

一种支持可重构混成系统的操作系统设计与实现

乔磊 齐骥 龚育昌

(中国科学技术大学计算机科学技术系 合肥 230027)

摘 要 可重构硬件和指令集处理器构成的混成系统兼有运算速度高和编程灵活的优点,是近年来学术界研究的热点.已有的面向该类系统的操作系统由于受到传统抽象模型的制约,不能充分发挥可重构硬件的优势.文中在分析该类系统对操作系统的需求和已有运行模型缺陷的基础上,提出了一种基于服务体/执行流模型的操作系统 SEF-OSHRs.它具有统一的系统对象抽象和通信接口,并可支持控制流的直接转换,因而能充分发挥混成系统的优势.文中详细介绍了该操作系统的基本抽象、系统结构和运行方式,并通过实验结果说明了该系统的可用性和高效性.

关键词 可重构混成系统;操作系统;服务体;执行流

中图法分类号 TP302

DOI号: 10.3724/SP.J.1016.2009.01046

An Operating System for Hybrid Reconfigurable Systems: Design and Implementation

QIAO Lei QI Ji GONG Yu-Chang

(Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027)

Abstract Hybrid systems that are composed of reconfigurable hardware and instruction set processors can achieve high performance as well as flexible programmability simultaneously. However the existing operating systems (OS) cannot fully utilize the reconfigurable computing resources in such systems. By exploiting a novel OS construction model SEFM (Servant & Execution-Flow Model), this paper presents an OS named SEF-OSHRs, which has a uniform system object abstraction model and inter-servant communication interface. SEF-OSHRs can transfer the control flow directly along with the inter-servant communication and thus make full use of the advantages of hybrid reconfigurable systems. The fundamental abstraction model, system architecture and run-time environment are discussed in detail. The usability and efficiency are proved by experiments.

Keywords hybrid reconfigurable system; operating system; servant; execution-flow

1 引 言

可重构混成系统(以下简称混成系统)既可发挥专用集成电路 ASIC(Application Specific Integrated

Circuit)速度上的优势,又具有指令集处理器 ISP (Instruction Set Processor)的灵活性,近年来在嵌入式领域已有广泛的应用.支持可重构混成体系结构的操作系统 OSHRS(Operating System for Hybrid Reconfigurable System)能够支持可重构资源

的动态管理,并提供基本的运行环境和编程模型,从而可更有效地利用可重构计算资源.图 1 是由部分可重构 FPGA(Field Programmable Gate Array)和 ISP 等器件构成的混成系统结构示例,本文所讨论的操作系统即基于该种混成结构.所谓部分可重构,是指 FPGA 可在运行时动态改变某一部分的配置,而器件上其余未更改部分仍按原有方式正常工作.

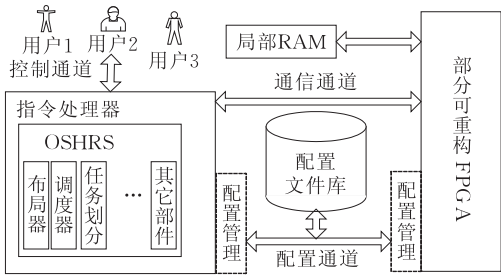


图 1 可重构混成系统结构

Brebner^[1]是最早在操作系统方面对可重构技术进行研究的人之一,他提出的虚拟硬件资源的概念类似于传统操作系统中的虚存,将可重构资源划分成类似于“页”的、具有固定面积和输入/输出的互联单元.该方案大大降低了系统的灵活性,资源浪费和碎片现象也很严重;文献[2]着重讨论了可重构操作系统应提供的基本服务,如任务划分、任务布局、任务间布线等,并用时间复杂度相对较低算法加以实现.但未对运行模型做出明确说明;文献[3]通过扩展 Linux 操作系统的方法,实现了一个 OSHRS 的原型,其目的是使 ISP 端的应用程序和 FPGA 端的逻辑模块都受到支持,并可自由在混成系统中迁移.可重构资源被组织为一系列 ICN(Inter-Connection Network)单元,每个 ICN 单元在操作系统看来相当于一个跟 ISP 异构的处理器.该操作系统采用进程/线程作为基本运行对象,并维护与进程相关的原语.但该操作系统中存在很多与所谓“硬件任务”无关的数据和控制域,浪费了系统资源,且增加了维护的难度.由于大部分的任务间通信操作都会引起进程调度,因此开销很大.此外,用可重构硬件实现进程/线程相关原语和维护相关语义使操作系统复杂度大大增加;文献[4]提出了一种基于统一多任务模型可重构系统的实时操作系统,但该系统采用静态方法生成任务配置表和调度任务,要求预先知道“硬件任务”的数据依赖关系等信息,灵活性不高,而且任务间通信采用进程的通信原语,效率较低;NIOS[®]是 Altera 公司推出的软核嵌入式 RISC 处理器,但无法支持动态可重构.综上所述,已有的

OSHRs 或者没有明确指出运行模型,或者沿用传统的进程/线程模型 PTM(Process/Thread Model),然而该模型以及该模型相关的进程间通信机制不能有效支持混成系统.

本文介绍了一种基于服务体/执行流模型 SEFM(Servant & Execution-Flow Model)^[5-7]的 OSHRS. SEF-OSHRs 以同步通信方式作为系统对象间通信的主要手段,控制流的转换不依赖于调度,且对软、硬件应用采用统一的抽象,可有效地支持混成系统.文章的最后通过实验证明了 SEF-OSHRs 的可用性和高效性.

2 OSHRS 的结构与机制

可重构硬件的运算方式与 ISP 有着显著区别.以电平触发逻辑为例,当硬件逻辑模块在输入端有电平信号的变化时,就会触发一定的逻辑功能,完成对特定输入的处理,然后产生输出信号.可见,硬件逻辑模块间具有直接控制流转换的能力,控制流转换不需要,也没必要引入额外的开销.而在传统的 PTM 操作系统中,程序间的函数调用、参数传递等直接关系被共享内存、管道、消息等进程间通信机制所封装,以至控制流的转换需要通过调度间接完成,开销很大.

运行模型应该站在控制流的角度对系统的底层机制进行准确的抽象,以便更加有效地利用硬件平台提供的计算资源.执行流和服务体是 SEFM 的两个基本抽象.基于 SEFM 的 OS 较单内核结构(Monolithic kernel architecture)OS 具有更好的模块化特性和空间隔离机制,较微内核结构(Micro-kernel architecture)OS 具有更高的系统对象间通信效率^[7].本文提出的 SEF-OSHRs 根据混成系统的特点,对 SEFM 进行了扩充,提出了硬服务体的概念(图 2),并进而将运行于 ISP 端的服务体称为软服

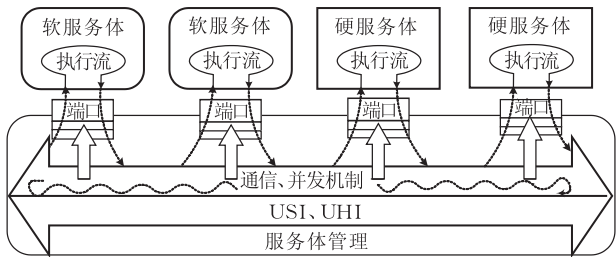


图 2 服务体/执行流模型示意图

务体,将运行于FPGA端的服务体称为硬服务体.操作系统中地各功能模块(如在ISP端运行的文件系统、内存管理器、驱动程序以及在FPGA端运行的加密/解密、视频压缩等逻辑电路)都以服务体的形式存在.基于SEFM的OSHS逻辑结构如图2所示.

2.1 软服务体相关机制

2.1.1 基本抽象

对ISP而言,执行流是机器执行指令能力最原始的抽象,它是连续的概念,从系统上电复位开始到关闭时结束,中途不会被阻塞、挂起和停止.每个ISP提供一个执行流,若采用超线程技术则每个ISP可提供多个并行执行流.执行流不与固定的地址空间绑定,可自由跨越系统组件的边界,其作用是推动服务体进行消息处理.软服务体是具有通信功能、拥有地址空间的能够完成某种功能的代码、数据集合,它是静态的概念,其生命周期不依赖于执行流,具有持久性.用户程序(包括驱动程序)和系统模块均以服务体的形式存在,各服务体之间以端口作为通信的接口,并在执行流的推动下进行消息处理.

核心服务体在SEFM中是一个关键的组件,它在逻辑上相当于传统操作系统中的内核,但它不需要运行在内核态.核心服务体提供服务体间通信机制、服务体/执行流的管理、中断/异常等基础服务,它还维护系统关键抽象(如服务体等)的数据结构.

2.1.2 通信机制

端口是服务体对外通信的唯一接口,记录特定消息处理例程的入口.服务体由端口向外界提供服务,执行流只能从端口进入服务体.一个端口辖括若干小端口,用于记录执行流通过端口进入服务体时的上下文信息和一些关键的机器状态,主要包括一个堆栈和一个寄存器快照结构.消息是服务体间信息交换的载体,具有确定的格式,由消息头部和消息体组成.消息头指明消息的目的端口等信息,消息体用于携带数据.按照面向对象的观点,服务体可看作一个类,“发送消息”是这个类的方法,端口是一个公共的成员,小端口则是与端口相关的私有成员.

消息推动机制中基本通信原语的编程接口为

```
msg_body_t * msg_push(msg_body_t * hdr,
                        msg_option_t option);
```

msg_push()有3种消息发送方式,分别为

(1)同步-连续方式.以此种方式发送消息后,执行流立即进入目标服务体,待完成服务后返回发送方继续执行,由msg_push()返回应答的消息.该过程相当于一次受保护的跨地址空间函数调用.

(2)同步-分离方式.发送消息后,执行流立即进入目的服务体,完成服务后应答到指定的端口.类似于尾递归优化.

(3)异步方式.发送消息后,执行流仍在本服务体内,消息被异步地处理和应答.该方式提供了异步语义.

2.1.3 并发和中断机制

核心服务体维护一个全局小端口调度队列,根据一定的原则,从这个队列中挑选一个小端口并将执行流派发到其中.被选中的小端口为其所属的端口提供了运行环境,支持服务体进行消息处理.当该消息处理结束时:(1)若不需应答而又没有发送新的消息,则通过向核心服务体发送无应答端口的同步分离消息放弃执行流.核心服务体收到消息后,继续选择合适的小端口派发执行流.(2)否则,按照代码序列和消息推动的方向继续执行.

事务端口是为了执行特定的任务创建的,不作为提供服务的接口,它在数据结构上与普通端口没有区别.创建事务端口后,系统将其辖括的某一小端口置入全局调度队列,调度器在适当的时机将执行流派发到该小端口中.事务端口在语义上代表一个事务的执行,系统的并发实际上就是事务的并发.

如图3所示,消息的发送或事务相关小端口时间片的过期推动ISP端执行流在各服务体间流动.可见系统中仅有一个执行流.

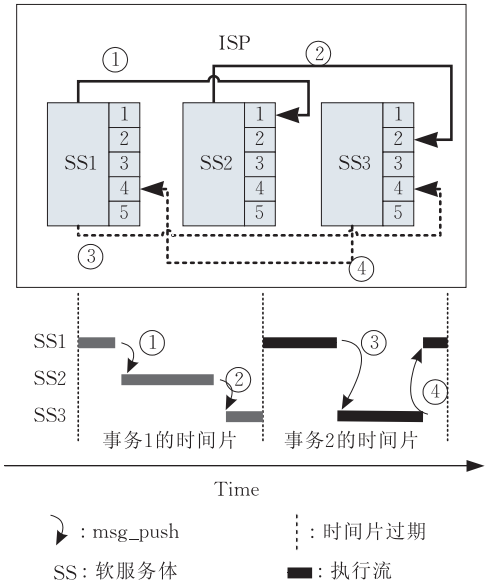


图3 ISP端并发示意图

SEFM的中断相当于硬件在当前的代码序列中随机插入一条指令,这条指令的作用相当于向核心服务体发送一条同步连续的中断消息.核心服务体

则根据该消息触发中断服务例程,中断完成后返回发生中断的服务体继续执行。

关于 SEFM 在 ISP 端工作机制更详细的说明,请参考文献[5-7]。

2.2 硬服务体相关机制

2.2.1 基本抽象

对 FPGA 而言,执行流是各逻辑模块运算能力的抽象,多个执行流可并行执行,其作用就是推动硬服务体完成消息处理。硬服务体是指拥有可重构计算资源、经过内部综合并具有固定的形状、面积和时间特性的逻辑模块,它是对 SEFM 中软服务体概念面向 OSHRS 的扩充。硬服务体的存储与加载位置无关,即 OSHRS 可将硬服务体在 FPGA 上重新定位,它的时间特性包括硬件逻辑的时钟频率、时序特征等。

2.2.2 统一软件接口、统一硬件接口

统一软件接口 (Uniform Software Interface, USI)、统一硬件接口 (Uniform Hardware Interface, UHI) 是可重构平台的支撑协议,为混成系统中的软、硬件应用提供统一操作接口,协助完成重构配置、服务体间通信等功能,在 OSHRS 中所起的作用类似于传统操作系统中的体系结构相关层。USI 提供的接口有:发送/接收数据和控制指令、拷贝硬件逻辑模块比特流到 FPGA、修改 FPGA 芯片上特定位置的配置信息、对硬服务体重定位等;UHI 主要实现对不同体系结构 FPGA 的抽象和提供硬件逻辑模块间通信的方法。USI 和 UHI 将操作系统与底层机制分割,给 OSHRS 提供了最大限度的平台无关特性,也使 OSHRS 的设计者摆脱复杂底层机制的困扰。USIS (USI 服务体) 和 UHIS (UHI 服务体) 分别为 USI 和 UHI 协议在 OSHRS 中的实现,他们所处层次以及在消息发送过程中的作用如图 4 所示,可见由软服务体向硬服务体发送消息需通过 USIS 和 UHIS。

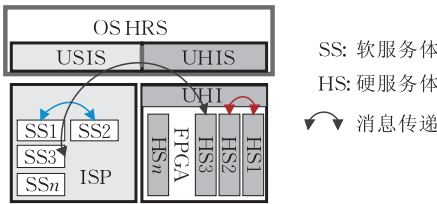


图 4 通过 USIS、UHIS 发送消息

2.2.3 通信机制

OSHRS 对软、硬服务体采用统一通信接口 (包括消息发送原语、消息结构等),以实现更好的模块

间独立性和可扩展性。服务体间的通信有如下 3 种可能:(1) 软服务体之间发送消息:此种情况没有 USIS、UHIS 的参与,其过程详见文献[5-7]中相关叙述;(2) 软服务体向硬服务体发送消息,流程如图 5 所示;(3) 硬服务体发送消息,流程如图 6 所示。为简单起见,仅示出同步分离消息通信且不需要应答的情形。

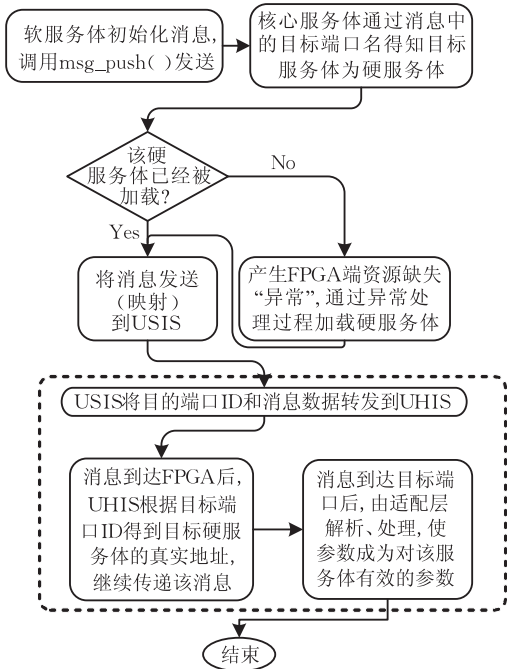


图 5 软服务体向硬服务体发送消息流程图

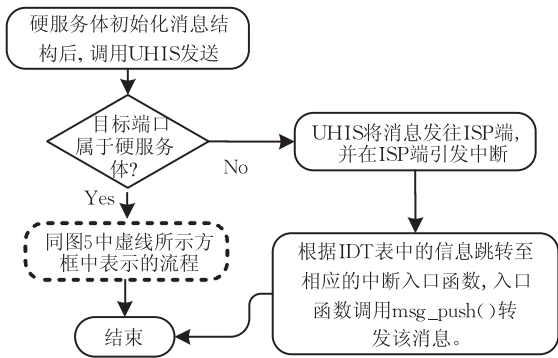


图 6 硬服务体发送消息流程图

2.2.4 其它重要机制

由于 ISP 与 FPGA 运行方式的差异,OSHRS 需要专门提供可重构资源的管理和硬服务体加载、卸载等重要机制。

调度和布局

OSHRS 中调度器的作用为:(1) 决定各软服务体小端口获得执行流的时机;(2) 为硬服务体分配可重构计算资源,决定硬服务体的加载时机;(3) 安排硬服务体处理消息的顺序。布局器是调度器的一

个重要部件,它管理芯片上可重构计算资源,解决运行时资源分配问题. 调度和布局在 ISP 端完成.

图 7 所示调度器与布局器的工作过程如下:①系统欲加载硬服务体时,调度器询问布局器 FPGA 上是否有足够空间容纳新的硬服务体;②布局器以该硬服务体需占用面积为参数尝试分配空间;③若成功,则返回该硬服务体被布局的位置;④⑤通过 USIS 加载该硬服务体;⑥若失败,则创建一个事务端口,该事务端口描述了加载硬服务体所需的操作. 核心服务体将与该事务端口绑定的小端口置入调度队列,等待执行流进入;⑦待成功布局硬服务体后,事务端口向核心服务体发送消息请求撤销本端口.

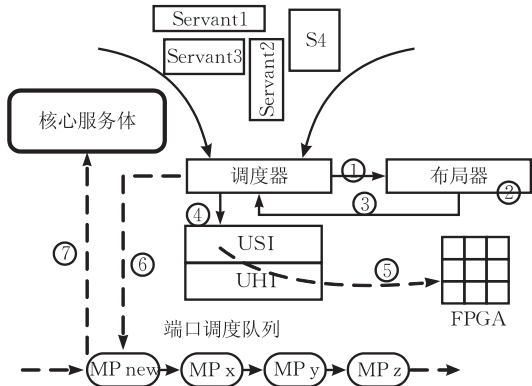


图 7 调度器与布局器工作示意图

碎片整理

OSHRs 运行过程中,可重构资源被频繁地分配/回收,这使得 FPGA 的空闲空间分布趋于离散,互不连续,即使 FPGA 上空闲总面积大于某些待加载的模块,亦不能成功加载. 此种现象被称为可重构芯片的碎片化,严重时会使 OSHRS 陷于瘫痪.

在传统 ISP 系统级软件中,使用内存分配/回收的算法来尽可能地减少内存碎片. 比如 Java 虚拟机中的垃圾回收器就负责把不再被引用的内存对象释放,并整理存在于内存对象间隙的碎片. 在 OSHRS 中,负责实现碎片整理的组件称为碎片整理器. 碎片整理器一般在可重构芯片处于空闲状态时(即没有硬服务体在处理消息时)被触发,进行碎片的整理工作,以求有效提高可重构资源的利用率,又不降低系统工作效率.

硬服务体的加载和卸载

硬服务体初始以库的形式存储于非易失存储介质,其比特流有两种加载模式:

(1) 显式加载. 用户显式的向核心服务体发送消息,请求加载硬服务体. 核心服务体得到消息后利

用 USIS 提供的 FPGA 配置端口完成加载.

(2) 隐式加载. 用户向未加载的硬服务体发送消息,产生“硬服务体缺失”异常,由异常处理程序将该服务体加载,处理过程如下:①若触发异常的消息为同步消息,异常处理代码向 USIS 报告缺失的硬服务体 ID,USIS 利用配置端口将该服务体的比特流从存储部件配置到 FPGA;②若该消息为异步消息,核心服务体立即创建一个事务端口,该事务端口负责完成加载硬服务体的任务并转发消息. 核心服务体将该事务端口的小端口置入调度队列,等待调度;③执行流返回发送方服务体继续执行后续代码. 当上述小端口获得执行流时,才真正执行加载过程,加载完毕后事务端口被撤销.

与加载情况类似,硬服务体的卸载也有显式和隐式两种方式. 显式卸载向核心服务体发送卸载消息;隐式卸载即下文所述的硬服务体换出.

硬服务体换出

由于可重构计算资源有限,当系统负载较重时,布局器将无法找到足够空闲加载新的服务体,此时必须将某些不再处理消息的硬服务体从 FPGA 上换出. 本文采用一种类似 LRU 的方法选择适合换出的硬服务体,即首先计算每个硬服务体的“面积加权年龄”AAQ(Area Age Quality),然后选择 AAQ 值最高的换出. AAQ 的计算公式为 $AAQ = Area \times Age$. 其中 $Area$ 是硬服务体的面积, Age 是该硬服务体近期被访问情况的估计. 与传统的多道程序操作系统中任务换出的操作不同,硬服务体的换出仅需更新布局器中的空闲资源列表.

2.2.5 OSHRS 的工作方式

OSHRs 启动时,首先对混成系统中的各硬件进行初始化,并装载核心服务体. 核心服务体引导执行流加载其他所需软、硬服务体,进而将执行流引入相应的服务体中进行各自的初始化,此后执行流又被交还给核心服务体.

从执行流的角度看,执行流最初是从 ISP 端的核心服务体出发进入软服务体的,当软服务体向硬服务体发送消息时,执行流才有机会从 ISP 端进入 FPGA 端. 比如,软服务体发送同步连续消息给硬服务体之前,ISP 端可能存在一个或多个执行流(数目与 CPU 的数目一致),该消息发送过后,则 ISP 端的执行流数目减一,FPGA 端执行流数目加一. 当硬服务体完成对消息的处理后,FPGA 端发送应答消息,将执行流返还给 ISP 端.

3 实现与测试

MiniCore^[5-7]是一个已实现了的基于 SEFM 的操作系统,我们通过裁减和扩充 MiniCore 来构造 SEF-OSHRs.如图 8 所示,SEF-OSHRs 的基本机制层提供了系统运行所需要的基础机制,特别是用于支持硬服务体抽象的加载/卸载、调度/布局服务体.服务层提供了基本的系统服务功能.应用层运行用户提交的程序.系统中各种组件都是以服务体的形式实现,通过统一的服务体间通信机制进行交互,因此耦合程度低,灵活性高,易于维护和扩展,特别适用于混成计算环境.

本节分别测试了 SEF-OSHRs 的几项基础参数和若干基于 SEF-OSHRs 应用的性能,并对实验结果进行了分析.

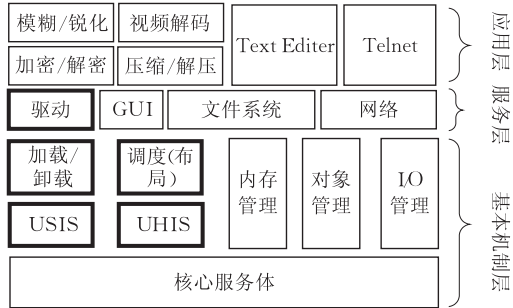


图 8 SEF-OSHRs 逻辑结构图

3.1 实验平台与测试工具

我们开发了 One-Recon 可重构实验环境对 SEF-OSHRs 的性能进行测试.该实验环境由两部分构成,分别是 Xilinx 公司提供的 XUP Virtex II Pro 开放平台^①和研扬(AAEON)的 PCM-6892 工控开发板,两者之间通过 100Mbps 高速以太网相连(如图 9 所示).

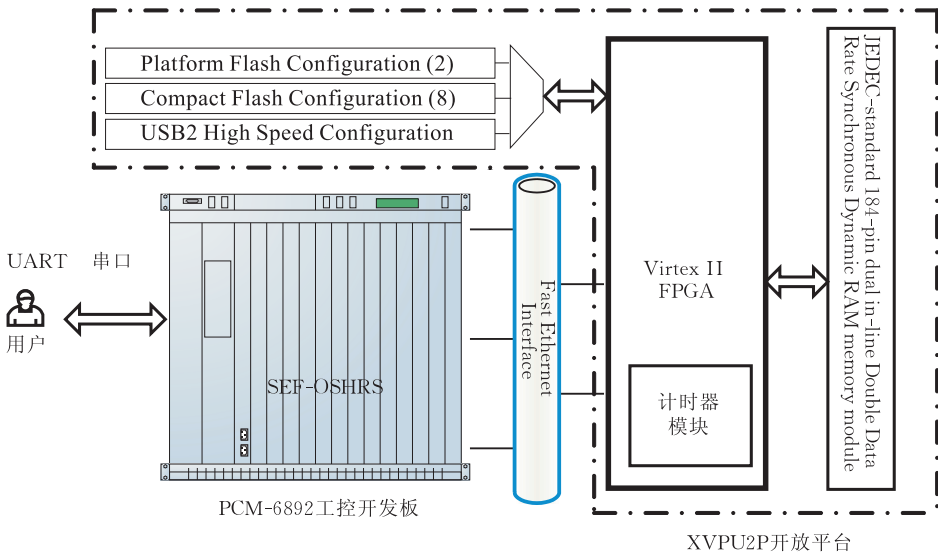


图 9 实验平台示意图

为测试性能,硬件上要提供精确计时的手段.对于 PCM-6892 上的 C3 CPU,可通过 rdtsc 汇编指令读取 TSC 寄存器的值进行计时,其精度可达 1ns.实现 FPGA 端的计时需要增加额外的模块,我们在 XVP2P 的 Virtex II FPGA 中设计了一个计时器模块,计时精度为 5ns.

在 3.2 节的各项测试中,SEF-OSHRs 运行于 One-Recon 可重构实验环境,Linux 运行于如表 1 所示的环境.

表 1 对比实验中 Linux 的运行环境

操作系统	实验平台	开发环境
Fedora Core II (Linux Kernel 2.4.18)	Intel PIII 1.0GHz+ 256MB Main Memory	GCC 2.96

3.2 性能测试与结果分析

3.2.1 SEF-OSHRs 基础参数测试

本节对 SEF-OSHRs 中服务体间通信开销和服务体加载延迟进行测试.由于 Linux 等操作系统不直接实现同步通信语义(而是通过异步原语对同步通信进行模拟),所以测试 1 中仅将 SEF-OSHRs 的测试结果与 MiniCore 的测试结果相比较.

测试 1. 对同步连续方式的通信效率进行评估.测试方法是编写两个服务体 A 和 B,A 得到执行流后,以同步连续方式向 B 发送消息,重复测试 100 次.

① Xilinx Inc. Xilinx XUP Virtex II Pro Development System.
<http://www.xilinx.com/univ/xupv2p.html>

表 2 测试 1 的结果 (单位: μ s)						
	SEF-OSHRs			MiniCore		
	Max	Min	Avg	Max	Min	Avg
软-软	1. 849	1. 804	1. 827	1. 725	1. 557	1. 641
软-硬	18. 117	17. 763	17. 940	—	—	—
硬-硬	0. 133	0. 133	0. 133	—	—	—
硬-软	17. 563	17. 261	17. 412	—	—	—

注:其中“软-软”表示软服务体向软服务体发送消息,“软-硬”和“硬-硬”同理. Max 是 100 次重复测试中的最大值,Min 是最小值,Avg 是平均值,表中打“—”的项目代表无法测得的数据.

由测试 1 的结果可知:(1) SEF-OSHRs 与 MiniCore 中“软-软”方式同步连续消息发送开销基本一致. 其原因在于:SEF-OSHRs 由 MiniCore 扩展而来,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服务体,就直接调用原 MiniCore 中的函数. (2) “软-硬”与“硬-软”方式的开销基本一致,它们都大大高于“软-软”方式. 其原因在于:两种方式的消息跨越软硬件边界都经过了一次 USI/UHI 解析,产生的延迟基本一致. 而跨越软硬件边界时申请总线和中断等耗时动作使得这两种方式的耗时大大高于“软-软”方式. (3) 硬服务体之间发送消息的速度最快. 其原因在于:UHI 实现了所有硬服务体间消息发送的相关原语.

测试 2. 对同步分离方式的通信效率进行评估. 测试方法同样是编写两个服务体 A 和 B,A 得到执行流后,以同步分离方式向 B 发送消息,重复测试 100 次.

表 3 测试 2 的结果 (单位: μ s)						
	SEF-OSHRs			MiniCore		
	Max	Min	Avg	Max	Min	Avg
软-软	2. 772	1. 436	2. 104	2. 141	1. 906	2. 023
软-硬	21. 383	19. 221	20. 302	—	—	—
硬-硬	0. 133	0. 133	0. 133	—	—	—
硬-软	20. 605	19. 089	19. 847	—	—	—

注:发送同步连续消息时,目标服务体接收到消息后,马上获得执行流进行消息处理,而后将执行流通过不带应答端口的同步分离消息返回给源服务体. 因此,测试 1 即相当于完整同步分离消息的前半程,而测试 2 则为后半程.

观察表 3 中的结果可发现与测试 1 类似的特点. 综合测试 1 和测试 2 的结果,我们知道在 SEF-OSHRs 中,服务体间的通信开销波动较大,而且每当消息跨越软-硬件边界的时候就会产生非常大的开销. 因此,当程序员在 SEF-OSHRs 中设计具体应用时,应该慎重使用跨越软-硬件边界的消息. 其它情况下服务体间的通信开销都很小.

测试 3. 对异步消息的通信效率进行评估. 分别编写两个服务体 A 和 B, A 得到执行流后以异步方式向 B 发送一个消息. 重复测试 100 次. 对于其他

操作系统,则创建两个线程 A 和 B 相互发送消息.

表 4 测试 3 的结果 (单位: μ s)				
操作系统	通信方式	Max	Min	Averag
RedHat 8. 0	Pipe	39. 38384	38. 65657	39. 01111
	Unix	51. 99327	49. 75084	50. 65387
	TCP	378. 2155	343. 8316	355. 9232
VxWorks	Pipe	23. 42424	23. 08754	23. 37104
	TCP	290. 3165	286. 165	287. 5882
Windows 98	Pipe	617. 6771	590. 858	595. 8866
	TCP	1701. 336	1622. 555	1652. 475
MiniCore	异步	36. 7592	35. 4270	36. 0931
SEF-OSHRs	软-软 异步	37. 8324	37. 0109	37. 4217
	软-硬 异步	61. 9917	57. 6965	59. 8441
	硬-硬 异步	—	—	—
	硬-软 异步	63. 2953	58. 7597	61. 0275

观察测试 3 的结果可知,SEF-OSHRs 的“软-软”异步消息与 MiniCore 的异步消息具有基本相同的开销,它们与 Linux 的管道(Pipe)方式效率相当.

测试 4. 对软、硬服务体的加载延迟进行评估. 测试方法是编写两个服务体 A 和 B,A 得到执行流后加载 B.

表 5 测试 4 的结果 (单位: ms)						
	SEF-OSHRs			MiniCore		
	Max	Min	Avg	Max	Min	Avg
软-软	3. 84	4. 46	4. 15	3. 93	4. 31	4. 12
软-硬	43. 26	52. 44	47. 85	—	—	—
硬-硬	7. 98	7. 98	7. 98	—	—	—
硬-软	4. 42	5. 01	4. 71	—	—	—

注:其中“软-软”代表软服务体的事务端口负责加载硬服务体,依此类推. 待加载的硬服务体为 FIR 滤波器,规模为 398. 76KByte,待加载的软服务体规模与 FIR 滤波器硬服务体相同.

观察表 5 可发现同为“软-软”方式加载服务体时,SEF-OSHRs 与 MiniCore 有类似的开销. 对于 SEF-OSHRs“软-硬”方式与“硬-软”方式有相近的延迟,但前者开销稍大,这是因为前者需要先通过 UHI 发送信号给 ISP 端,跨越了一次软硬件的边界. 通过硬服务体加载硬服务体受到 XUPV2P 平台 50M 配置速度的局限因此更慢一些. 通过软服务体加载硬服务体则受到 10Mbps 以太网速度的局限,速度最慢.

3. 2. 2 应用实例测试

支持可重构混成体系结构的操作系统目前是一个相对较新的领域,因此还没有一个被广泛认可的基准程序对 OSHRS 在应用中的表现进行测试和评估. 本节分别介绍 SEF-OSHRs 在加密/解密和数字滤波器方面的应用,实验数据表明,基于 SEF-OSHRs 的实现相对于基于 Linux 的实现获得了较高的加速比.

应用 1. 将 Anubis^① 加密算法分别在 SEF-OSHRs 和 Linux 上实现,对比其性能.

经典 Anubis 算法的密钥为 128 位,当前市售的商业化器件不能提供足够的计算资源,因此我们对经典的 Anubis 算法作了简化.简化后算法要求的密钥长度为 16 位,数据块的长度为 16 位,加密循环的执行轮数为 2 轮.

在 SEF-OSHRs 上实现简化 Anubis 算法时,我们设计了如下硬服务体:(1)加密流程硬服务体;(2)加密密钥编排硬服务体;(3)解密密钥编排硬服务体.由于 Anubis 算法具有对合结构(involutional structure),即算法使用的加密流程模块和解密流程模块构成相同,仅加密密钥和解密密钥的编排方案有差异,因此不需实现解密流程硬服务体.我们还在 ISP 端设计了描述加密算法操作流程的应用层软服务体及事务端口.当算法开始执行时,先加载 ISP 端的用户服务体,激活相应的事务端口,而后启动加/解密任务.在算法运行过程中,硬服务体(1)始终存在于 FPGA,硬服务体(2)和(3)则根据需要进行换入/换出,上述流程均由消息推动执行. Anubis 算法相关的硬服务体如图 10 所示.

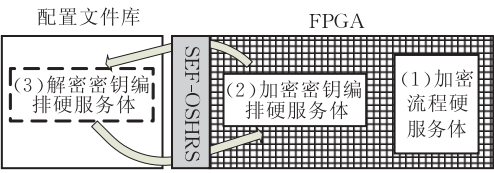


图 10 Anubis 算法相关的硬服务体示意图

衡量加密算法性能的标准是算法实现的吞吐率(*Throughput*).块加密算法的吞吐率计算公式为 $Throughput = BlockSize \times Frequency$. 其中, *Block-size* 是指加密数据块的规模, *Frequency* 是指加密执行的频率.表 6 和表 7 分别是简化的 Anubis 算法在采用不同实现方案时得到的测试结果,可见基于 SEF-OSHRs 的方案效率明显高于基于 Linux 的方案.这一方面体现了 FPGA 与 CPU 相比具有绝对的速度优势,另一方面也证明了 SEF-OSHRs 的有效性.

表 6 简化的 Anubis 算法 Linux 实现的性能数据表

	<i>Blocksize/bit</i>	<i>Frequency/MHz</i>	<i>Throughput/bps</i>
加密	16	1.131	18.10M
	16	1.138	18.22M
	16	1.154	18.46M
解密	16	1.126	18.02M
	16	1.129	18.06M
	16	1.149	18.38M

表 7 简化的 Anubis 算法 SEF-OSHRs 实现的性能数据表

	<i>Blocksize/bit</i>	<i>Frequency/MHz</i>	<i>Throughput/bps</i>
加密	16	199.417	3.191×10^3 M
解密	16	198.168	3.171×10^3 M

应用 2. 将 10 阶、20 阶 FIR 滤波器分别在 SEF-OSHRs 上以硬服务体的方式实现,并将其性能与在 Linux 上的实现相对比.数字滤波器是对离散时间信号进行滤波处理以得到期望的响应特性的离散时间系统,其中 FIR 滤波器易于实现并且非常稳定,应用最为广泛.

表 8 和表 9 是 10 阶 FIR 在两种操作系统上实现的性能测试结果.可见,10 阶 FIR 的 SEF-OSHRs 实现相对 Linux 实现的加速比为 4.5.

表 8 10 阶 FIR Linux 实现的性能数据表

<i>Blocksize/bit</i>	<i>Frequency/MHz</i>	<i>Throughput/bps</i>
16	16.532	264.5M
16	16.447	263.1M
16	16.419	262.7M

表 9 10 阶 FIR SEF-OSHRs 实现的性能数据表

<i>Blocksize/bit</i>	<i>Frequency/MHz</i>	<i>Throughput/bps</i>
16	73.216	1.171×10^3 M

表 10 为 20 阶 FIR 在 Linux 上实现的性能测试结果.20 阶 FIR 的 SEF-OSHRs 实现与 10 阶 FIR 的 SEF-OSHRs 实现具有相同的吞吐率(实现 20 阶 FIR 的硬服务体规模更大,流水级更多).可见 20 阶 FIR 的 SEF-OSHRs 实现相对 Linux 实现的加速比为 8.9.

表 10 20 阶 FIR Linux 实现的性能数据表

<i>Blocksize/bit</i>	<i>Frequency/MHz</i>	<i>Throughput/bps</i>
16	8.271	132.3M
16	8.184	130.9M
16	8.223	131.5M

分析上面的数据可以发现,采用 Linux 实现,由于 ISP 串行执行指令,所以当 FIR 的阶数升高时,吞吐率也有所下降;而采用 SEF-OSHRs 的实现则不存在此问题,只要增加可重构资源即可保证同样的吞吐率,获得更高的加速比.

4 结论与展望

本文在揭示已有 OSHRs 抽象模型本质缺陷的

① The Anubis Block Cipher. <http://paginas.terra.com.br/informatica/paulobarreto/AnubisPage.htm>

基础上,提出硬服务体的概念,设计且实现了基于SEFM 的 OSHRS. SEF-OSHRs 具有如下优点:(1)运行模型适合混成系统的运算方式,具有直接控制流转换的能力,支持高效同步消息发送,可充分发挥混成系统的性能;(2)具有统一的消息发送界面和系统对象抽象,可扩展性强;(3)如实验表明SEF-OSHRs 具有可用性和高效性.

进一步的工作包括以下 3 方面:(1)提供配置比特流的预取缓存机制和芯片上特殊资源的用户接口,进一步优化运行环境/编程模型;(2)深入研究软/硬服务体的动态迁移技术. 在混成系统运行过程中,FGPA 和 ISP 的负载随着时间和周围环境的变化而改变,平衡异构计算资源间的负载,是提高操作系统整体吞吐率的关键所在;(3)MiniCore 是 ISP 端基于 SEFM 的原型操作系统,在 SEF-OSHRs 的开发过程中,MiniCore 有新的进展,比如加入事务端口链和执行对象等新的概念. 下一步的工作还包括将 MiniCore 中新的机制引入到 SEF-OSHRs 中,这样,MiniCore 的应用程序将直接与 SEF-OSHRs 相兼容.

参 考 文 献

[1] Brebner G. A virtual hardware operating system for the Xilinx XC6200//Hartenstein R, Glesner M. Field-Programmable Logic and Applications. Berlin: Springer, 1996: 327-336

[2] Wigley G, Kearney D. The first real operating system for reconfigurable computers//Proceedings of the 6th Australasian

Computer Systems Architecture Conference, New York, 2001: 130-137

[3] Nollet V, Coene P, Verkest D et al. Designing an operating system for a heterogeneous reconfigurable SoC//Proceedings of the 17th International Parallel and Distributed Processing Symposium. New York: IEEE Press, 2003: 174

[4] Zhou Bo, Wang Shi-Ji, Qiu Wei-Dong, Peng Cheng-Lian. SHUM-UCOS: A real-time operation system for reconfigurable systems using uniform multi-task model. Chinese Journal of Computers, 2006, 29(2): 208-218(in Chinese)

(周博, 王石记, 邱卫东, 彭澄廉. SHUM-UCOS: 基于统一多任务模型可重构系统的实时操作系统. 计算机学报, 2006, 29(2): 208-218)

[5] Li Hong, Gong Yu-Chang, Zhao Zhen-Xi, Wu Ming-Qiao. Design of a servant based operating system. Journal of Computer Research and Development, 2005, 42(7): 1272-1276(in Chinese)

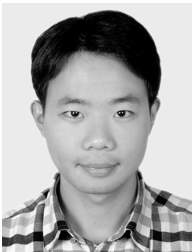
(李宏, 龚育昌, 赵振西, 吴明桥. 服务体模型与操作系统内核设计技术. 计算机研究与发展, 2005, 42(7): 1272-1276)

[6] Gong Yu-Chang, Chen Xiang-Lan, Li Xi et al. Operating system based on servant/exe-flow model. People's Republic of China Patent, 2005, 21(37): 209-215(in Chinese)

(龚育昌, 陈香兰, 李曦等. 基于服务体/执行流模型的操作系统. 中华人民共和国发明专利, 2005, 21(37): 209-215)

[7] Wu Ming-Qiao, Chen Xiang-Lan, Zhang Ye, Gong Yu-Chang. A new operating system construction model based on servant and executive flow. Journal of University of Science and Technology of China, 2006, 36(2): 230-236(in Chinese)

(吴明桥, 陈香兰, 张晔, 龚育昌. 一种基于服务体/执行流的新型操作系统构造模型. 中国科学技术大学学报, 2006, 36(2): 230-236)



QIAO Lei, born in 1982, Ph. D. candidate. His research interests include operating system, file system and reconfigurable computing.

QI Ji, born in 1978, Ph. D. candidate. His research interests include reconfigurable computing, embedded operating system.

GONG Yu-Chang, born in 1943, professor, Ph. D. supervisor. Her research interests include operating system, database, system and reconfigurable computing.

Background

This work is supported by the National Natural Science Foundation of China (No. 60273042) and the Innovation Foundation of CAS. The projects aim at constructing an Operating System for Hybrid Reconfigurable Systems (OSHRs), so as to improve the utilization of reconfigurable hardware and simplify the design flow of hybrid reconfigurable applications. The authors have focused their work on the area of RC

(Reconfigurable Computing) for more than 5 years. Their research interests also include the dynamic placement/scheduling of hardware logic blocks and the programming model for RC. This paper discusses the fundamental abstraction model, system architecture and run-time environment of the SEF-OSHRs (Servant and Execution Flow OSHRS) in detail, and proves the usability and efficiency of SEF-OSHRs by experiments.