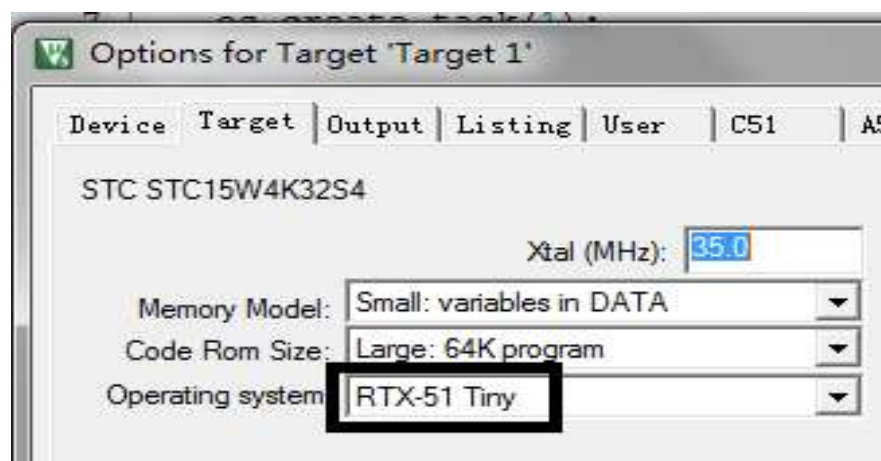
unsigned char counter0;      //定义无符号char类型变量counter0
unsigned char counter1;      //定义无符号char类型变量counter1
void job0 (void) _task_ 0    //定义任务0
{
    os_create_task (1);       //创建一个任务
    while (1)                 //无限循环
    {
        counter0++;           //更新计数器
    }
}
```

# RTX51操作系统实现1

```
void job1 (void) _task_ 1    //定义任务1
{
    while (1)//无限循环
    {
        counter1++;    //更新计数器
    }
}
```

# RTX51操作系统实现1

注：在使用RTX51操作系统时，需要在Options for Target 'Target 1' 对话框界面中，选择Target标签。在该标签窗口界面中，在Operating system右侧的下拉框中，选择RTX-51 Tiny选项。



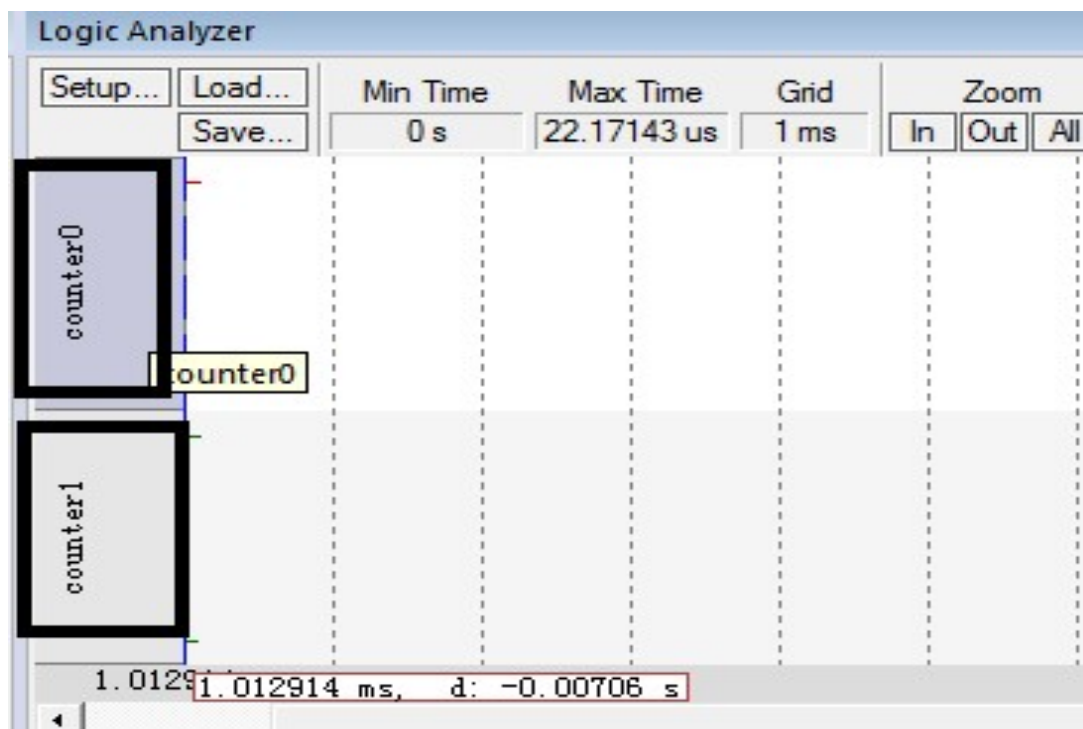
# RTX51操作系统实现1

下面通过Keil  $\mu$ Vision调试器，说明RTX51的运行机制，步骤主要包括：

- 在例子17-1目录下，打开该设计。
- 在Keil  $\mu$ Vision当前设计主界面主菜单下，选择Debug->Start/Stop Debug Session，进入调试器模式。
- 在当前调试器主界面主菜单下，选择View->Logic Analyzer选项
- 在当前调试器主界面右侧上方，出现Logic Analyzer界面。



# RTX51操作系统实现1

- 在该界面中，添加counter0和counter1两个变量



# RTX51操作系统实现1

- 鼠标右键分别单击counter0和counter1，出现浮动菜单。在浮动菜单内选择state选项。
- 在当前调试器主界面主菜单下，选择View->Watch Windows->Watch 1。
- 在当前调试器主界面右下方出现Watch1窗口界面。在该界面中，添加counter0和counter1两个变量。

Watch 1		
Name	Value	Type
 counter0	0	uchar
 counter1	0	uchar
<Enter expression>		






# RTX51操作系统实现1

- 按F5按键或者在当前调试主界面主菜单下，选择Debug->Run，运行该程序。
- 观察Logic Analyzer窗口。通过观察counter0和counter1，很明显，两个任务：任务0和任务1在分时运行。



# RTX51操作系统实现1

- 观察Watch 1窗口，可以看到counter0和counter1两个变量的值在交替变化。

Watch 1		
Name	Value	Type
 counter0	93 'I'	uchar
 counter1	135 '?'	uchar
 <Enter expression>		

## RTX51操作系统实现2

在这个例子中，job0使能job1。但是现在，当递增counter0后，job0调用os\_wait函数暂停3个时钟滴答。

- 在这个时间，RTX51切换到下一个任务job1。
- 当job1递增counter1后，它也调用os\_wait暂停5个时钟滴答。
  - 现在，RTX51没有其他任务需要执行，它进入空闲状态3个时钟滴答，然后继续执行job0。

# RTX51操作系统实现2

【例】 使用RTX51使用os\_wait函数延迟执行C语言描述的例子

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

```
unsigned char counter0;
```

```
unsigned char counter1;
```

```
void job0 (void) _task_ 0
```

```
{
```

```
    os_create_task (1);
```

```
    while (1)
```

```
    {
```

```
        counter0++;
```

```
        os_wait (K_TMO,3,1); }}
```

```
//定义无符号char类型变量counter0
```

```
//定义无符号char类型变量counter1
```

```
//定义任务0
```

```
//创建任务1
```

```
//无限循环
```

```
//更新计数器
```

```
//暂停3个时钟嘀嗒
```

# RTX51操作系统实现2

```
void job1 (void) _task_ 1           //定义任务1
{
    while (1)                       //无限循环
    {
        counter1++;                 //更新计数器
        os_wait (K_TMO,5,1);        //暂停5个时钟嘀嗒
    }
}
```

## RTX51操作系统实现2

下面通过Keil  $\mu$ Vision调试器，说明RTX51的运行机制，步骤主要包括：

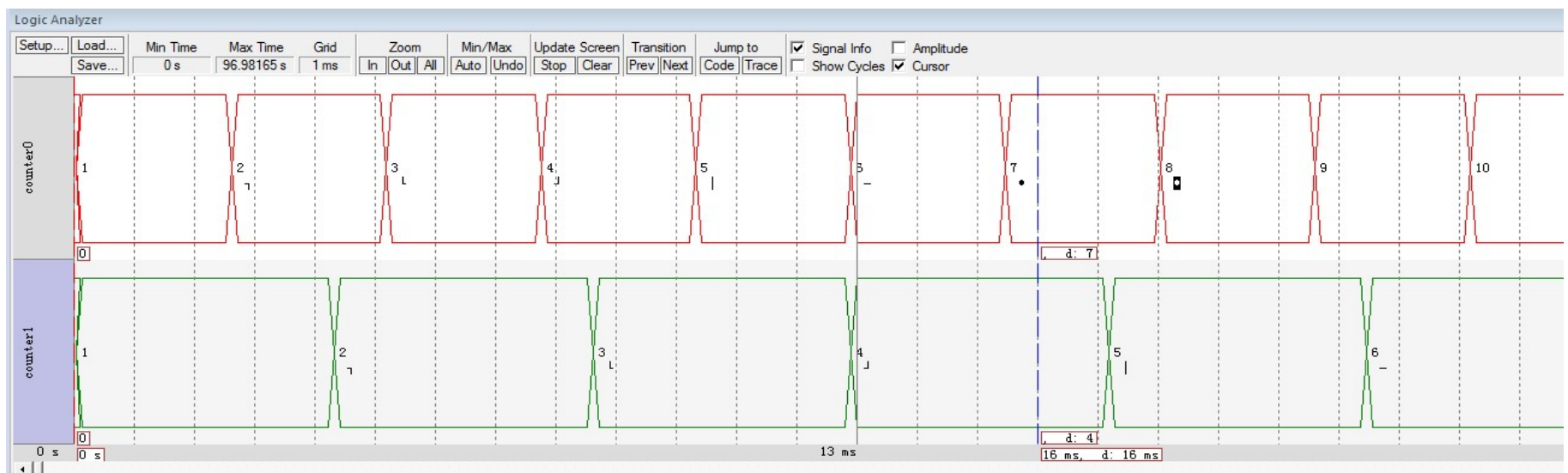
- 在例子17-2目录下，打开该设计。
- 在Keil  $\mu$ Vision当前设计主界面主菜单下，选择Debug->Start/Stop Debug Session，进入调试器模式。
- 在当前调试器主界面主菜单下，选择View->Logic Analyzer选项
- 在当前调试器主界面右侧上方，出现Logic Analyzer界面
- 在该界面中，添加counter0和counter1两个变量。

## RTX51操作系统实现2

- 鼠标右键分别单击counter0和counter1，出现浮动菜单。在浮动菜单内选择state选项。
- 在当前调试器主界面主菜单下，选择View->Watch Windows->Watch 1。
- 在当前调试器主界面右下方出现Watch1窗口界面。在该界面中，添加counter0和counter1两个变量。
- 按F5按键或者在当前调试主界面主菜单下，选择Debug->Run，运行该程序。

# RTX51操作系统实现2

- 观察Logic Analyzer窗口，如图所示。通过观察counter0和counter1，很明显，每3个定时器滴答后递增counter0，每5个定时器滴答后递增counter1。
- 观察Watch 1窗口，可以看到counter0和counter1两个变量的值在交替变化，很明显，counter0变化的比counter1要快。





## RTX51操作系统实现3

在这个例子中，job1等待它从其他任务接收到信号。

- 当它接收到一个信号时，然后递增counter1。
- job0连续的递增counter0，直到溢出到0。
  - 当发生这种情况时，job0发送信号到job1，RTX51让job1准备运行。job1不会启动，直到RTX51得到它的下一个定时器“滴答”。

# RTX51操作系统实现3

## 【例】 RTX51使用信号调度两个任务C语言描述的例子

```
#include <rtx51tny.h>

unsigned char counter0;           //定义无符号char类型变量counter0
unsigned char counter1;           //定义无符号char类型变量counter1
void job0 (void) _task_ 0        //定义任务0
{
    os_create_task (1);           //创建任务1
    while (1)                     //无限循环
    {
        if (++counter0 == 0)      //更新计数器
            os_send_signal (1);   //给任务1发送信号
    }
}
```

# RTX51操作系统实现3

```
void job1 (void) _task_ 1      //定义任务1
{
    while (1)                  //无限循环
    {
        os_wait (K_SIG, 0, 0); //等待信号
        counter1++;            //更新计数器
    }
}
```

## RTX51操作系统实现3

下面通过Keil  $\mu$ Vision调试器，说明RTX51的运行机制，步骤主要包括：

- 在例子17-3目录下，打开该设计。
- 在Keil  $\mu$ Vision当前设计主界面主菜单下，选择Debug->Start/Stop Debug Session，进入调试器模式。
- 在当前调试器主界面主菜单下，选择View->Logic Analyzer选项
- 在当前调试器主界面右侧上方，出现Logic Analyzer界面。
- 在该界面中，添加counter0和counter1两个变量。

# RTX51操作系统实现3

- 鼠标右键分别单击counter0和counter1，出现浮动菜单。在浮动菜单内选择state选项。
- 在当前调试器主界面主菜单下，选择View->Watch Windows->Watch 1。
- 在当前调试器主界面右下方出现Watch1窗口界面。在该界面中，添加counter0和counter1两个变量。
- 按F5按键或者在当前调试主界面主菜单下，选择Debug->Run，运行该程序。

# RTX51操作系统实现3

## ■ 观察Logic Analyzer窗口

