# SHUM-UCOS:基于统一多任务模型可重构系统的实时操作系统

周 博[1) ] 王石记[2)] 邱卫东[1)] 彭澄廉[1)]

[1)](复旦大学计算机与信息技术系 上海 200433)

[2)](哈尔滨工业大学通信技术研究所 哈尔滨 150001)

**摘 要** 在分析软硬件任务本质区别的基础上,提出并实现了一种基于统一多任务模型的实时操作系统,称为SHUM-UCOS.它能够跟踪和管理可重配置资源的使用,通过硬件任务预配置技术,提高了资源利用率和任务并行性.定义了两种标准硬件线程接口,对总线型和星型通信拓扑结构提供支持.Rhealstone Benchmark 测试和实际应用都表明,SHUM-UCOS 能够在提升系统性能的同时,有效缩减从软件实现到硬件实现的迁移时间.

**关键词** 可重构系统;统一多任务模型;预配置;实时操作系统;现场可编程门阵列;UCOS

**中图法分类号** TP316

# SHUM-UCOS: A Real-Time Operation System for Reconfigurable Systems Using Uniform Multi-Task Model

ZHOU Bo[1)] WANG Shi-Ji[2)] QIU Wei-Dong[1)] PENG Cheng-Lian[1)]

[1)](*Department of Computer and Information Technology, Fudan University, Shanghai* 200433)

[2)](*Communication Research Center, Harbin Institute of Technology, Harbin* 150001)

**Abstract** Based on the essential differences between software-tasks and hardware-tasks, this paper presents and implements a real-time operation system(RTOS) for reconfigurable systems using uniform multi-task model, called SHUM-UCOS(Software-Tasks Hardware-Tasks Uniform Management UCOS), which is designed with the UCOSII as prototype. This real-time operation system(RTOS) traces and manages the usage of reconfigurable resources(FPGAs), and can improve the utilization of these resource and the parallelism of the tasks with the hardware-tasks preconfiguration. SHUM-UCOS also defines two types of standard hardware-task interface, which can support bus protocol and point-to-point protocol separately. And it has been proved by rhealstone benchmark and applications that SHUM-UCOS can shorten the transition time from software implements to hardware implements with the performance improvement.

**Keywords** reconfigurable computing system; uniform multi-task model; preconfiguration; real-time operation system(RTOS); FPGA; UCOS

**收稿日期**:2004-12-20;**修改稿收到日期**:2005-08-20.**本课题得到国家自然科学基金**(60573105)**资助**.**周 博**,男,1979 年生,博士研究生,主要研究方向为可重构计算、演化硬件与智能优化算法. E-mail: allenzhou@xasamail.com. **王石记**,男,1979 年生,博士研究生,主要研究方向为高性能计算、超宽带通信、卫星通信. **邱卫东**,女,1972 年生,博士研究生,主要研究方向为嵌入式快速样机平台和实时操作系统建模与实现. **彭澄廉**,男,1941 年生,教授,博士生导师,主要研究领域为软硬件协同设计、分布式系统监测、容错计算等. E-mail: clpeng@fudan.edu.cn.

# 1 引　言

为了得到优化的嵌入式产品设计，设计人员常常需要进行设计空间搜索，其中系统功能的软硬件划分是至关重要的问题. 以往由于软硬件缺乏统一的模型支持，在同一功能由软件实现向硬件实现迁移时，即使是细微的变化，也往往需要对原有系统进行大量改动. 这种迁移也往往与具体应用场景相关，缺乏通用性. 极端的情况就是在功能划分不变的情况下，由于实现方式迁移，仍需要为单一产品设计多种实现. 这种方式显著增加了设计负担与难度，为此目前嵌入式系统设计领域提出了多种方案以跨越软硬件边界. 如 Ptolemy[1]，SystemC[2] 的目标是提供系统级的描述能力以便把软件编译和硬件综合纳入到统一的框架内；而 Handel①，Streams-C② 则主要研究从高级语言到 RTL 级的综合技术. 这些工作极大推动了嵌入式系统设计方法学的发展，遗憾的是他们都偏重于研究高层抽象和方法学，与实际应用仍然存在距离.

现场可编程门阵列(Field Programming Gate Array，FPGA)出现后，由于其具有可编程、重配置的特性，在嵌入式系统内大量应用. 然而在传统的设计中，FPGA 在系统中类似于 ASIC，基本上都是作为硬件加速器，由用户直接管理和使用，操作系统作为资源管理者并不参与管理，至多提供一些驱动作为支持. 这种方式忽略了 FPGA 内在的可重构特性以及任务潜在的并行性，增大了系统设计难度，也降低了 FPGA 的使用率.

考虑到 FPGA 应用范围不断扩大的趋势及其固有的特点，操作系统必须进行改进以适应器件和应用的发展.

面对这样的情况，Andrews 等人在文献[3]中提出了将硬件功能模块视作硬件任务，纳入操作系统管理范围内的观点，并采用统一多任务模型对嵌入式系统进行了抽象. Walder 等人也认为操作系统对可编程器件进行抽象是必须的，而且在文献[4]中定义了一种能够支持可重构硬件的操作系统框架.

统一多任务模型是已被广泛接受的多线程(任务)模型(如 POSIX 线程规范[5])的进一步扩展. 由于是传统设计方法的自然迁移，因此更容易被设计人员采纳. 而且该模型侧重于软、硬部件的交互和管理，与其它系统级设计与验证技术的研究重点并无冲突，而是互为补充.

尽管 Andrews 等人提出了统一多任务模型，并做了相当杰出的工作，但其模型还存在以下不足：(1)仅支持硬件任务静态创建和共享内存方式通信方式，在实际应用中存在局限性. (2)缺乏对可重配置资源的管理. (3)缺乏对软硬件任务的区别，未发挥硬件任务的并行性.

在上述工作的基础上，本文改进了多任务模型，设计并实现了对软硬件任务统一管理的实时操作系统(RTOS)SHUM-UCOS. 该 RTOS 具有以下特点：(1)支持动态硬件任务创建，并采用了基于调度器的硬件任务预配置技术. (2)定义了标准硬件任务接口，能够对多种拓扑和通信方式支持. (3)通过对任务图的静态分析，提高了任务并行性和可重配置资源的利用率.

# 2 相关概念

**定义 1.** 最小可重配置单元(Minium ReConfigurable Unit，MRCU)定义为当电路功能 $F$ 发生任意小改变 $\sigma$ 时，系统需要配置的最小单元.

FPGA 是目前最主要的可重配置器件，根据每次可配置的范围，分为部分配置和完全配置两种类型[6]. 完全配置类型的 FPGA，如 Altera 的 Cyclone 系列③，由于任何电路功能的改变都会造成整片 FPGA的配置 SRAM 被重写，因此这种类型的 FPGA 的最小可重配置单元即为单片 FPGA 芯片. 部分配置类型的 FPGA，如 Xilinx 的 Virtex 系列④，可以对其中的一个 slot 进行配置，因此其最小可重配置单元为芯片内的单个 slot.

**定义 2.** 可配置任务 $T_i$ 通常指可以被配置到可重配置器件中的数字功能模块. 运行时可配置任务 $T_i$ 的属性包含面积和时间两个方面，可以将其看作五元组 $T_i=\langle w_i, a_i, e_i, c_i, d_i\rangle$，其中，

$w_i$ 表示 $T_i$ 所占用的 MRCU 数目，反映了其对资源的占用情况.

$a_i$ 表示可配置任务的到达时间.

$e_i$ 表示可配置任务的执行时间，可以通过计算可配置任务的时钟周期数确定任务的执行时间.

$c_i$ 表示可配置任务的配置时间，与任务的位流文件长度成正比.

① Celoxica Corporation, Handle-C. Language Overview. http://www.celoxica.com, 2002

② Maya Gokhale. SC2 Reference Manual, Los Alamos National Laboratory, 2002, http://www.streams-c.lanl.gov

③ Altera Corporation, Cyclone Programmable Logic Device Family Datasheet, 2003, http://www.altera.com

④ Xilin Corporation. Virtex 2.5V FPGA Complete Data Sheet(all four Modules), 2002, http://www.xilinx.com

$d_i$表示可配置任务的截止期(deadline). 实时任务必须满足以下时间约束关系:$d_i \geq a_i + e_i + c_i$.

**定义 3.** 可配置任务的实现与可配置单元的分布无关. 对于给定可配置任务集合 $T=\{T_1, T_2, \cdots, T_n\}$与资源集合 $R=\{R_1, R_2, \cdots, R_r\}$,根据任务粒度与资源粒度的相对关系,我们称以下三种情况为可配置任务的实现与可配置单元的分布无关(其中 $Area(T_x)$表示 $T_x$ 占用的面积). 同时为了表述方便,我们将下述 3 类无关条件分别称为无关分类 I,无关分类 II 和无关分类 III.

(1)当 $Area(MRCU) \geq Max(Area(T_i))$, $T_i \in T$ 时,对于任意给定任务子集 $P=\{T_1, T_2, \cdots, T_p\}$, $P \subseteq T$,若满足$\sum_{i=1}^{p} Area(T_i) < Area(MRCU)$,则其中的 $p$ 个任务都可以被实现.

(2)当 $Area(MRCU) \leq Min(Area(T_i))$, $T_i \in T$ 时,对于任意给定资源子集 $Q=\{R_1, R_2, \cdots, R_q\}$, $Q \subseteq R$,若满足$\sum_{i=1}^{q} Area(Q_i) \geq Area(T_i)$, $T_i \in T$,则任务 $T_i$都可以被实现.

(3)当 $Min(Area(T_j)) < Area(MRCU) < Max(Area(T_i))$, $T_i, T_j \in T$ 时,任务集合 $T$ 可以被拆分为两个互不相交的子集$T'$, $T''$,即$T' \cup T'' = T$, $T' \cap T'' = \varnothing$,其中$T'$满足上述条件1, $T''$满足条件2.

可配置任务与可配置单元分布无关是后续讨论的重要前提条件. 首先它保证了调度器对硬件任务的可选择性,同时也用于指导任务划分. 实际应用时,鉴于无关分类 III 的复杂性,更倾向于使任务划分粒度落入无关分类 I 或无关分类 II 中. 事实上,一种简单的原则就是:如果最小可重配置单元的粒度较大,采用无关分类 I(最大任务面积小于最小可重配置单元的面积),反之采用无关分类 II. 本文采用的实验平台是基于 Altera 公司的 FPGA,可配置器件的粒度较大,因此我们主要讨论无关分类 I. 由于无关分类 I 和无关分类 II 的主要区别在于:任务集合与可配置资源集合间的映射方向相反,因此本文的大部分设计方法和结论仍然适用于无关分类 II.

## 3 关键问题

硬件任务和软件任务由于实现方式和物理基础的不同,存在很大差异. SHUM-UCOS 的最终目标是使得软硬件任务对设计者透明,因此操作系统设计者需要了解以下软硬件任务的根本区别,同时它们也是 SHUM-UCOS 需要解决的关键问题.

(1)硬件任务的数量受限于可重配置资源数量,而软件任务的数量受限于存储器容量.

(2)硬件任务的创建通常是通过对 FPGA 进行配置来实现(全部配置或部分配置),其创建时间较长(往往在毫秒级,甚至秒级). 对于实时嵌入式系统,硬件任务创建代价不可以忽略不计.

(3)软件任务必须通过任务切换分享计算资源 CPU,硬件任务通常创建后就独占相应计算资源,因此硬件任务在运行态不必进行任务切换[3].

(4)通过保存内部寄存器以及任务堆栈,软件任务较容易执行挂起操作. 硬件任务的挂起操作与实现方式相关. 如果用有限状态机的概念来分析,设原有状态集合为 $S=\{S_1, S_2, \cdots, S_n\}$,则增加与前 $n$ 个状态连接的新状态 $S_b$,将得到新的状态集合 $P=\{S_1, S_2, \cdots, S_n, S_b\}$. 此时将增加 $2n$ 个迁移条件,同时每个状态都要判断是否需要挂起. 根据实验,当状态图规模较大时,与原有状态全连接的方式会对硬件任务性能产生较大影响. 主要是由于新增加的状态与其它状态不同,是"全局的". 在 EDA 工具综合时,将出现大量全局连线,增加布局布线难度,并影响最终时序性能,这种影响在布线资源紧张的情况下尤为明显. 因此在设计硬件任务时,通常只能选择一些关键状态,使之能够响应挂起操作原语,如图 1.

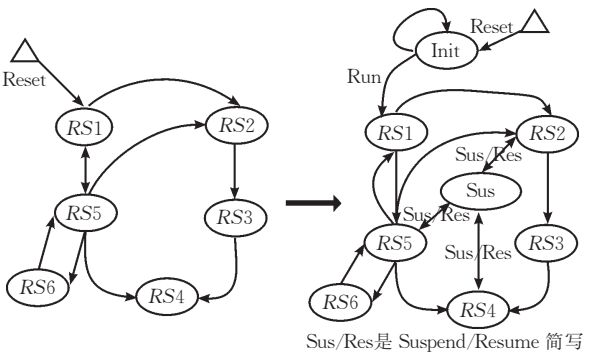



图 1 硬件任务挂起的状态图

(5)与软件任务不同,同一时刻可以有多个硬件任务处于激活状态.

## 4 SHUM-UCOS 的统一任务模型概述

SHUM-UCOS 通过扩展 UCOSII 来增加管理范围,保留了大部分数据结构,并沿用了基于优先级的调度策略. 当处理软件任务时,二者基本一致. 涉及硬件任务时,SHUM-UCOS 采用改进的统一任务模型来处理,如图 2. 该模型将硬件处理过程作为新的任务类型——硬件任务,与软件任务一起纳入管理. 这样一方面使得硬件细节对设计人员透明,另一

方面帮助设计人员更关注逻辑结构，采用统一模型组织用户程序.

从整体上，该模型分为 3 部分(见图 2 右侧)：CPU、硬件任务管理器、可配置器件(FPGAs). 软件任务在 CPU 上运行，硬件任务在 FPGA 上运行. SHUM-UCOS 也由软件部分和硬件部分构成. 软件部分包括软件任务接口、调度器和资源管理器. 硬件部分称为硬件任务管理器，通常采用 FPGA 实现，包括通信控制器、两类任务接口(类型 I 和 II)、配置接口以及硬件任务配置控制器.

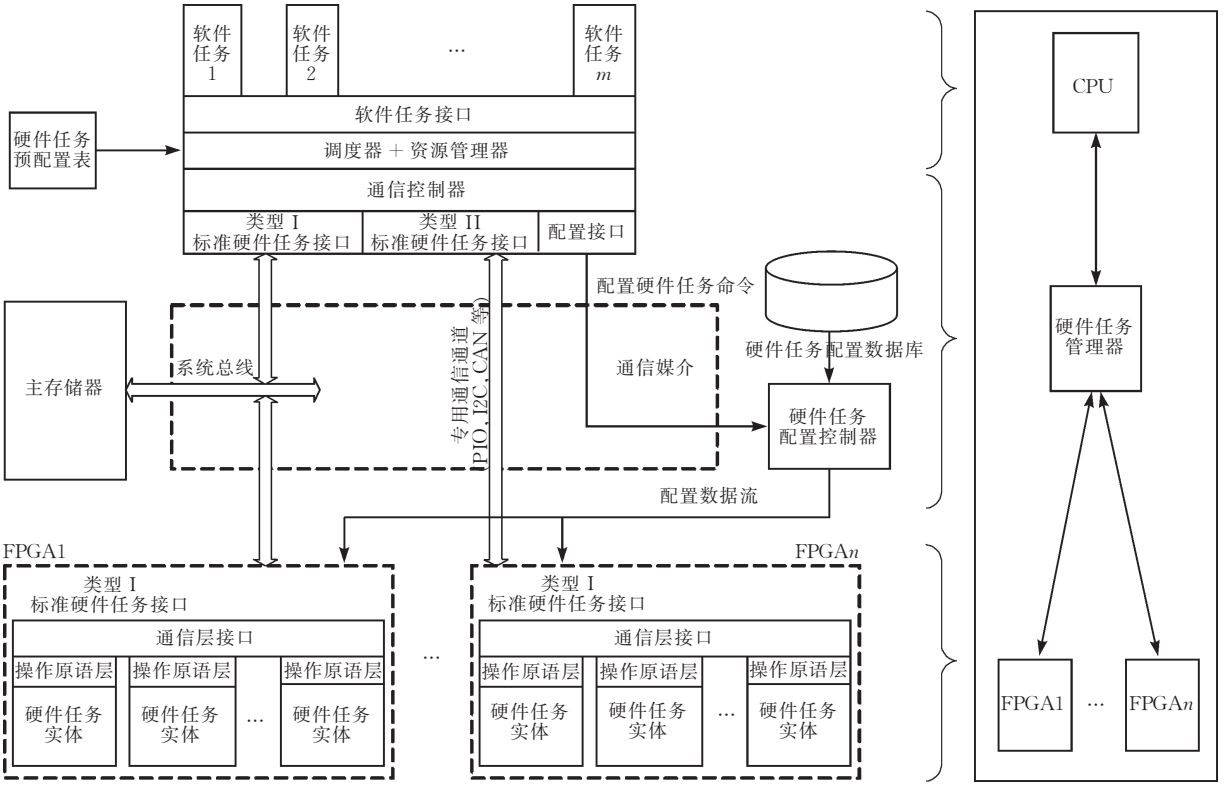



图 2　SHUM-UCOS 统一任务模型

下面简要按照从上至下的顺序介绍模型中的各个组成部分：

软件任务接口. 它是一组 API 函数的通称，这组函数作为软件任务与操作系统的交互接口，为用户任务提供系统服务，如信号量的创建、删除等.

硬件任务预配置表. 为了减小运行时的硬件任务配置代价，通过对任务执行流的静态分析，可以得到硬件任务的配置顺序. SHUM-UCOS 把其保存为硬件任务预配置表，使得调度器能够在硬件任务运行之前就进行配置.

调度器. 它是操作系统核心. 负责管理软硬件任务的状态，处理系统的同步或异步事件. 如调度软件任务或配置硬件任务.

资源管理器. 由于硬件任务的动态创建和释放，可重配置资源的使用情况也在不断变化. 资源管理器跟踪这些变化，作为调度器选择硬件任务的判断依据.

通信控制器. 该模块处理底层通信细节，与任务划分与应用场景相关，通过不同的硬件任务接口，可以实现不同的物理层协议或者通信拓扑结构.

硬件任务配置数据库. 存放事先由综合工具综合，并存储在非易失存储器中的 FPGA 的配置数据.

硬件任务配置控制器. 它是与可配置器件配置方式密切相关的控制器，接受调度器的配置命令后，从配置数据库中取出对应 FPGA 的配置数据，完成其配置. 可采用简单的 4 位或 8 位控制器实现. 本文采用 8051 软核实现.

标准硬件任务接口. 硬件任务的设计，由于要跨越 CPU-FPGA、FPGA-FPGA 边界，所以常常受到多种条件制约，如信号完整性、功耗、通信拓扑等，需要采用不同的实现方式. SHUM-UCOS 提供两种标准硬件任务接口，以提供对总线型和星型通信拓扑的支持.

## 5　模型实现

### 5.1　生成硬件任务预配置表

在进行嵌入式系统分析时，任务间的依赖关系

可以采用任务图表述. 任务图是加权有向无环图 DAG,定义为 $G=(V,E,W,A)$,其中 $V$ 是应用中所有任务的集合, $|V|=n$. $E$ 是有向边集,边 $(V_i,V_j)$ 限定任务的数据依赖关系,任务 $V_i$ 是前驱,任务 $V_j$ 是后继,任务执行的数据依赖关系决定了任务执行的顺序关系:只有前驱任务执行完毕,后继任务才可以执行. 每个任务 $v_i\in V$ 都具有任务截止期 $w_i\in W$,并占用一定的可配置资源 $a_i\in A(1\leq i\leq n)$.

在无关分类条件 I 下,单个最小可重配置单元内可以容纳多个任务. 生成硬件任务预加载表,可以分解为两方面的任务:(1)在空间上将任务分组,使得每组任务的面积和小于最小可重配置单元的面积. (2)在时间上,对任务组进行调度,使得其占用的可重配置资源最少. 对于 DAG 图来说,无论是任务的分组还是任务的调度,都是 NP 完全问题[7~9].

在无关分类条件 II 下,每个任务都需要一个或多个最小可重配置单元. 即配置单元的粒度小于任务粒度,只需要在时间上对任务调度,使得其占用的可重配置资源最少.

特别的,通过对任务图的静态调度分析,获得任务执行的并行性,这个问题在多处理机领域已多有讨论,并相继提出了多种算法,如 MCP 算法、DCP 算法等[7].

文献[8]中则讨论了在面积约束条件下的任务划分问题,并提出了两种算法:(1)基于层次的划分算法. (2)基于簇的划分算法. 两种算法的目标不同,前者的目标是获得最大并行度,而后者则是尽可能减小通信代价.

SHUM-UCOS 中生成硬件任务预配置表的基本思想是:将可重构资源的配置看作具有截止期的任务,通过对分组和调度,获得硬件任务预加载表. 为此 SHUM-UCOS 通过以下步骤获得预配置表,如图 3.

(1)移除原任务图 $G_1$ 中的软件任务,得到仅关于硬件任务的任务子图 $G_2$,其任务依赖关系保持和 $G_1$ 一致. 见图 3(b).

(2)将硬件任务子图 $G_2$ 的节点 $T_i$ 替换为配置任务 $C_i$. $C_i$ 的截止期等于任务 $T_i$ 的到达时间与其配置时间之差,见图 3(c).

(3)划分硬件任务分组:在资源面积约束条件下,采用基于层次的划分算法得到任务分组. 见图 3(d).

(4)合并任务分组为配置节点. 见图 3(d).

(5)采用 DCP 算法得到硬件任务预配置表. 见图 3(e).

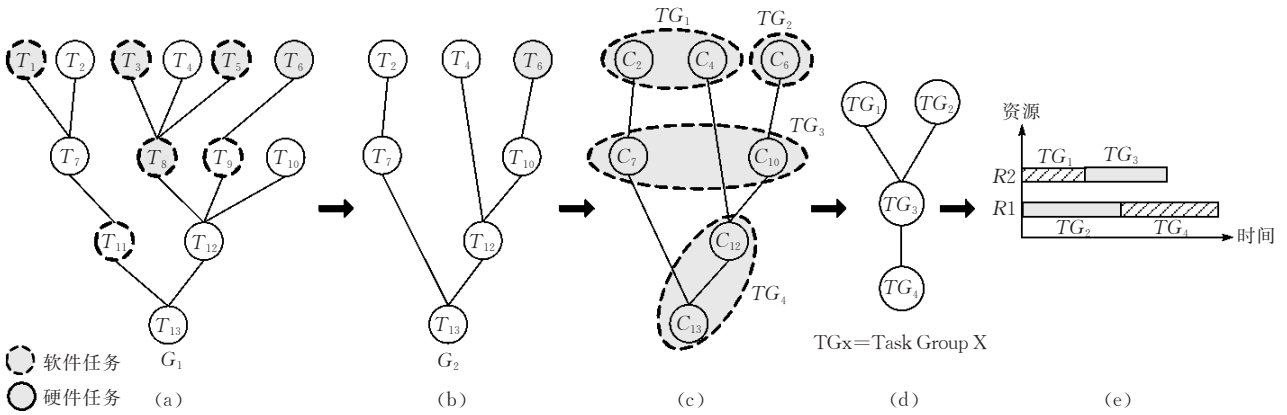



图 3 生成任务预配置表

按照以上步骤可以得到硬件任务预加载表,但不能保证其划分和调度结果最优. 不过在多数的应用场景中,不存在预先确定的任务执行流,而且任务之间的同步也会加大任务到达时间的不确定性. 所以在动态执行过程中,静态分析的最优结果往往不是最优解,但硬件任务预加载对于调度总是有益的.

### 5.2 硬件任务的管理

SHUM-UCOS 将 UCOS 的最大任务数从 64 个扩展到 128 个,优先级为偶数的任务为软件任务,优先级为奇数的任务为硬件任务. 如第 3 节所述,由于硬件任务不存在剥夺和抢占现象,因此在调度时硬件任务的优先级已不具备优先权的概念,硬件任务只要就绪就可以运行. 仅在对共享资源竞争访问时,需要考虑硬件任务的优先级.

当相同的配置数据被配置到不同资源中时,系统中就出现多个同样的任务. 借用面向对象的概念,我们称存储在硬件任务配置数据库中的配置数据为硬件任务类,一旦被配置到可重配置资源中,就称为该硬件任务类的实例.

SHUM-UCOS 不仅支持硬件任务的动态创建,

也支持同一硬件任务类创建多个实例.

SHUM-UCOS 中，每个硬件任务都有唯一的硬件任务控制块（Hardware-Task Control Block，HCB），该控制块是一个由调度器管理的数据结构，驻留在内存中，其中数据成员记录了硬件任务的运行状态、相关指针. 根据用户的需要，该控制块内的数据成员可以变化，但至少包括以下成员：

OSResourceNO. 它用来指明该硬件任务位于哪个可重配置资源内.

OSHwType. 硬件任务类，用于从硬件任务配置数据库中检索出配置数据.

OSHCBPrio. 该硬件任务的优先级.

OSHCBStat. 硬件任务的执行状态，如就绪、运行、阻塞等.

OSHCBDly. 硬件任务超时限制，通常在需要硬件任务延时或者挂起一段时间时使用. 每个系统时钟节拍，该变量减 1.

OSHCBPendTO. 它用于指示挂起是否结束的标志.

OSTCBEventPtr. 指向事件控制块的指针，事件控制块是 UCOS 中的同步和通信机制共用的数据结构，如互斥量、信号量和邮箱等，其中含有一个链表，用于表示等待该事件的任务.

OSTCBMsg. 当使用邮箱进行进程间通信时，该参数指向需传递的数据头部.

OSMsgLen. 邮箱所传递的消息长度，以字节计算.

OSInterfaceType. 用于说明该硬件任务的接口（类型 I 或类型 II）.

OSInterfaceControl. 指向该硬件任务的接口控制 TICB，该控制块定义了硬件接口可以访问的寄存器，见第 6 节.

OSHCBNext 和 OSHCBPrev. SHUM-UCOS 将在同一资源内的任务组织成双向链表，OSHCBNext 和 OSHCBPrev 分别用于指明当前任务的后继任务和前驱任务.

### 5.3　可重配置资源管理

在 SHUM-UCOS 中定义了资源控制块（Resource Control Block，RCB）来管理可重配置资源的状态以及使用. 它是一个数据结构，如下所示.

```
typedef struct os_rcb {
  INT8U ResourceArea;
  INT8U ResourceNo;
  INT8U ActiveTaskCount;
  struct os_rcb *OSRCBNext;
  struct os_hcb *OSHCBFirst;
} OS_RCB
```

ResourceArea. 指明该可配置资源的面积大小.

ResourceNo. 该资源标号，在 SHUM-UCOS 中，为每个资源定义不同的标号.

ActiveTaskCount. 该资源内处于激活（非休眠）状态的任务数目.

OSRCBNext. 指向链表内的下一个资源控制块.

OSHCBFirst. 指向该可配置资源内的第一个任务.

SHUM-UCOS 定义资源只能处于以下 4 种状态之一：(1)占用状态. 资源内已经配置了任务组，且任务组中至少存在一个任务处于调度器的管理之下（运行、就绪、阻塞）. (2)预配置状态. 资源内已经配置了任务组，所有任务都处于休眠状态，等待被激活. (3)空白状态. 资源内未配置任务组，或者需要重新配置任务组. (4)配置状态. 资源正在配置任务组.

SHUM-UCOS 中存在四个链表分别与这 4 种状态对应（如图 4）. 处于预配置状态的任务如果在规定的时间内被调度器创建或调用，称为预配置命中. 否则称为预配置不命中，该任务所占用的资源被置为空白状态.

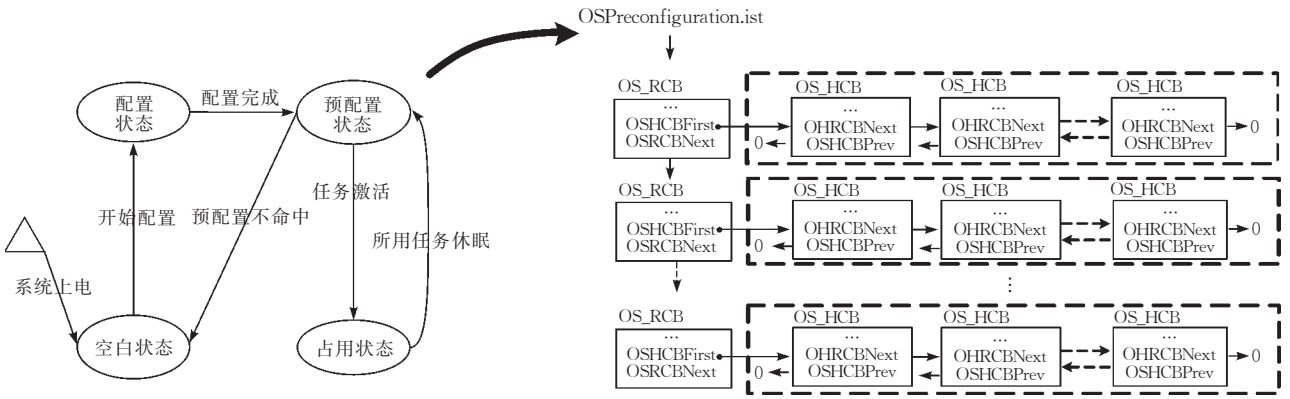



图 4　SHUM-UCOS 中的资源管理

为了减少资源配置代价,当一个资源内的所有任务都被调度器删除时,SHUM-UCOS 并不直接将该资源置为空白状态,而是先置为预配置状态,若在规定时间内配置不命中,才置为空白状态. 对于周期性出现的任务,这种方式能够有效提高预配置的效率.

例 1. 创建任务.

在 SHUM-UCOS 中,软硬件任务的创建采用相同的函数,不加以区别,由创建函数根据优先级判断. 事实上,SHUM-UCOS 中的绝大多数函数都采用这种方式. 若用户希望将某个功能从软件实现迁移至硬件实现时,只需要:(1)修改任务优先级为奇数. (2)提供硬件任务的实现,存储到硬件任务数据库中,其索引将作为硬件任务类型(HCB 中的 OS-HwType 参数).

以创建任务为例说明 SHUM-UCOS 的处理过程,该过程将涉及软硬件任务以及资源管理(设需创建任务 task_i 的实例,其优先级为 prior_i):

1. 判断 prior_i 是否为偶数. 是,则说明用户将创建硬件任务,跳转至步 3;否,则下一步.

2. 调用 UCOSII 的创建任务函数,创建优先级为(prior_i/2)的软件任务,退出.

3. 检索预配置资源链表,如果存在与 task_i 一致的硬件任务,且处于休眠状态(说明已经预配置),则唤醒它,同时修改该资源状态为占用状态,跳转至步 9.

4. 检索配置资源链表,如果存在与 task_i 一致的硬件任务,跳转至步 8.

5. 检索硬件任务预加载表,找到 task_i 所在的 task group.

6. 根据该 task group 的面积,从空白资源链表中选择合适的资源.

7. 配置该资源,并将其从空白资源链表中移出,添加到配置资源链表中.

8. 等待配置完成. 如果成功,将该资源从配置资源链表移出,添加到预配置资源链表中.

9. 初始化硬件任务控制块中的参数.

10. 根据 Hwtask_I 对应的任务接口类型,选择对应的初始化函数对硬件任务初始化.

11. 如果用户创建硬件任务时,要求任务立即运行,则向任务发送运行命令.

### 5.4 硬件任务接口设计

SHUM-UCOS 将硬件任务实现分为 3 层:第 1 层为通信层,用于将非标准时序转换为操作原语层的标准存储器时序,例如将 CAN,I2C 之类的串行通信协议转换为存储器时序,实际上是一个时序转换桥;第 2 层为操作原语层,负责管理任务的运行状态以及提供同步机制;第 3 层为实体层,具体实现用户定义的功能.

SHUM-UCOS 提供通信层和操作原语层,并将它们分别封装为 IP(Intellectual Property),层与层之间采用标准存储器时序访问. 这种方式具有以下优点:(1)简化硬件任务的设计工作,用户在设计实体层时就不必设计过多操作系统细节;(2)减少系统集成时的难度;(3)提高部件的复用程度.

表 1 两类硬件接口比较

| 硬件任务接口 | 通信层接口 | 与 CPU 耦合程度 | 实体层设计难度 | 通信拓扑 | 本地存储器 | 任务间通信方式 |
|---|---|---|---|---|---|---|
| 类型 I | 并行 | 高 | 难 | 总线 | 无 | 全局变量或消息传递 |
| 类型 II | 并行或串行 | 低 | 低 | 星型(点对点) | 有 | 消息传递 |

根据应用的不同,硬件任务的实现也不同,但至少要满足以下条件:

(1)能够响应硬件任务的操作原语,并反馈执行状态. 如对硬件任务进行复位、运行、挂起、恢复、删除的操作.

(2)至少能够采用消息传递方式与 CPU 通信. 在 SHUM-UCOS 中,硬件任务与其它任务间的通信手段包括全局变量和消息传递. 其中消息传递方式是基本方式,包括互斥量、信号量、邮箱. 目前对信号量队列和邮箱队列还未提供支持.

SHUM-UCOS 中提供两类硬件任务接口,可以分别用于实现总线型和星型拓扑结构. 但是在用户设计中,所有用户必须选用一致的硬件任务接口. 其原因来源与下面矛盾:一方面,目前的硬件综合技术还不支持在线综合,FPGA 引脚的功能通常在运行时是固定的①. 另一方面,SHUM-UCOS 需要支持硬件任务动态创建,即任务必须能够在运行时被配置到任意 FPGA 中. 因此要求对于与硬件任务管理器连接的 FPGA,其接入引脚必须一致. 如图 5. 由于两种硬件任务接口占用的引脚数目与功能定义不一致,因此 SHUM-UCOS 不支持两种结构混合.

类型 I 硬件任务接口采用共享存储器方式,通过总线申请能够直接访问存储器. 如图 6(a)所示.

① 尽管通过数据选择器可以在运行时改变单个引脚功能,但当运行硬件任务的 FPGA 与硬件任务管理器之间的引脚数目较多时(特别是涉及到双向总线时),需要实现高维的全连通交叉开关,将增大量额外电路,因此本文不予考虑.

如果系统的主存储器时序与标准时序一致（如SRAM），通信层可以简化或者省略. 操作原语层可分为：(1)数据通路. 与共享存储器连接，使得任务实体能够直接访问数据. (2)控制通路. 与总线仲裁器或 CPU 的 DMA 通道相连，协调总线的使用. (3)初始化通路. 与硬件任务控制器相连，用于在硬件任务创建后，对必要寄存器初始化. 由于该通路上的数据量极小而且只有创建任务时才处于激活状态，应采用串行总线协议以减少对 FPGA 引脚的占用（本文采用 I2C 总线）.

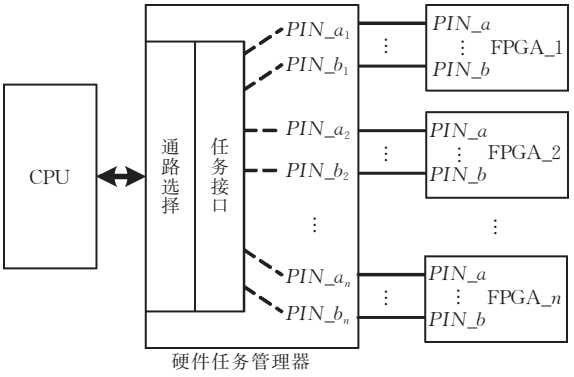
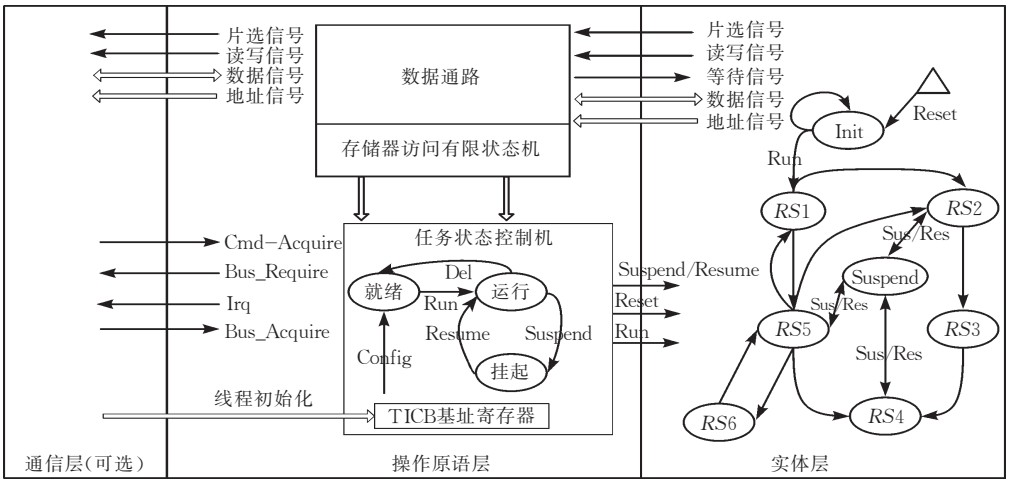
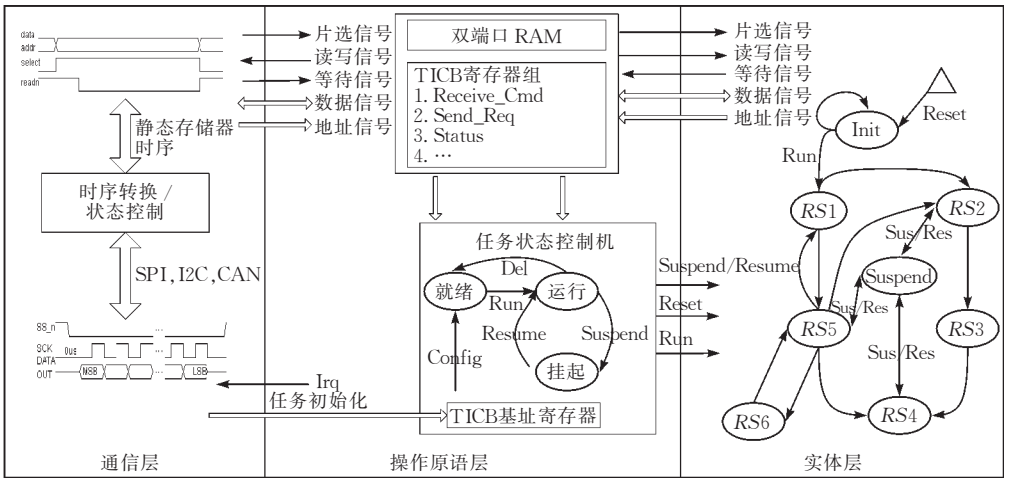



图 5　FPGA 与硬件任务管理器连接




(a) 类型 I 硬件任务接口




(b) 类型 II 硬件任务接口

图　6

每个硬件任务在 SHUM-UCOS 中都拥有相应的任务接口控制块（Task Interface Control Block，TICB），用于定义其控制寄存器. TICB 的具体定义如下.

```
typedef struct os_ticb {
  INT32U Receive_Cmd;   //由 CPU 发给硬件任务的命令
  INT32U Send_Req;      //硬件任务发送给 CPU 的请求
  INT32U Return_Code    //操作返回码
  INT32U Param_Reg      //命令参数寄存器
  INT32U Pointer_Reg    //数据指针寄存器
  INT32U Len_Reg        //数据帧长度寄存器
} OS_TICB;
```

硬件任务被创建后，首先通过初始化通道将任务 ID 和 TICB 的首地址（这些参数在运行时才能得

到)存入硬件任务的寄存器中.通过 TICB 首地址,任务状态控制机能够访问主存储器中的 TICB 数据.

如果 CPU 需要发送命令给硬件任务,首先将命令写入其 TICB 的 Receive_Cmd 参数中,然后置 Cmd_Aquire 信号有效通知硬件任务,最后硬件任务申请总线控制权取得数据.

如果硬件任务需要申请 CPU 的服务,首先将服务种类写入 Send_Req 中,然后用中断通知 CPU,最后 CPU 根据该值选择合适的入口函数处理.

在采用接口类型 I 设计硬件任务时,硬件任务中使用到的全局变量在软件编译完成前,无法确定其在内存中的地址.因此需要首先将该地址定义为常数,在软件编译后再确定,称为变量地址反标.

在类型 I 任务接口中,操作原语层不含本地寄存器和存储器,所有数据,包括 TICB,都保存在主存中.硬件任务与 CPU 之间具有较紧密的耦合关系.但是由于共享存储器,系统内会出现多条与存储器数据总线宽度一致的总线,将占用大量宝贵的布线资源(片内资源或者片外引脚),并且使得线路拓扑复杂.

若硬件任务与 CPU 之间的耦合较松,SHUM-UCOS 提供类型 II 硬件任务接口(与类型 I 硬件任务接口的区别见表 1).其操作原语层中包括双口 RAM 和 TICB 寄存器.这些本地存储器被映射到 CPU 的存储器空间中,其起始地址也在任务创建时写入 TICB 寄存器.类型 II 硬件任务接口中不能直接访问主存,因此实体层不能采用全局变量与其它任务通信.但由于实体层仅访问局部数据,降低了设计难度.

## 6 实验结果及分析

### 6.1 实验平台

在测试与评估 SHUM-UCOS 的过程中,我们采用了复旦大学计算机与信息技术系自主研发的嵌入式快速样机平台.该平台能够提供 4 片 Altera 公司的 FPGA(Cyclone 系列 EP1C6)作为可配置资源.

CPU 采用了 Altera 公司的 Nios 软核 CPU[10].主要是考虑到软核 CPU 具有可裁减、易调试的特点,同时由于软核 CPU 位于 FPGA 内,便于与硬件任务管理器进行集成.

FPGA 采用被动配置方式,其配置时间与配置时钟频率成正比.当配置时钟频率固定时,配置时间为常数.

### 6.2 SHUM-UCOS 性能测试

本文采用 Rhealstone 测试程序集[11]测试 SHUM-UCOS 的性能,并与 UCOSII 的测试结果比较.

Rhealstone 是用于测试实时多任务操作系统的基准程序,分为 6 组,分别用于测试 6 个与实时操作系统密切相关的参数:任务切换时间、任务抢占时间、中断延迟、信号量阻塞时间、死锁解除时间、邮箱传输延迟.在 Rhealstone 中,采用了简化的 Whetstone 测试程序作为测试任务,以屏蔽不同体系结构 CPU 的差异.

UCOSII 中所有任务都采用软件实现,为了保证比较的客观性,SHUM-UCOS 的硬件任务并不实现 WhetStone 测试程序(否则比软件实现执行时间小很多),而只是等待与软件实现相同的时间.

同时,为了评估不同硬件任务实现方式对性能的影响,本文中分别用 3 种方式实现了硬件任务:(1)采用类型 I 硬件任务接口,通过 DMA 方式申请总线使用权;(2)采用类型 II 硬件任务接口,不采用通信层的时序转换,直接将通信原语层接入 Avalon 总线.(3)采用类型 II 硬件任务接口,通信原语层采用 I2C 总线协议,串行时钟频率 1MHz(高速模式).

Rhealstone 第 1 组实验用于测量相同优先级任务间的切换时间.但 SHUM-UCOS 和 UCOSII 一样采用基于优先级的调度,不允许有相同优先级的任务出现.因此该组实验省略.

其它实验条件如下:(1)FPGA. Altera 公司的 EP1C6-8.(2)CPU. Altera 公司的 32 位软核 CPU (NiosII/S),工作频率 50MHz.(3)测试程序. Rhealstone Benchmark,每次邮箱通信传输 10 个字节.(4)测试对象. SHUM-UCOS Ver1.0 和 UCOSII Ver2.76.系统定时周期:1ms.主存储器为 SRAM (IDT71V416).

分析表 2 内的测量结果,可以发现以下特点.

(1)若测量参数是在软件任务之间的,SHUM-UCOS 与 UCOS 的结果基本一致.其原因在于:SHUM-UCOS 是在 UCOS 上扩展得来,保留了其绝大部分数据和程序结构.多数函数一旦发现操作对象中不含硬件任务,就直接调用 UCOS 本身的函数.因此,对于软件任务的操作,SHUM-UCOS 与 UCOS 基本一致.

(2)对于同一个参数,按照交互对象,测试结果由小到大的顺序如下:软件任务之间,硬件任务与软件任务之间,硬件任务之间.其原因在于:在 SHUM-UCOS 中,任务间的通信与同步控制都是软件处理的.如果软件任务获得消息,只需相应任务控制块的标志即可.对于硬件任务,则需要进行触发中断、申请总线等操作,因此,硬件任务越多,通信延迟越大.然而这种情况是可以改善的,如果通过操作系统协

同设计，将一部分同步原语（如信号量、互斥量）用硬件实现，就能够提升操作系统的性能.

（3）对同一类测试，按照硬件任务接口的接入方式，测试结果由小到大的顺序如下：直接接入总线的类型 II 硬件任务接口，类型 I 硬件任务接口，采用 I2C 串行总线的类型 II 硬件任务接口. 其原因在于：当类型 II 硬件任务接口接入总线时，类似与 CPU 的外设，自身能够判断 CPU 写入的命令. 而 CPU 与类型 I 硬件任务接口通信时，需要经过写-通知-读的过程，因此延迟较大. 而采用 I2C 串行总线的类型 II 硬件任务接口，其通信延迟还包括串并转换与控制开销，因此延迟最大.

表 2　SHUM-UCOS 性能测试及与 UCOSII 性能比较　　（单位：us）

| 测试项目 | | 任务抢占时间 | 中断延迟 | 死锁解除时间 | | | 信号量阻塞时间 | | | 邮箱传输延迟 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SHUM-UCOS | 实现方式一(2) | 78.974 | 0.631 | 133.792 | 186.121 | 251.720 | 104.351 | 159.526 | 206.383 | 114.804 | 252.970 | 367.253 |
| | 实现方式二(1) | 78.631 | 0.631 | 132.870 | 213.417 | 301.032 | 99.819 | 181.702 | 287.003 | 114.172 | 272.334 | 403.861 |
| | 实现方式三(3) | 79.053 | 0.631 | 133.611 | 459.583 | 814.278 | 102.007 | 426.203 | 787.865 | 115.036 | 759.425 | 1214.601 |
| UCOSII | | 76.380 | 0.631 | 131.810 | 无 | 无 | 101.403 | 无 | 无 | 113.865 | 无 | 无 |
| 备注 | | 软件任务之间 | 软件任务之间 | 软件任务之间 | 硬件任务与软件任务之间 | 硬件任务之间 | 软件任务之间 | 硬件任务与软件任务之间 | 硬件任务之间 | 软件任务之间 | 硬件任务与软件任务之间 | 硬件任务之间 |

注：(1)采用类型 I 硬件任务接口；(2)采用类型 II 硬件任务接口，Avalon Bus Slave；(3)采用类型 II 硬件任务接口，通信层采用 I2C 总线.

### 6.3　应用实例

在与某研究所合作研发的调度终端设备中，我们采用了 SHUM-UCOS 作为操作系统，并通过该实例评估了 SHUM-UCOS 的性能. 该调度终端的作用是把来自受话器的用户语音封装成 RTP 数据包发往要求的网络地址，同时从网络接收发给自己的话音 RTP 数据包转换为语音放出. 为能实现各种调度功能，调度终端需要接收来自调度控制台的命令，并根据用户在终端面板上的按键指令完成各项工作. 其功能框图如图 7.

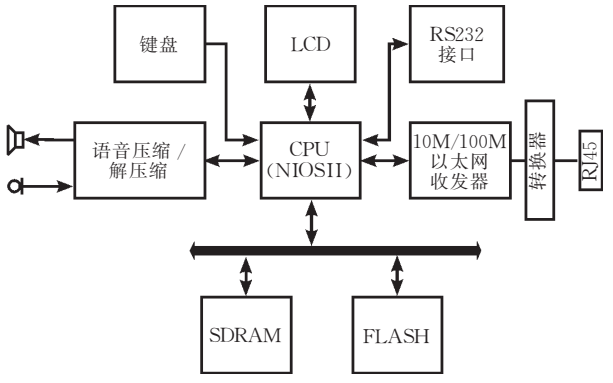



图 7　调度终端

在该应用中，语音压缩与解压缩是重要的计算类任务，其效果直接影响系统性能，如信噪比、传输延迟、占用带宽等. 因此在设计时，我们首先用软件任务实现压缩与解压缩，然后再用硬件任务方式实现，比较二者性能差异. 语音应用中通常都会采用数据缓冲，以便进行防抖、数据排队等操作，因此设计时采用了类型 II 硬件任务接口，硬件任务每采集一帧数据，就通过邮箱发消息给 CPU. 为了能够体现出性能差异，我们采取了以下措施：(1)选用 G.726 语音压缩标准[13]. 该标准采用自适应脉冲编码调制方式(ADPCM)压缩，压缩率可调. 这样通过调整压缩率，可以逐步增大系统的数据处理量. (2)设置 CPU 的主频为较低的 15MHz，当 CPU 无法处理当前数据帧时，就将其丢弃. 通过丢帧率考察系统性能. 表 3 是采用不同压缩率和实现方式所得的测试结果.

表 3　不同实现方式下的丢帧率

| 压缩方式 | 软件任务实现(%) | 硬件任务实现(%) |
|---|---|---|
| G.726 ADPCM(16K) | 0 | 0 |
| G.726 ADPCM(24K) | 17.94 | 0 |
| G.726 ADPCM(32K) | 41.36 | 3.47 |
| G.726 ADPCM(40K) | 57.81 | 11.20 |

可以看到，采用硬件任务实现后，系统性能显著提升. 但客观地看，几乎任何系统功能在从软件实现到硬件实现后，都会对性能提升有所贡献，贡献幅度取决于该功能在系统中的重要性：越是系统性能瓶颈，迁移后获益越大. 但在采用 SHUM-UCOS 后，对系统的其它任务只需极少改动就可以实现任务的自然迁移. 在该实例中，在压缩算法的迁移过程中，除了添加任务管理器以外，对于其它软件任务共修改了 13 处.

## 7　结论和未来工作

本文在轻量级实时操作系统 UCOSII[14] 的基础上，采用统一任务模型重新设计了能够同时管理软

硬件任务的操作系统 SHUM-UCOS. 该操作系统具有以下特点:(1)支持硬件任务的动态创建,拓展了统一任务模型的适用范围. (2)采用资源管理器对 FPGA 等可重构资源进行统一管理. (3)通过硬件任务预配置表提前创建硬件任务,以减少其创建时的阻塞,提高并行性. (4)提供标准硬件任务接口与实现. 实验结果表明,该操作系统是有效的,能够管理可重构资源、提高系统性能,同时缩短同一任务由软件实现到硬件实现的迁移时间.

但该操作系统仍然存在一些有待研究和改进的问题,如支持混合硬件任务接口和混合类型 FPGA、减少任务间的通信延迟、采用动态资源预配置、提供邮箱队列支持等,这些都是我们未来需要进一步研究的问题.

## 参 考 文 献


1 Lee E.. Overview of the Ptolemy Project. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, CA, USA, 2003

2 Alexander P., Kong C.. Rosetta: Semantic support for model centered systems level design. Computer, 2001, 34(11): 64～70

3 Andrews D., Niehaus D.. Programming models for hybrid FPGA-CPU computational components: A missing link. IEEE Transactions on Micro, 2004, 24(4): 42～53

4 Walder H., Platzner M.. Reconfigurable hardware operating systems: From design concepts to realizations. In: Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA'03), Las Vegas (NV), USA, 2003

5 The ISO POSIX Working Group. ISO/IEC 9945: 2002 POSIX Standard, 2002

6 Donthi S., Haggard R. L.. A survey of dynamically reconfigurable FPGA devices. In: Proceedings of the 35th Southeastern Symposium on System Theory, Morgantown, West Virginia, USA, 2003, 422～426

7 Kwork Y. K., Ahmad I.. Dynamic critical-path scheduling: An effective technique for allocation task graphs to multiprocessors. IEEE Transactions on Parallel and Distributed System, 1996, 7(5): 506～521

8 Karthikeya M., Purna G., Bhatia D.. Temporal partitioning and scheduling data flow graphs for reconfigurable computers. IEEE Transactions on Computer, 1999, 48(6): 579～590

9 Cormen T. H., Leiserson C. E.. Introduction to Algorithms. Cambridge, MA: The MIT Press, 2001, 1043～1054

10 Peng Cheng-Lian, Zhou Bo. SOPC Design and Practice Using NIOS. Beijing: Tsinghua Press, 2004(in Chinese)
(彭澄廉,周 博等. 挑战 SOC-基于 NIOS 的 SOPC 设计与实践. 北京: 清华大学出版社, 2004)

11 Kar R. P.. Implementing the rhealstone real-time benchmark. Dr. Dobb's Journal, 1990, 15(4): 46～55

12 Levine D. L., Flores-Gaitan S.. An empirical evaluation of OS support for real-time CORBA object request brokers. In: Proceedings of the Real-Time Technology and Applications Symposium(RTAS), Vancouver, British Columbia, Canada, 1999, 38～47

13 The International Telecommunication Union, ITU-T G.726 Recommendations, 1990

14 Labrosse J. J.. Micro/OS-II The Real-Time Kernel, Second Edition. CMP Books, 2002



**ZHOU Bo**, born in 1979, Ph. D. candidate. His research interests are reconfigurable computing, evolvable hardware and intelligent optimization algorithm.

**WANG Shi-Ji**, born in 1979, Ph. D. candidate. His research interests are high performance computing, UWB communication and satellite communication.

**QIU Wei-Dong**, born in 1972, Ph. D.. Her research interests are rapid embedded system prototype and RTOS modeling and implement.

**PENG Cheng-Lian**, born in 1941, professor, Ph. D. supervisor. His research interests are software-hardware codesign, distributed system and fault-tolerant computer system.


**Background**

Attracted by the flexibility, computing ability and low power potential of RC(reconfigurable computing), the authors have focused their works on this interesting area for a long while. Their research interests about RC include task-graph partition, scheduling and system modeling and have published several papers in international conferences and journals in recent years. This paper is a piece of the technical report on "operating system support of RC", which main aim is to provide engineers familiar and easy-to-use programming environment while utilizing RC's merits. Fortunately, the proposed RTOS framework has achieved most of their design objects. Moreover, combined with the works on system codesign, the proposed RTOS is proved as a powerful and adaptive platform.