**Computer Science and Application** 计算机科学与应用, 2017, 7(7), 654-661
Published Online July 2017 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2017.77076

Hans汉斯

# The Management Method of Onboard Embedded Software Timer Based on C51 Language

**Jing Wang[1], Lei Hu[2], Zhenhua Wang[1]**

[1]Beijing Institute of Control Engineering, Beijing
[2]Beihang University, Beijing
Email: emwjsj@163.com

Received: Jun. 28th, 2017; accepted: Jul. 10th, 2017; published: Jul. 17th, 2017

## Abstract

**Based on the analysis of the problem of 51 series single-chip timer on board, this paper summarizes the three types of typical problems, and gives a solution. On this basis, we summed up a management method of onboard embedded software timer based on C51 Language. The Method can solve the problem caused by interrupt conflict and timer carry, and can be extended to other embedded field of use of timer.**

## Keywords

**Embedded Software, C51 Language, Timer**

# 基于C51语言的星载嵌入式软件定时器管理方法

**王　晶[1]，胡　磊[2]，王振华[1]**

[1]北京控制工程研究所，北京
[2]北京航天航空大学，北京
Email: emwjsj@163.com

收稿日期：2017年6月28日；录用日期：2017年7月10日；发布日期：2017年7月17日

## 摘　要

**本文通过对星载51系列单片机定时器常见问题进行分析，归纳出三类典型问题，分别给出解决方案，并**

文章引用: 王晶, 胡磊, 王振华. 基于 C51 语言的星载嵌入式软件定时器管理方法[J]. 计算机科学与应用, 2017, 7(7): 654-661. https://doi.org/10.12677/csa.2017.77076

**在此基础上总结出一套基于C51语言的星载嵌入式软件定时器管理方法，该方法可以解决因中断冲突和定时器进位而产生的问题，同时可推广到其他嵌入式领域的定时器使用中。**

## 关键词

**嵌入式软件，C51语言，定时器**



## 1. 引言

在航天嵌入式软件领域，尤其是星载 51 系列单片机软件中，采用 C51 语言进行开发的配置项一直占有较高的比例[1]，其中对于定时器的使用极为常见，是完成某些功能的必要手段。定时器可使某些行为在特定时间执行，也可针对脉冲进行计数从而完成某些功能。但定时器的使用也是航天器软件问题较为集中之处，有关定时器的使用曾发生过多次质量问题。这类问题与时序相关，出现概率低，难复现，甚至在整个动态测试周期都不会出现[2]，那么针对该类型问题，通过动态测试方法一般不能保证对所有故障情况的覆盖[3]，我们采用静态分析的方式进行归纳总结，发现其根源主要集中出现在：① 有中断出现时的定时器读写操作② 读取定时器时间码时有进位发生。

本文结合实例对上述问题进行机理分析，根据不同情况，给出有效的解决方法，并结合航天器软件的设计要求，给出一种通用且可靠的定时器管理方法。

## 2. 51 系列单片机定时器简述

51 系列单片机硬件集成有 16 位可编程定时器。使用最多的是 2 个通用定时器，即定时器 T0 和 T1。其核心是一个 16 位的加计数器，由 2 个 8 位计数器组成的，即 T0 由 TH0 和 TL0 构成，T1 由 TH1 和 TL1 构成[4]。每输入一个脉冲，计数器加 1。当低 8 位计数器增至 255 时，再有脉冲输入，低 8 位计数器数值清零，高 8 位计数器数值加 1，按照该形式累加，当计数器全为 1 时，如再有脉冲输入，计数器将溢出，此时计数器清零，控制寄存器 TCON 中置位中断标志位向 CPU 发出中断处理申请。当使用计数功能时，计数脉冲来自于外部输入引脚 T0 或 T1。当输入信号有负跳变脉冲时计数器加 1。当使用定时器方式时，T0 和 T1 的定时功能也是通过计数实现的，只不过此时计数脉冲来自于单片机的内部时钟脉冲。其内部结构图如图 1 所示。

## 3. 定时器使用问题分析

通过对近年来航天器 C51 软件定时器问题进行总结，归纳出因定时器产生的问题主要分为 3 类，分别为：**中断冲突下定时器操作错误，非中断冲突下定时器操作错误，定时器时间码读取错误**。某些情况下这三种情况会叠加出现，为简化论述，分别对其进行分析。

### 3.1. 中断冲突下定时器操作错误

目前航天器软件多采用中断驱动型架构，该架构下，当主程序及中断程序存在共用资源[1]，如共享全局变量、TH0、TL0、TH1、TL1 等，若软件设计不当，会出现访问冲突。共用资源冲突形式多样，但

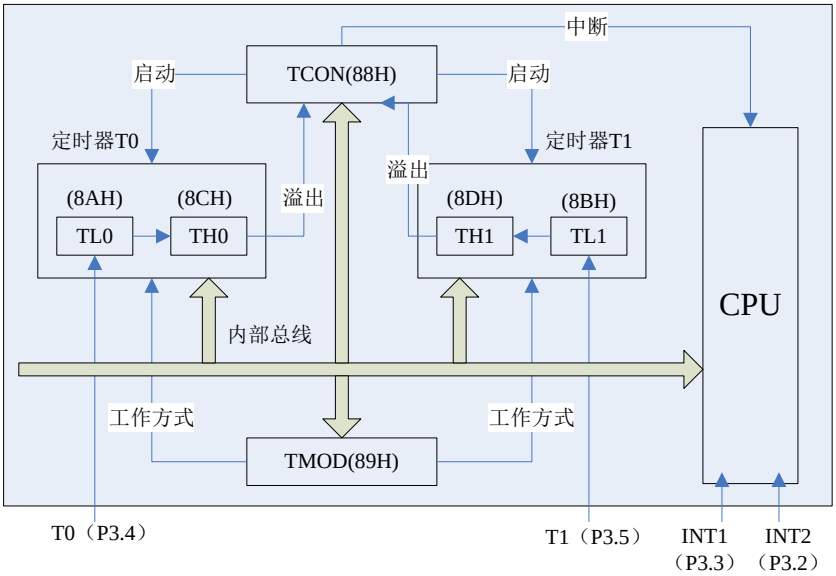



**Figure 1.** Internal structure of 51 series Single chip timer
**图 1.** 51 系列单片机定时器内部框图

究其根源在于对共用资源的非预期更改上[5]。我们从代码层面进行归纳，将存在共用资源的情况进行分类，得到表 1 中的 8 种工况。

当发生如下低概率事件：中断出现时刻恰为箭头指向位置。则在 MAIN()与 INT()中会出现对共享资源的操作，则会出现如下几种工况：

工况 1：中断函数 INT()中读取的全局变量 tm，其数据高 8 位为新数据，而低 8 位由于还未执行 tm.byte.l = TL0 语句，依旧是旧数据。工况 4：主程序 MAIN()中获得的 tm，其数据低 8 位是中断函数 INT()中获取的新数据，而高 8 位是旧数据。工况 6：主程序 MAIN()中读取的定时/计数器数据中，低 8 位 TL0 是中断函数中获得的新数据，而高 8 位 TH0 则是旧数据。工况 7：中断函数 INT()中读取的定时/计数器，其数据高 8 位为新数据，而低 8 位则由于还未执行 TL0 = tm.byte.l 操作，仍然为旧数据。工况 1、4、6、7 均出现由新旧数据拼成的 16 位非预期数据。工况 2 和 8：MAIN()与 INT()均对共享资源 tm 或定时器执行写操作，该工况一定会出现非预期数据。工况 3：INT()中对定时器进行读操作，虽然不更改 TH0 与 TL0,但中断函数自身存在执行时间，会造成时延，导致主程序中最终赋值给 tm 的定时器数据不是同一时刻数据，也属非预期数据，这也是定时器区别于普通共享资源的一个重要特征。工况 5：MAIN()及 INT()均不改变 tm 取值，主程序中对 TH0 及 TL0 的重新赋值不受影响，该工况下不存在冲突问题。

由分析可知，上述 8 种工况中有 7 种存在共用资源冲突的可能性。此外，上述案例伪码针对定时器操作均是先高后低，如果实际代码中顺序相反，结论依然成立。

## 3.2. 非中断冲突下定时器操作错误

区别于 3.1 节，本节针对非中断冲突下定时器操作错误进行分析，即主程序及中断函数不存在共享资源的情况下易出现的时序问题。与 3.1 节对应，同样分为两类进行分析，见表 2。

**工况 1**：主程序赋值过程中高 8 位赋值完毕，低 8 位尚未赋值之时有中断到来。即使中断函数中不存在对 tm、TH0、TL0 的任何操作，但因中断函数自身存在执行时间，导致最终赋值给 tm 的高低 8 位不为同一时刻数据，是非预期数据。**工况 2**：在箭头处有中断到来，中断函数中不存在对 tm、TH、TL0 的任何操作，tm 数据不会变化，tm 无随时间变化的特性，故最终赋值给 TH0、TL0 的数据是正常数据。

**Table 1.** Timer operation case classification in Interrupt conflict
**表 1.** 中断冲突下定时器操作工况分类

| | 工况1 | 工况2 | 工况3 | 工况4 | 工况5 | 工况6 | 工况7 | 工况8 |
|---|---|---|---|---|---|---|---|---|
| 中断函数 INT() | 读共享全局变量tm | 写共享全局变量tm | 读定时器 T0 | 写定时器 T0 | 读共享全局变量tm | 写共享全局变量tm | 读定时器 T0 | 写定时器 T0 |
| 主程序 MAIN() | 定时器数据赋值给某共享全局变量<br>⟹ tm.byte.h = TH0;<br>tm.byte.l = TL0; | | | | 共享全局变量赋值给定时器数据<br>⟹ TH0 = tm.byte.h;<br>TL0 = tm.byte.l; | | | |

注：tm 为 16 位无符号整形全局变量，中断出现时机为箭头指向位置，以 T0 为例，下同。

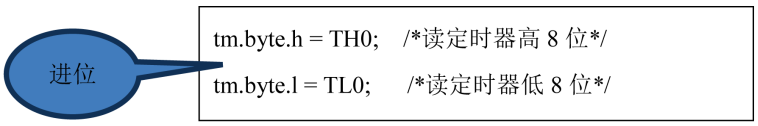
**Table 2.** Timer operation case classification in Noninterrupt conflict
**表 2.** 非中断冲突下定时器操作工况分类

| 工况1：主程序中将定时器数据赋值给变量tm | 工况2：主程序中将变量tm赋值给TH0与TL0 |
|---|---|
| ⟹ tm.byte.h = TH0;<br>tm.byte.l = TL0; | ⟹ TH0 = tm.byte.h;<br>TL0 = tm.byte.l; |

### 3.3. 定时器时间码读取错误

1) 16 位时间码读取错误

定时器使用中，除上述两类因中断出现的问题外，存在另一种易忽略的低概率事件导致读取定时器错误。以 T0 为例，定时器是自加计数，读取 T0 过程中，当有进位情况出现，即低 8 位 TL0 由 255 变为 0，高 8 位 TH0 加 1，如果进位恰好出现在 T0 高 8 位读取完毕，而低 8 位还未读取时，则读取的数据与真实计数器数据最多会出现 255 的差别。如下述代码：




假设执行语句 tm.byte.h = TH0 时，其真实时刻为 0x03ff(即 TH0 为 0x03；TL0 为 0xff)，则读取的定时器高 8 位为 0x03，而执行 tm.byte.l = TL0 语句时，因为定时器继续自增产生进位，所以当前真实时刻为 0x0400(TH0 = 0x04；TL0 = 0x00)，而读取的定时器低 8 位为 0x00，因此最终读取的 16 位数据为 0x0300，与真实时刻 0x0400 相差 256，读取时间码错误。

2) 32 位时间码读取错误

对上述工况扩展，如软件所需时间度量较大，16 位定时器量程不够，通常采用 32 位整形表示时间或计数。如图 2 所示。

该 32 位时间码的高 16 位为中断溢出次数，低 16 位为定时器，因此不仅存在 D0~D7 与 D8～D15 的进位问题，同时也存在 D0~D15 与 D16~D31 的定时器溢出进位问题。原理同上，当低 16 位定时器溢出清零时刻读取时间码，将导致最终读取的数据与真实定时器数据间出现 65536 的差别，引入计算会发生非预期的结果。

## 4. 定时器使用问题解决方法

针对第 3 节中提出的定时器使用问题，我们按照是否因中断引发将其分为 2 类：定时器读写操作过程中有中断出现，即 3.1 节、3.2 节所述问题；读取时间码时有进位发生，即 3.3 节所述问题。下面就解决方案分别进行描述。

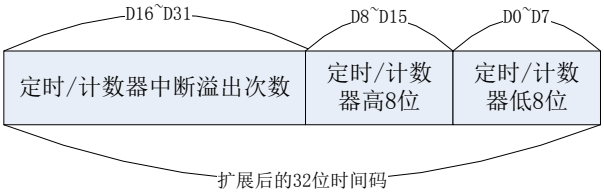



**Figure 2.** Extended 32-bit time code
**图 2.** 扩展的 32 位时间码

### 4.1. 中断导致定时器读写错误的解决方案

针对由中断导致的定时器读写操作错误问题，即 3.1 节提到的 8 种工况，可以采用如下两种方式解决。

1) 在执行定时器读写操作时关闭中断，避免因中断出现而导致定时器读写异常。其伪码如下：

```
EA = 0;//关中断
tm.byte.h = TH0; //读操作
tm.byte.l = TL0;
EA = 1;//开中断
```

```
EA = 0;//关中断
TH0 = tm.byte.h; //写操作
TL0 = tm.byte.l;
EA = 1;//开中断
```

该处理方式有较多应用，简单便捷，是解决中断冲突的首选方法。但若工程应用中对中断响应实时性需求较高，不允许关闭中断，考虑下一种解决方案。

2) 设立中断标志 Intflag，在中断函数中置为真，在主程序读写定时器之前将中断标志设置为假，在读写操作后判断 flag，若为真，则可判定在读写定时器过程中有中断到来，认为操作无效，重新读写定时器，其伪码如下：

| 主程序 MAIN() | | 中断程序 INT() |
|---|---|---|
| //设置中断标志为假<br>Intflag = false;<br>do//定时计数器读操作<br>{<br>  tm.byte.h = TH0;<br>  tm.byte.l = TL0;<br>}//判断中断标志<br>while(Intflag == true) | //设置中断标志为假<br>Intflag = false;<br>do//定时计数器写操作<br>{<br>  TH0 = tm.byte.h;<br>  TL0 = tm.byte.l;<br>}//判断中断标志<br>while(Intflag == true) | //设置中断标志为真<br>Intflag = true; |

### 4.2. 进位导致读取时间码错误的解决方案

针对 3.3 节提出的 2 类问题分别给出解决方案。

1) 16 位时间码读取错误解决方案:前后读取两遍定时器数据，利用数值判断首次读取过程中是否有进位产生，如果存在进位，则进行补偿。其伪码如下：

```
mh1 = TH0;
ml1 = TL0;
mh2 = TH0;
ml2= TL0;
if(((mh1<<8)+ml1+125) <= ((mh2<<8)+ml2)){
   真实时间码 = ((mh1+1)<<8) + ml1;
}
else{
  真实时间码 = (mh1<<8) + ml1;
}
```

2) 32 位时间码读取错误的解决方案：扩展到 32 位时间码后，需获取低 16 位时间码进位情况，采用判断定时器溢出标志的方式，当标志为真，存在进位，则需要考虑补偿。因高低 8 位时间码仍然存在进位可能，将上述伪码进行改进，同时避免高低 8 位及高低 16 位进位的影响，其伪码如下：

```
mh1 = TH0;
ml1 = TL0;
mh2 = TH0;
ml2= TL0;
if(定时器溢出标志为真)
{
   if(((mh1<<8)+ml1+125) <= ((mh2<<8)+ml2))
  {
     真实时间码 = ((定时器溢出次数+1)<<16) + ((mh1+1)<<8) + ml1;
  }
else
  {
    真实时间码 = ((定时器溢出次数+1)<<16) + (mh1<<8) + ml1;
  }
}
else
{
  mh1 = TH0;
  ml1 = TL0;
  mh2 = TH0;
  ml2= TL0;
  if( ((mh1<<8)+ml1+125) <= ((mh2<<8)+ml2) )
  {
     真实时间码 = (定时器溢出次数<<16) + ((mh1+1)<<8) + ml1;
  }
  else
  {
    真实时间码 = (定时器溢出次数<<16) + (mh1<<8) + ml1;
  }
}
```

## 5. C51 定时器通用读写方法

第 4 节针对定时器/计数器使用中的问题提出解决方案，所述问题与时序相关，出现概率极低，我们针对其问题总结如下 3 条原则：

**原则 1**：主程序与中断函数中存在共用资源时需考虑保护，不仅包括“读写”、“写读”、“写写”[6]，针对“读读”形式也需要保护(**定时器特有**)。**原则 2**：主程序与中断函数中不存在共用资源时，主程序中的读定时器操作同样需要保护。**原则 3**：无论所处位置是主程序还是中断函数，所有定时器读操作需要进位保护。

基于上述原则，总结出一套针对定时器读写操作的通用管理方法。

**1)** 将主程序及中断函数中涉及定时器读写操作遍历出来

**2)** 针对不同的操作进行判断

① 若主程序中“定时器数据赋值给共享全局变量”且存在共用资源，中断函数中存在如下 4 类行为需要保护：a 读共享全局变量；b 写共享全局变量；c 读定时器；d 写定时器。则主程序中对定时器的读操作需要保护，首选关中断方式进行保护，如有特殊需求不允许关中断，则采用设立中断标志，使用重新读写的方式进行保护。

② 若主程序中“共享全局变量赋值给定时器数据”且存在共用资源，中断函数中存在如下 3 类行为需要保护：a 写共享全局变量；b 读定时器；c 写定时器。则主程序中对定时器的写操作需要保护，保护方式同上。

③ 若主程序中“将定时器数据赋值给某变量”且不存在共用资源，主程序中的定时器读操作同样需要进行保护，保护方式同上。

**3)** 针对步骤 1 中遍历出来的读定时器操作，无论位于主程序还是中断函数，均需进行定时器进位保护，按照使用的具体情况，分别对 16 位时间码及 32 位时间码进行进位保护。具体保护方式的伪码见 4.2 节。

定时器读写操作通用方法流程图如下，见图 3 及图 4。

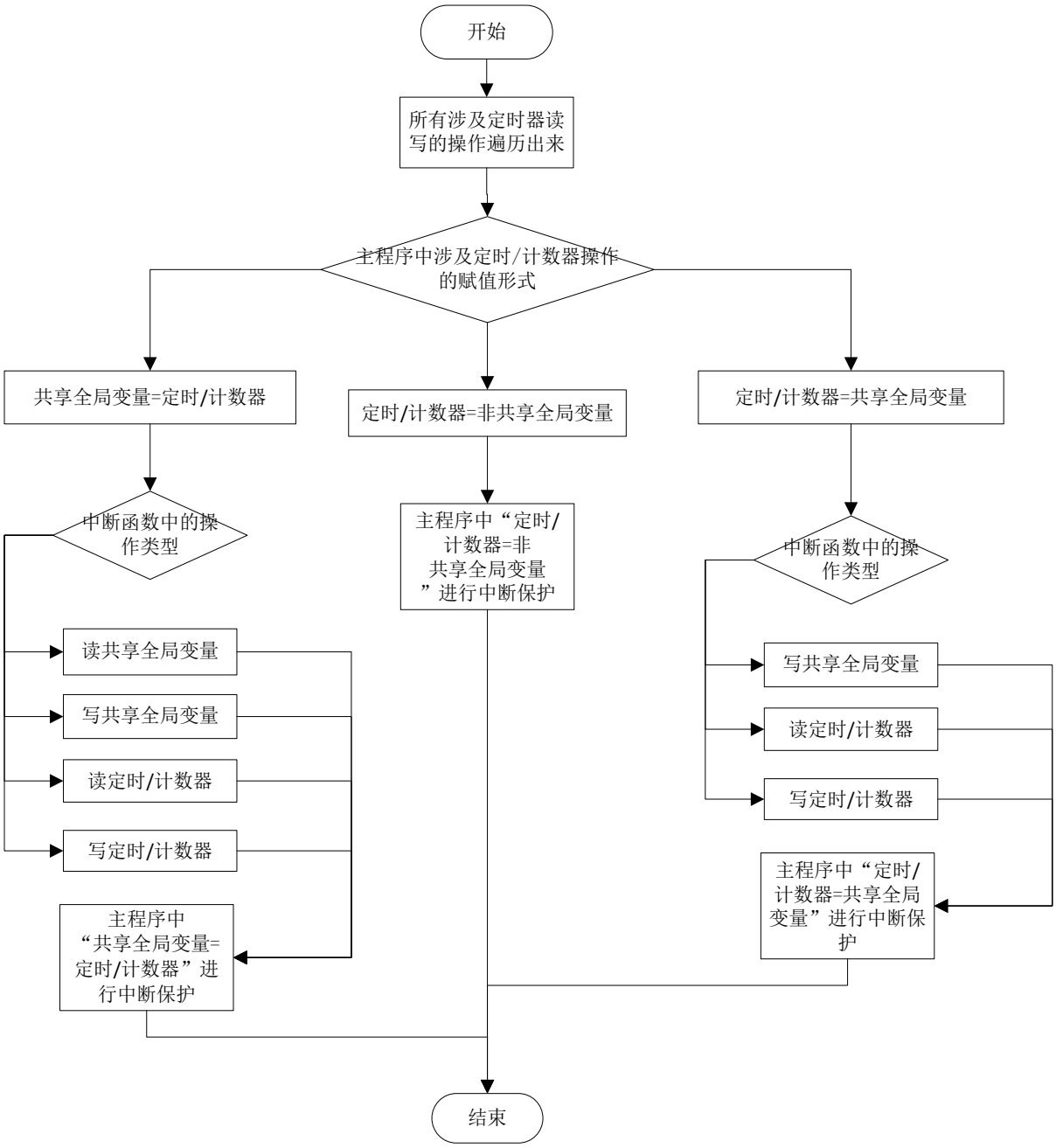



**Figure 3.** General method of interrupt protection flow chart
**图 3.** 中断保护通用方法流程图

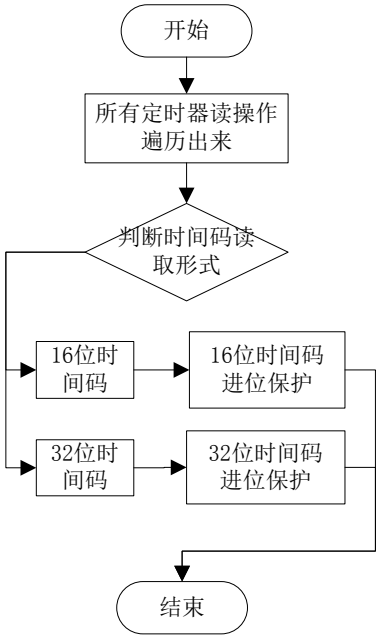



**Figure 4.** General method of carry protection flow chart
**图 4.** 进位保护通用方法流程图

为简化论述，我们将中断保护及进位保护分别论述，真实状态下两类保护方式应叠加出现，即进位保护通用方法应出现在所有进行定时器读取之处。其具体实现方式的伪码见 4.1 及 4.2 节。该方法具有可扩展性，当中断存在嵌套时，采用该方式同样可完成定时器管理操作。

## 6. 总结

本文描述了航天器星载 51 系列单片机软件中定时器使用问题，该类问题存在偶发性，与时序相关。文中按照问题产生机理将其分为 3 类，采用静态分析的方式详细解析了问题成因并提出解决方案，给出一种可靠的定时器管理方法，成功应用于实际型号工作中，极大地提升了定时器使用的可靠性。文中提及的关于定时器的一些使用方法和保护原则可以推广到其他嵌入式工业领域的定时器的研制使用中。

## 参考文献 (References)

[1] 侯成杰. 航天器 C51 语言软件中断资源冲突分析方法[J]. 空间控制技术与应用, 2015, 41(4): 58-62.

[2] 高建军, 兰天, 王斌. 一种高可靠的星载软件时间管理方法[J]. 计算机测量与控制, 2013, 21(3): 806-808.

[3] 霍玮, 于洪涛, 冯晓兵. 静态检测中断驱动程序的数据竞争[J]. 计算机研究与发展, 2011, 48(12): 2290-2299.

[4] 郭坚, 付连芳, 武莹. 基于 8051 单片机的星载软件中断服务程序设计[J]. 计算机测量与控制, 2007, 15(8): 1093-1095.

[5] 杨芳, 齐璇, 董丽, 叶东升. 嵌入式软件中断系统资源冲突检测技术研究[J]. 计算机工程与设计, 2010, 31(23): 5036-5038.

[6] 付佩儒, 谢鹏. 嵌入式软件中断数据竞争安全性检测技术研究[J]. 航天控制, 2015, 33(3): 79-82.