维普资讯 http://www.cqvip.com

文章编号:1003-5850(2007)06-0046-03

μC/OS-II 针对 DSP 的改进和移植

Improvement and Transplanting of μC/ OS-II toward DSP

徐志永 朱凤新

(中北大学 太原 030051)

【摘 要】首先讨论在 Ti 公司 2000 系列芯片 TMS320F2812 上移植嵌入式操作系统的必要性,旨在解决 DSP 软件编程中的复用性和实时性问题。然后介绍源代码公开的实时操作系统 μC/OS-II 的软件架构及其针对 DSP 系统所作的任务堆栈的改进,最后将其移植到该平台上。代码测试结果表明改进和移植是成功的,对解决此类问题具有普遍性。

【关键词】μC/OS-II, TMS320F2812, 嵌入式, 移植

中图分类号: TP368. 1 文献标识码: A

ABSTRACT This paper discusses the necessity of transplanting embedded operation system on TMS320F2812, which is considered a way to solve the reusable and real-time problem on DSP platform. Then the fabric of μC/OS-II and the improvement about task stack toward DSP are introduced. Finally we will transplant μC/OS-II on platform of TMS320F2812. Code test indicates that the improvement and transplanting is successful and that it is a general method.

KEYWORDS μC/OS-II, TMS320F2812, embedded, transplant

数字信号处理器(Digital Signal Processor)是微电子学、数字信号处理、计算机技术这三门学科综合研究的成果,在 20 世纪 70 年代末诞生了首个基于数字信号处理器以来,DSP 不断向高性能、低功耗方向发展。本文中讨论的 TMS320F2812 是 C2000 系列中性能较高的一款芯片。随着 DSP 性能的提高,DSP 所能处理的事务越来越复杂,编制的软件规模越来越大。如何提高软件的复用率,保证软件的实时性,降低软件开发难度是每个 DSP 开发者必须面对的问题。为此,引进实时操作系统是许多 DSP 系统的较好的解决方案,这里采用由 Jean J. Labrosse 所写的实时操作系统 μC/OS-II。

1 移植操作系统的必要性

关于 TMS320F2812 是否适合引入操作系统有两种看法,一种认为 C2000 系列作为工业控制使用,可以做一些控制算法,不如 ARM9 那样适合用来做大系统,DSP 没有必要移植操作系统;另一种认为操作系统确实会带来额外的开销,只要开销在允许的范围内就可以移植操作系统。我们可以先分析一下 2812 这款芯片本身。2812 最高运行频率为 150Mi/s,做 1 024 点的 FFT 变换,实测时间在 1ms 量级上,而实时操作系统的系统时钟一般在 10~100ms 之间,可以看出 CPU 负荷是不高的。2812 采用流水线进程,任务切换会终止流水线,频繁的任务切换会严重影响 DSP 性能。对于 DSP 系统来说,A/D 转换可能是最频繁的引起任务切换的中断源。以 10k 采样率计算,每 0.1ms 就有一次 A/D 转换,如果以这样的速度切换,2812 宝贵的流水线资源会浪费大半。幸好我们可以采取补救措施,如采用 1 024 个字节的 FIFO,在采样到 1 024 个数据后引发一次中断,这样可以把中断切换时间降到系统可以接受 100ms 左右。我们认为只要系统时间要求不是特别苛刻,在能做适当的处理,任务切换频率降低到一定程度的基础上,2812 是适合移植操作系统的。

2 μC/OS-II 的架构分析

μC/OS-II 主要包括:实时操作系统内核、与处理器相关部分、与应用相关部分等。

2.1 实时操作系统内核

主要包括任务分配和调度、系统时钟服务、内存管理、消息机制、异常处理等。μC/OS 所有系统服务均由内核提供。这部分文件是不需要我们修改的。

2.2 与处理器相关的代码

这是移植中最关键的部分。它是内核和硬件之间有一个中间层,处理器不同,这部分代码也不同。在 μC/OS 中这一部分代码分成三个文件:OS _ CPU. H、OS _ CPU _ A. ASM、OS _ CPU _ C. C。

OS _ CPU. H:包括了用 #define 定义的与处理器

* 2006-12-13 收到,2007-04-08 改回

** 基金项目:山西省自然科学基金(20041051)资助。

*** 徐志永,男,1977 年生,在读硕士,研究方向:计算机科学与技术。

维普资讯 http://www.cqvip.com

相关的常量,宏和类型定义。具体来讲有系统数据类型定义,栈增长方向定义,关中断和开中断定义,系统软中断的定义等等。

OS_CPU_A.ASM:这部分需要对处理器的寄存器进行操作,所以必须用汇编语言来编写。包括四个子函数:OSStartHighRdy ( )、OSCtxSw ( )、OSIntCtxSw()、OSTickISR()。

OS_CPU_C.C:μC/OS定义了6个函数在该文件中。但是最重要的是OSTaskStkInit(),其他都是对系统内核的扩展时用的。OSTaskStkInit()是在用户建立任务时系统内部自己调用的,对用户任务的堆栈进行初始化,每个任务专有固定字节的堆栈段。使建立好的进入就绪态任务的堆栈与系统发生中断并且将环境变量保存完毕时的栈结构一致。

2.3 与应用相关的代码

这一部分是用户根据自己的应用系统来定制合适的内核服务功能。包括两个文件:OS_CFG.H,INCLUDES.H。OS_CFG.H来配置内核,用户根据需要对内核进行定制、设置系统的基本情况。INCLUDES.H是系统头文件,包括了内核和用户的头文件。

3 任务堆栈结构的改进

在μC/OS-II内核中,各个不同的任务使用独立的堆栈空间,堆栈的大小按每个任务所需要的最大堆栈深度来定的,这种方法的优点是切换时间很短,缺点是会造成一些堆栈空间浪费。但往往DSP平台的内存空间不是很充足,DSP常作的FFT变换和数字滤波程序也要占用大量的内存。这里采用一种新的任务栈管理方法,多个任务使用同一段连续空间作为堆栈,这样各个堆栈之间就可以互补使用。当然这样会增加任务堆栈切换的复杂性,这是一种以时间换空间的作法。采用这种方法可节约大概一半左右的内存。

如图1,假定任务1首次运行时任务栈为空,运行一段时间后任务2运行,堆栈空间继续往上生长,这次任务切换不需要修改CPU的SP数值。但需要记下任务1的栈顶位置SP1,在任务2运行一段时间后,RTOS又切换到任务1运行,在切换时不能简单将SP指针修改回SP1的数值,这样堆栈向上生长会破坏任务2堆栈中的数据。办法是将原来任务1堆栈保存的数据移到栈顶的位置,而将任务2的堆栈数据下移到栈底的位置,这样堆栈指针SP不需修改。

下面给出改进的堆栈管理办法的具体实现方法:

先为每个任务定义一个堆栈指针结构体

```
Typedef struct TaskSTKPoint{
    int TaskID;
```

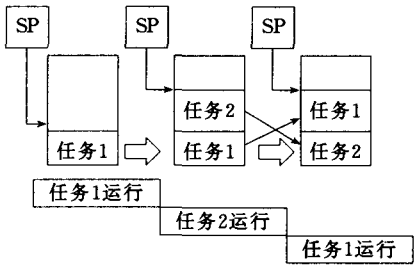

图1 共用任务堆栈时堆栈内容移动

```
    int pTopSTK;
    int pBottomSTK;
}TASK_STK_STRUCT;
TASK_STK_STRUCT pTaskSTK[MAX_TASK_UNM];
```

然后在每个任务在进行任务切换前,运行以下程序段(仅给出示意性代码)

```
STK_ReArrage (int CurTaskID, int NextTaskID)
{
int TempSTK[MAX_TASK_Length];
L=任务号为NextTaskID的堆栈长度;
将TaskSTK顶部的L个字节移动到TempSTK中;
将NextTaskID任务的堆栈内容移动到TaskSTk顶部;
将NextTaskID堆栈上方(移动前)所有内容下移L个字节;
修改NextTaskID堆栈上方(移动前)所有任务栈顶和栈底指针(在pTaskSTK中修改)}。
```

4 实时操作系统的移植代码改进

需要用户编写的主要是四个与CPU相关的函数:tartHighRdy ()、OSIntCtxSw ()、OSTickISR ()、OSCtxSw()。前三个是常规操作,只有最后OSCtxSw()需要细心编写,因为它还和我们的编译环境CCS2.2和DSP都有关系。

```
PUSH        ALL
MOVW        DP,#_OSTCBCur
MOVL        XAR4,@_OSTCBCur
MOV         AL,SP
MOV         *+XAR4[0],AL
MOVW DP,#_OSTCBHighRdy
MOVL ACC,@_OSTCBHighRdy
MOVW        DP,#_OSTCBCur
MOVL        @_OSTCBCur,ACC
MOVW        DP, #_OSPrioHighRdy
```

(下转第50页)

维普资讯 http://www.cqvip.com

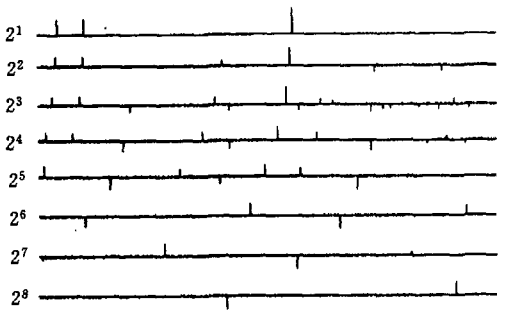

图 3　设定阈值 0.4 滤波后的极大值点

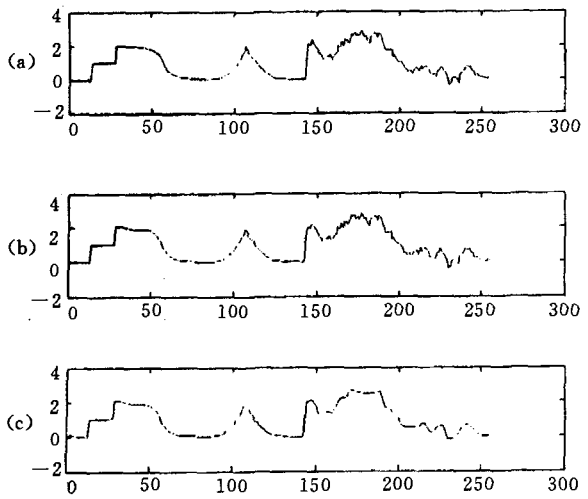

图 4　(a)原始信号　(b)极大值重构
(c)阈值滤波后极大值重构

综合上述分析，我们还可以看出在缓慢变化的低频段，只用少量模极大值点就可表征信号的变化。以图 2 信号的前 128 个点为例，$j>4$ 时小波变换不反映突变点，取 $j<4$ 的点其压缩比为 15/128＝1/9，对于像图 4(a)前 128 个点一样的平稳信号，取 $j<3$ 则只有 5 个极大点，原始信号有 128 点时可获得 5/128＝1/25 的压缩比。

4　结论

本文探讨了小波变换极大值在数据压缩方面的应用，给出了采用二次样条小波对一维数据进行压缩的原理及方法。理论分析及实验结果表明，与传统利用小波多分辨率特性的压缩方法相比，小波极大值压缩可以在相同误差下获得更大的压缩比，而且对于平稳信号则可以获得传统方法无可比拟的极大的压缩比。可以预见小波变换极大值一定会在一些数据压缩领域得到很好的应用。

参 考 文 献

[1]　刁彦华. 基于小波变换模极大值的信号奇异性检测[J]. 河北工业科技，2004，21 (1)：1-3，32.

[2]　Logan B. Information in the zero-crossings of band pass signals [J]. Bell System Tech J，1977(56)：510.

[3]　Mallat S，Zhang S. Characterization of signal from multiscale edges [J]. IEEE Trans Pattern Analysis and Machine Intelligence，1992，14(7)：710-732.

[4]　RIOUL O，VETTERLI M. Wavelets and signal processing IEEE Signal Processing [J]. Mag，1991，5 (2)：14-38.

[5]　Stephane Mallat . A Wavelet Tour of Signal Processing. Academic Press，1998.

(上接第 47 页)

```
MOV     AL, @_OSPrioHighRdy
MOVW    DP, #_OSPrioCur
MOV     @_OSPrioCur,AL
MOVW    DP,#_OSTCBHighRdy
MOVL    XAR4,@_OSTCBHighRdy
MOV     AL, *+XAR4[0]
MOV     @SP,AL
POP     RPC
POP     ALL
IRET
```

5　用户应用系统的编写

这里特别指出，为了移植的方便，所有中断函数都采用 C 语言加 interrupt 关键字的方式给出。但是开发软件 CCS2.2 对这种中断调用并不是把当前任务全部寄存器压入堆栈，而是由开发软件把中断服务程序要用到的寄存器压入堆栈。这本来是 CCS2.2 为了缩短中断反应时间而作出的优化，却给我们移植带来了不便，因为 μC/OS-II 要求所有任务堆栈中 CPU 寄存器部分是统一的。网上流传很广的移植代码并没有注意到这个问题。解决的办法是中断服务程序中不要定义任何变量，只写一条调用函数语句，由这个被调用的函数完成所有功能。更详细的解释可参照 TI 技术文档 SPRU514。

至此，完整的移植过程就完成了。经实际测试，各个功能部件都能很好地工作，移植工作是成功的。

参 考 文 献

[1]　Labrosse. 嵌入式实时操作系统 μC/OS-II[M]. 北京：北京航空航天大学出版社，2003.

[2]　TI. TMS320C28x Optimizing C/C + + Compiler User's Guide，TI，SPRU514，2001.

[3]　TI. TMS320C28x DSP CPU and Instruction Set Reference Guide，TI，SPRU430，2002.