第37卷第9期 中国科学技术大学学报 Vol. 37, No. 9
2007年9月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Sep. 2007

文章编号:0253-2778(2007)09-1066-07

支持组合设计的一种可重构计算系统任务模型

罗 赛,周学海

(中国科学技术大学计算机科学技术系,安徽合肥 230027)

摘要:针对现有可重构模型的不足,提出一种面向任务的支持组合设计的可重构计算模型系统.以生产者-消费者数据类型显式描述任务间通讯关系,并形式化描述任务的设计和运行时特征,提供组合设计规则,能够自动生成合成任务的各项属性,方便了应用系统的构建和性能评估.基于该模型可进行快速设计空间搜索,寻找最佳的系统设计方案.最后给出该模型的两种不同应用示例.

关键词:可重构计算;任务组合模型;组合生成规则;TaCoM

中图分类号:TP302.1 **文献标识码**:A

A task-composition model for reconfigurable computing systems

LUO Sai, ZHOU Xue-hai

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

Abstract: After investigating the inability of the current reconfigurable computing model, a task-oriented model featuring composition design was proposed. The model took the data type to explicitly describe the communication topology among tasks. It recorded the task's design-time and run-time characteristics formally, and automatically calculated the composite task's attributes according to the generation rule. This helped application system construction and performance evaluation. It can be used as a basis for fast design space exploration in search of an optimal design solution for a specific system. Finally, examples of two application cases of the presented model were given.

Key words: reconfigurable computing; task-composition model; generation rule; TaCoM

0 引言

可重构计算(reconfigurable computing)是一种新兴的时空域上的计算模式[1].在此之前,有两种传统的计算方法.首先是使用通用处理器软件编程,在时域上串行执行各条指令,灵活性高但速度较低;其次是使用专用ASIC芯片,直接在空域的硬件上并行执行目标算法,这虽可获得很高的处理速度,但电路不可改变,灵活性差.可重构计算模式填补了二者之间的空白.该系统包含有大量的可编程逻辑和路由资源,可在运行时自由定制硬件,使底层结构能更好地匹配高层算法的特征,具有比指令集处理器高得多的性能.可重构计算因具有高速计算能力和灵活编程的特性,得到了越来越广泛的关注.

Makimoto指出[2],半导体行业每隔10年发生一次标准化与定制化间的变革,如图1中实线所示.第一代硬连线时代是直接以硬件实现算法,算法不可改变.第二代是过程性编程时代,以冯·诺依曼计

收稿日期:2006-04-26;**修回日期**:2006-10-23
基金项目:高等学校博士学科点专项科研基金(20050358040)资助.
作者简介:罗赛,男,1979年生,博士生.研究方向:计算机体系结构,可重构计算.E-mail:keithluo@mail.ustc.edu.cn
通讯作者:周学海,教授.E-mail:xhzhou@ustc.edu.cn

算模型为基础,用软件可以实现不同的算法,但随着处理器与内存速度差距的拉大,出现了内存壁垒(memory wall)效应[3],系统的性能几乎完全取决于内存速度,即使提高处理器速度也没有效果.可重构系统在更为广阔的时空域上,以硬件分散式并行执行的方式弱化了内存瓶颈对系统性能的负面影响.硬件可编程能力使其能够充分利用未来的1 000核CPU[4]的强大资源.Hartenstein 认为可重构计算是未来体系结构的发展方向[5],如图 1 虚线所示.

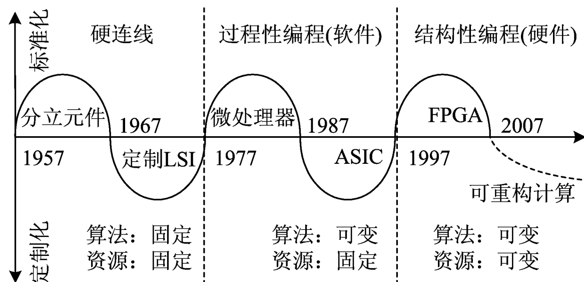

图 1　Makimoto 半导体发展曲线(经 Hartenstein 补充)

Fig. 1　Makimoto's wave (revised by Hartenstein)

可重构计算系统设计包括应用分析、软硬件划分、设计空间搜索、硬件逻辑综合、布局布线、软件编译链接、系统集成等步骤.提供系统级建模方法以支持完整的设计流程是可重构计算研究的关键问题.传统 EDA 工具,通常只涵盖独立的软硬件设计流程,不能进行系统级建模和设计空间搜索等,并且速度很慢,编译中等规模的设计就需要十几分钟甚至几个小时,不能用来进行快速的迭代式开发.

为了加快应用设计,必须有一种更高层次的系统级描述手段.基于任务、功能模块或算法对系统进行建模是可重构计算中常用的方法.它们的本质是在更高的粒度上辅助用户设计,忽略了从系统角度看来不必要的细节问题,从而减少了 EDA 工具的运算量,缩短了用户开发时间.如 El-Araby 等[6]提出的最优设计方法,就是从算法角度对应用进行分析.根据问题描述生成数据流图,获得应用的最大并行性,并映射到有物理约束的目标平台上.由系统模型和映射关系可直接估算出系统的性能.这种算法级的方法对后期工具的约束映射能力要求较高,一般不能手工参与优化.Le 等[7]的图形界面设计环境,将应用分解为不同的功能模块,由人工映射到处理单元上.Cret 等[8]的 CREC 体系结构可近似认为是一种动态结构的 VLIW,即以指令集作为软硬件之间的划分依据,根据实际使用的指令及其之间的并行性,从代码模板库中提取并生成适当的执行单元.这种基于模块或模板的方法易于进行系统整体设计,但缺乏或未显示描述执行时间、任务间互联等运行时的信息.Bondalapati 等[9]提出了任务级的混合系统架构模型 HySAM,以参数化的形式描述了特定配置的任务的各项属性,并支持组合生成新的任务,但该模型仅关注粗粒度属性,不支持细粒度电路设计.邹谊等[10]提出了一种软硬件划分算法,可用于可重构应用设计中.

本文针对上述模型的不足,提出了一种支持组合任务设计的可重构计算抽象模型 TaCoM,该模型对任务及其静态和动态属性进行形式化建模,可以直接计算出组合系统的性能参数,从而进行设计空间搜索等.它具有如下特点:(Ⅰ)采用生产者-消费者通讯机制,使任务具有更好的独立性;(Ⅱ)形式化任务建模,能够描述任务的功能、端口和属性等;(Ⅲ)层次化的设计方法,支持串行、并行和流水等多种组合方式,根据组合生成规则能够自动生成高层任务的功能和属性等;(Ⅳ)支持不同粒度的设计,包括细粒度门级电路以及粗粒度的任务等.

1　任务组合模型

任务组合模型(task-composition model,TaCoM)基于生产者-消费者通讯机制,对任务及其通讯进行建模,并可以支持不同粒度的任务.它形式化描述任务的功能、端口和各项属性,包括设计时相关的资源面积和运行时相关的时间延迟、运算带宽等;支持跨粒度层次设计,低层任务可组合生成高层任务,并通过规则自动计算出高层任务的功能和性能属性.本模型可以用于电路功能分析、性能分析和系统设计等.

1.1　类型化生产者-消费者通讯机制

任务与通讯的抽象是可重构系统模型的难点.在动态可重构系统中,为了匹配目标算法,用户可能随时改变任务模块的功能以及互连方式.由于系统拓扑结构是动态可变的,因此需要一种新型的通讯机制以支持动态可重构网络条件下系统功能和性能的建模.本文采用数据驱动的思想,以网络中的动态数据指导任务完成所需的功能,同时使用全局类型表达网络重构前后同一用途的数据,提出类型化的生产者-消费者(producer-consumer)通讯抽象机制,如图 2 所示.在该机制下,系统被划分为一个个表征功能的任务模块(图中方角矩形),它们是数据(图中

圆角矩形)的生产者或消费者.任务模块从产品池中读取("消费")需要的数据,经过自身运算处理后,将结果发布出去("生产").任务的计算和通讯是独立的,每个模块只关心自己感兴趣的某种或某些类型的数据,而不管数据从何而来,发往何处.任务的这种独立性,为任务动态重构奠定了模型基础.在通常的通讯模型中,数据发送方必须知道接收方的地址(如 IP 地址、进程编号等),这既增大了模块间的关联度,也削弱了系统的重构能力,是不必要的.同时,为了适应这种通讯机制,我们在应用设计的末期增加了系统整合阶段,在该阶段定义了系统的通讯拓扑结构,通过产品标签(代表了数据类型)将各任务关联起来.由于网络动态重构的存在,系统整合需要描述可能存在的网络连接方式的全集[11].

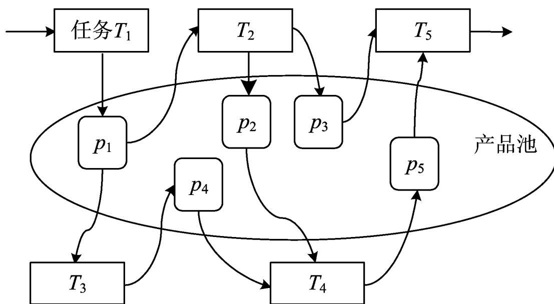

图 2　生产者-消费者通讯机制

Fig. 2　Producer-consumer communication paradigm

本通讯机制可被定义为带类型的集合系统.每个数据产品 p 具有类型 T_p,向量数据$\langle p, t\rangle$具有类型 $T_p\times T_t$.系统通常包含多种产品,所有的产品构成集合 P,称为数据产品集.产品集 P 的类型记为 $T_P=\{T_p|p\in P\}$.

任务通过端口访问数据产品池,端口也具有类型,任务执行由数据驱动.当产品池出现与任务消费端口类型匹配的数据时,触发任务执行,消费掉该数据产品,并生产出与其生产端口类型相符的数据,因此,数据类型是联系各任务的桥梁.图 2 中,任务 T_2 消费 T_{p_1}类型的数据,生产出 T_{p_2} 和 T_{p_3} 类型的数据.T_5消费掉 p_3.类型 T_{p_3} 联系了 T_2和 T_5.同时,这种数据驱动机制也易于用硬件实现,由与门检查数据类型,由握手信号指示数据的生产与消费.

1.2　任务建模

任务模块由五元组 task$=\langle f,A,\boldsymbol{B},\boldsymbol{C},\boldsymbol{D}\rangle$表示.其中,$f$ 是任务的功能函数(function),A 是任务的属性集(attributes set),$\boldsymbol{B}$ 是任务的配置数据流向量(bit-stream vector),$\boldsymbol{C}$ 是消费端口向量(consuming ports vector),$\boldsymbol{D}$ 是生产端口向量(producing ports vector).如图 3 所示.

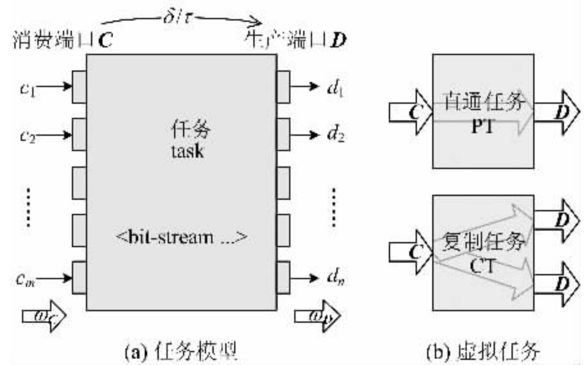

图 3　任务模型及虚拟任务

Fig. 3　Task model and virtual task

硬件配置数据流存储在向量 $\boldsymbol{B}$ 中.由配置数据流大小可以计算出配置时间 $t_R=|\boldsymbol{B}|/R$,$|\boldsymbol{B}|$表示向量长度,R 为配置速度.$\boldsymbol{B}$ 与具体实现方式有关.为完成相同功能的 f,允许有不同速度和面积的实现版本,用户在设计空间搜索阶段可以选择适用的版本.

任务具有一个或多个消费端口 c_i,该端口仅接收 T_{c_i}类型的数据.所有端口总记为向量$\boldsymbol{C}=\langle c_1,c_2,\cdots,c_m\rangle$,其类型为 $T_{\boldsymbol{C}}=T_{c_1}\times T_{c_2}\times\cdots\times T_{c_m}$,$m=|\boldsymbol{C}|$表示消费端口的数量.

同样,生产端口向量记为 $\boldsymbol{D}=\langle d_1,d_2,\cdots,d_n\rangle$,其类型为 $T_{\boldsymbol{D}}=T_{d_1}\times T_{d_2}\times\cdots\times T_{d_n}$,端口数量 $n=|\boldsymbol{D}|$.

f 描述该任务的功能,f:$T_{\boldsymbol{C}}\rightarrow T_{\boldsymbol{D}}$.严格来说,生产出的数据可能与消费数据的整个历史有关,即 f:$T_{\boldsymbol{C}}^{\infty}\rightarrow T_{\boldsymbol{D}}$.如 N 阶数字滤波器的输出与最近 N 次输入信号有关.对于这类复杂的应用,我们使用项重写(term rewriting)理论[12]描述其功能,具体方法参见文献[13].值得注意的是,项重写的逐步精化的步骤很好地匹配了本模型的任务组合的特征,本模型可作为功能组合语义的描述手段嵌入.对大多数应用而言,f 可简单地表示为针对当前数据的数学或逻辑表达式或者伪函数符号的形式.本文假定不考虑历史输入数据对结果的影响.

A 是任务的属性集,描述任务的各项参数特性,如性能、面积等.属性是判断设计方案优劣的依据,用于优化设计.设计组合任务时,根据不同组合方式,可由原任务属性计算出组合任务的属性.用户可根据需要扩展属性集,如加入功耗属性.下面列举了

本文使用到的属性：

端口延迟矩阵 $\boldsymbol{\delta}$ 用以描述组合逻辑电路中端口到端口的延迟. $\boldsymbol{\delta}\in \mathrm{R}^{m\times n}$，$\delta_{ij}$ 表示端口 c_i 到端口 d_j 的延迟. 该属性主要用于精确分析细粒度门级电路的延迟，为任务组合提供基础数据. 如果 d_j 和 c_i 间无依赖关系，则 δ_{ij} 取 $-\infty$.

τ 是描述该任务从消费掉旧数据到生产出相关新数据之间的最大延迟. 对于组合逻辑电路，$\tau=\max\limits_{i,j}\delta_{ij}$；对于时序逻辑，$\tau$ 由用户确定.

最大吞吐率 β 用以描述该任务的最大数据处理速率. 对于组合逻辑电路，$\beta\propto 1/\tau$；对于时序逻辑，如果采用流水线式设计，由于在 τ 时间段内处理了更多的数据，β 将更大.

a 为占用资源面积. 任务组合时，资源面积累加.

$\omega_{\boldsymbol{C}}$，$\omega_{\boldsymbol{D}}$ 分别为应用运行所需的消费、生产带宽. 当 $\omega_{\boldsymbol{C}}\leqslant\beta$ 时，任务才能够实时处理数据.

1.3　虚拟任务

为了能够支持复杂的任务组合方式，我们设计了两类虚拟任务，如图 3(b)所示. 直通任务(pass-through task，PT)将输入数据直接输出，复制任务(copy task，CT)将数据复制一份并同时发往两个输出端口. 根据端口数目和宽度的不同，这两类任务具有多种不同的实现版本. 它们统一描述为

$\mathrm{PT}.f(\boldsymbol{C})=\boldsymbol{C}$，$\mathrm{PT}.\boldsymbol{\delta}=\boldsymbol{0}$，$\mathrm{PT}.\tau=0$，$\mathrm{PT}.\beta=\infty$，$\mathrm{PT}.a=0$，$\mathrm{PT}.\omega_{\boldsymbol{C}}=\mathrm{PT}.\omega_{\boldsymbol{D}}=\infty$，

$\mathrm{CT}.f(\boldsymbol{C})=\langle\boldsymbol{C},\boldsymbol{C}\rangle$，$\mathrm{CT}.\boldsymbol{\delta}=\boldsymbol{0}$，$\mathrm{CT}.\tau=0$，$\mathrm{CT}.\beta=\infty$，$\mathrm{CT}.a=0$，$\mathrm{CT}.\omega_{\boldsymbol{C}}=\mathrm{CT}.\omega_{\boldsymbol{D}}=\infty$.

2　任务组合生成法则

可重构计算系统需要有高度自动化的方法辅助应用设计. 系统设计空间的大小随着可重构资源的多少呈几何级数增长，快速性能评估日益成为设计空间搜索中重要的一环. 本文通过分析各种任务组合方式对系统功能和性能的影响，提出了能够自动生成高层任务属性的组合生成法则. 该法则可以定义为 $\mathcal{G}$：task×task→task，它抽象地将两个任务组合在一起生成新任务的过程. 例如，对于支持部分可重构的器件如 Xilinx Virtex-Ⅱ系列 FPGA，配置新任务时不会影响其他区域旧任务的执行. 因此，配置数据流可以累加，同时配置时间和占用面积也随之累加，即

$$\mathcal{G}(T_1,T_2).\boldsymbol{B}=T_1.\boldsymbol{B}\circ T_2.\boldsymbol{B},$$

$$\mathcal{G}(T_1,T_2).a=T_1.a+T_2.a,$$

$$\mathcal{G}(T_1,T_2).t_R=T_1.t_R+T_2.t_R.$$

式中，$\circ$ 是向量串接运算符.

我们提出三种基本的组合方式，如图 4 所示. 根据这三种基本方式，配合虚拟任务可以生成更复杂的组合方式.

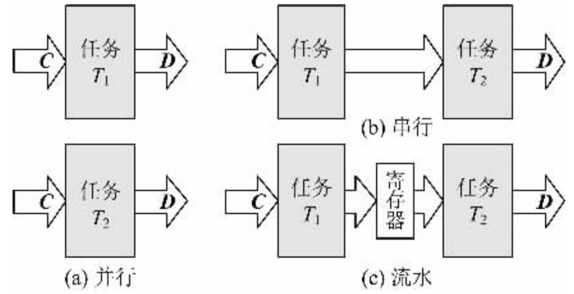

图 4　几种任务组合模式

Fig. 4　Task composition mode

(Ⅰ) 并行 $\mathcal{G}_{\mathrm{par}}$

$\mathcal{G}_{\mathrm{par}}$ 代表了并行组合方式下的任务生成法则，如图 4(a)所示. 这是最简单的任务组合方式，两个任务并行执行，互不影响. 该方式的通常是将相同的任务重复设置，以获得更高的数据吞吐率. 对于新生成的任务，它的端口是原端口的并(严格来说是矢量串接. 为了行文与标记方便，下文在不混淆的情况下，借用了集合的术语和符号)，延迟时间是原任务的最大值，吞吐量是原任务的和.

其计算法则为

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).f=\langle T_1.f,T_2.f\rangle,$$

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).\boldsymbol{C}=T_1.\boldsymbol{C}+T_2.\boldsymbol{C},$$

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).\boldsymbol{D}=T_1.\boldsymbol{D}+T_2.\boldsymbol{D},$$

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).\boldsymbol{\delta}=\begin{bmatrix}T_1.\boldsymbol{\delta} & -\infty\\ -\infty & T_2.\boldsymbol{\delta}\end{bmatrix},$$

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).\tau=\max(T_1.\tau,T_2.\tau),$$

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).\beta=T_1.\beta+T_2.\beta,$$

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).\omega_{\boldsymbol{C}}=T_1.\omega_{\boldsymbol{C}}+T_2.\omega_{\boldsymbol{C}},$$

$$\mathcal{G}_{\mathrm{par}}(T_1,T_2).\omega_{\boldsymbol{D}}=T_1.\omega_{\boldsymbol{D}}+T_2.\omega_{\boldsymbol{D}}.$$

其中，端口延迟 δ_{ij} 在跨任务的情况下为 $-\infty$，因为两端口之间无依赖关系.

(Ⅱ) 串行 $\mathcal{G}_{\mathrm{ser}}$

串行组合方式如图 4(b)所示，任务 T_1 生产的数据直接由 T_2 消费. 对于组合逻辑，输入信号必须经过中间端口才能到达输出端口，因此端口延迟等于所有路径下的最大值. 并且 τ 可根据 $\boldsymbol{\delta}$ 矩阵计算出来. 另外 τ 也可粗略估计为原任务的延迟之和，因

为新任务的关键路径在原任务中可能并不是关键路径,所以此数值偏大. 对于设计良好的模块,端口速率基本趋于平衡,两种计算方式的差别不大.

本文定义矩阵的圈乘运算 $\boldsymbol{X}=\boldsymbol{Y}\otimes\boldsymbol{Z}$,其中,$\boldsymbol{X}_{ij}=\max\limits_{k}(\boldsymbol{Y}_{ik}+\boldsymbol{Z}_{kj})$. 在 T_1 的生产速度与 T_2 的消费速度匹配的境况下,计算法则为

$$\mathcal{G}_{\text{ser}}(T_1,T_2).f=T_1.f\circ T_2.f,$$
$$\mathcal{G}_{\text{ser}}(T_1,T_2).\boldsymbol{C}=T_1.\boldsymbol{C},$$
$$\mathcal{G}_{\text{ser}}(T_1,T_2).\boldsymbol{D}=T_2.\boldsymbol{D},$$
$$\mathcal{G}_{\text{ser}}(T_1,T_2).\boldsymbol{\delta}=T_1.\boldsymbol{\delta}\otimes T_2.\boldsymbol{\delta},$$
$$\mathcal{G}_{\text{ser}}(T_1,T_2).\tau=T_1.\tau+T_2.\tau,$$
$$\mathcal{G}_{\text{ser}}(T_1,T_2).\beta=1\Big/\left(\frac{1}{T_1.\beta}+\frac{1}{T_2.\beta}\right),$$
$$\mathcal{G}_{\text{ser}}(T_1,T_2).\omega_{\boldsymbol{C}}=T_1.\omega_{\boldsymbol{C}},$$
$$\mathcal{G}_{\text{ser}}(T_1,T_2).\omega_{\boldsymbol{D}}=T_2.\omega_{\boldsymbol{D}}.$$

式中,∘表示函数复合运算.

(Ⅲ) 流水 $\mathcal{G}_{\text{pipe}}$

如图 4(c)所示,两个任务串行组合一起,为保证性能,在流水级中间添加缓冲数据的寄存器(或FIFO). 如果 T_1 的输出级或 T_2 的输入级含有类似功能的寄存器,则可直接连接而不需额外缓冲. 在该互连方式中,属性 $\boldsymbol{\delta}$ 已无意义,因为不再关心具体的延迟. 更实用的是总延迟时间 τ,即原延迟时间之和. 数据吞吐率 $\beta\propto 1/\tau$,具体倍数由流水线结构及性质而定. 该方式下有如下运算法则:

$$\mathcal{G}_{\text{pipe}}(T_1,T_2).f=T_1.f\circ T_2.f,$$
$$\mathcal{G}_{\text{pipe}}(T_1,T_2).\boldsymbol{C}=T_1.\boldsymbol{C},$$
$$\mathcal{G}_{\text{pipe}}(T_1,T_2).\boldsymbol{D}=T_2.\boldsymbol{D},$$
$$\mathcal{G}_{\text{pipe}}(T_1,T_2).\tau=T_1.\tau+T_2.\tau,$$
$$\mathcal{G}_{\text{pipe}}(T_1,T_2).\beta=\min(T_1.\beta,T_2.\beta)$$
$$\mathcal{G}_{\text{pipe}}(T_1,T_2).\omega_{\boldsymbol{C}}=T_1.\omega_{\boldsymbol{C}},$$
$$\mathcal{G}_{\text{pipe}}(T_1,T_2).\omega_{\boldsymbol{D}}=T_2.\omega_{\boldsymbol{D}}.$$

3 案例分析

本文分别以细粒度的组合电路延迟分析和粗粒度的设计空间搜索来演示 TaCoM 模型的描述能力,并验证组合计算规则的正确性.

3.1 组合电路延迟分析

本例分析细粒度门级组合电路的端口延迟,如图 5 所示,根据半加器的性能等参数计算全加器的相应参数. 其中,图 5(a)是半加器 HA,具有两个消费端口和两个生产端口. 定义 INT1={0,1}是一位数值集合,因此有端口类型 $T_{c_1}=T_{c_2}=T_{d_1}=T_{d_2}=$ INT1,端口向量 $\boldsymbol{C}=\langle c_1,c_2\rangle$,$\boldsymbol{D}=\langle d_1,d_2\rangle$,端口向量类型 $T_{\boldsymbol{C}}=T_{\boldsymbol{D}}=\text{INT1}^2$. 根据定义,半加器可表示为 $\text{HA}=\langle\langle c_1,c_2\rangle\mapsto\langle c_1\wedge c_2,c_1\oplus c_2\rangle,A,B,\langle c_1,c_2\rangle,\langle d_1,d_2\rangle\rangle$.

首先假设与门、或门的延迟为 1.0 ns,异或门为 1.1 ns,那么延迟矩阵 $\text{HA}.\boldsymbol{\delta}=\begin{pmatrix}1.0 & 1.1\\1.0 & 1.1\end{pmatrix}\in\mathbb{R}^{2\times2}$.

下面计算全加器的端口延迟. 如图 5(b),全加器 FA 的逻辑结构由两个半加器和一个或门组成,消费端口向量 $\text{HA}.\boldsymbol{C}=\langle c_1,c_2,c_3\rangle\in\text{INT1}^3$,生产端口向量 $\text{HA}.\boldsymbol{D}=\langle d_1,d_2\rangle\in\text{INT1}^2$. 三模块的组合形式较为复杂,不符合预定义的三种基本方式,故我们在电路中添加虚拟任务,见图 5(c)的 PT 任务. 设 $T_1=\mathcal{G}_{\text{par}}(\text{HA},\text{PT})$,$T_2=\mathcal{G}_{\text{par}}(\text{PT},\text{HA})$,$T_3=\mathcal{G}_{\text{par}}(\text{OR},\text{PT})$,$T_4=\mathcal{G}_{\text{ser}}(T_1,T_2)$,则 $\text{FA}=\mathcal{G}_{\text{ser}}(T_4,T_3)$. 根据 $\mathcal{G}_{\text{par}}$ 计算法则有

$$T_1.\boldsymbol{\delta}=\begin{pmatrix}1.0 & 1.1 & -\infty\\1.0 & 1.1 & -\infty\\-\infty & -\infty & 0\end{pmatrix},$$
$$T_2.\boldsymbol{\delta}=\begin{pmatrix}0 & -\infty & -\infty\\-\infty & 1.0 & 1.1\\-\infty & 1.0 & 1.1\end{pmatrix},$$
$$T_3.\boldsymbol{\delta}=\begin{pmatrix}1.0 & -\infty\\1.0 & -\infty\\-\infty & 0\end{pmatrix}.$$

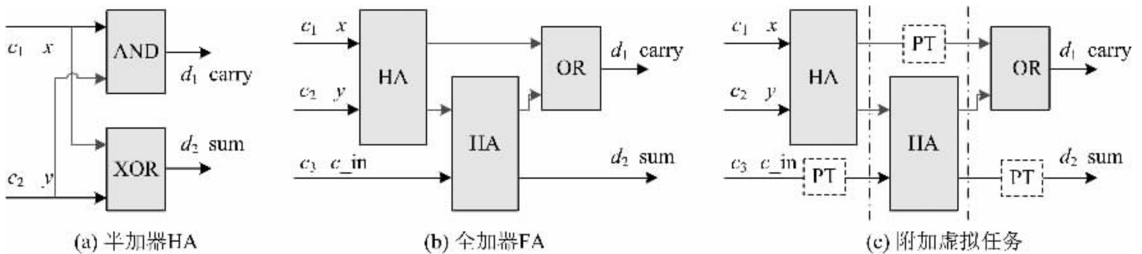

图 5 加法器(组合电路)设计

Fig. 5 Adder (combinatorial logic circuit) design

根据 $\mathcal{G}_{\rm ser}$ 计算法则有 $T_4.\boldsymbol{\delta}=T_1.\boldsymbol{\delta}\otimes T_2.\boldsymbol{\delta}=\begin{pmatrix}1.0&2.1&2.2\\1.0&2.1&2.2\\-\infty&1.0&1.1\end{pmatrix}$，全加器延迟矩阵 $FA.\boldsymbol{\delta}=T_4.\boldsymbol{\delta}\otimes T_3.\boldsymbol{\delta}=\begin{pmatrix}3.1&2.2\\3.1&2.2\\2.0&1.1\end{pmatrix}$. 由矩阵数据可知，$c_1$ 和 c_2 到 d_1 的延迟最长，为 3.1 ns；c_3 到 d_2 的延迟最短，为 1.1 ns. 由圈乘运算的计算过程可知，存在两条关键路径 $c_1\to$XOR$\to$AND$\to$OR$\to d_1$ 和 $c_2\to$XOR$\to$AND$\to$OR$\to d_1$. 延迟 τ 等于矩阵中最大的数，$FA.\tau=3.1$. 同时，我们通过手工分析得出了同样的结果，这在一定程度上反映了本模型计算规则的正确性.

另外，根据组合运算规则，全加器函数可描述为

$$FA.f(c_1,c_2,c_3)=\langle c_1\wedge c_2\vee(c_1\otimes c_2)\wedge c_3, c_1\oplus c_2\oplus c_3\rangle.$$

3.2　应用系统设计空间搜索

TaCoM 组合生成规则能够快速计算出应用系统的各项参数，下文通过粗粒度模块组合展示 TaCoM 在设计空间搜索中的应用. 以一个简单的信号采集处理系统为例，我们在 REARM-1 硬件平台上获取实验参数，并与通过规则计算出的结果作比较. REARM-1 是我们自主研制的可重构计算平台，由 ARM7 主控 CPU 和可重构硬件 Virtex-Ⅱ FPGA 组成，具有高速硬件处理能力和大容量的可重构资源[11].

本应用系统整体上由三级流水线组成，每级流水由独立的任务模块实现. 我们为各个模块书写了三种不同的实现版本，在此基础上展开设计空间搜索，如表 1 所示. 第一级流水 T_1 是数据采集模块，逻辑简单，占用面积很少. 其中，T_{1a} 以 5 MB/s 的速度采样单个信号. 由延迟 τ 可知，硬件可提供远高于 5 MB/s 的处理速度，但实际应用并不需要这么多. T_{1b} 和 T_{1c} 以同样的速度分别进行 4 和 16 通道采样，具有更高的数据吞吐率. 任务 T_2 是低通滤波模块，实现了 20 阶 FIR 算法. 其中，T_{2a} 采用 5 级流水的方式设计，运算带宽最高，但因流水级数较多，延迟也最长. T_{2b} 采用非流水的串行加法器设计，速度最慢. T_{2c} 采用非流水树型加法器设计，减少了加法器数目，面积最少，速度介于前两者之间. 任务 T_3 是频谱分析模块，进行 16 位 4 096 点 FFT 运算. 其中，T_{3a} 采用流水式结构，速度最快，但占用资源也最多. T_{3b} 采用基数为 4 的 Cooley-Tukey FFT 算法[14]. T_{3c} 是基数为 2 的同种算法，并行度较小，速度最慢. 所有任务均未采用部分重构的方法设计，受配置文件格式的限制，配置流长度(经压缩)与占用资源面积并不成比例.

表 1　功能块的各实现版本

Tab. 1　Implementations of task module

任务	$\lvert B\rvert$/字节	t_R/ms	a/%	τ/μs	β/(MB·s^{-1})
T_{1a}	137 019	2.7	0.0	0.00	5
T_{1b}	142 576	2.9	0.0	0.00	20
T_{1c}	158 235	3.1	0.1	0.00	80
T_{2a}	300 581	6.0	8.2	0.07	140
T_{2b}	371 197	7.4	7.6	0.05	40
T_{2c}	266 141	5.3	7.5	0.02	90
T_{3a}	464 944	9.3	72.8	37.24	220
T_{3b}	448 408	9.0	42.6	96.81	85
T_{3c}	394 568	7.9	18.7	320.89	25

本例设计空间大小为 3×3×3＝27，命名顶层应用设计 $S_{xyz}=\mathcal{G}_{\rm pipe}(\mathcal{G}_{\rm pipe}(T_{1x},T_{2y}),T_{3z})$. 根据组合生成规则，可以快速计算出各种设计方案的性能参数，部分结果列入表 2. 由表 2 可知，在追求最高数据处理速度的同时，要求面积最小，可选择设计方案 *Sccb*；如果对面积没有限制，只要求响应时间最快，可选择 *Scca*. 如果只需处理 4 个通道的信号，可取面积最小的设计 *Sbbc*，或延迟最短的设计 *Sbca*. 因前者响应太慢，后者面积太大，我们也可以取折衷设计 *Sbcb*. 单通道信号处理时有同样的分析. 设计空间大小通常呈几何级数增长，需要一定的空间压缩技术. 本文使用多维度剪枝方法，以不同侧重点(如面积、延迟和速度)最终选择了 7 种方案. 基于二分决策图的约束剪枝是另一种常用的方法[15].

表 2　部分组合设计结果

Tab. 2　Several design result

组合	a/%	τ/μs	β/(MB·s^{-1})	Ea/%	$E\tau$/%
Sacc	26.2	320.91	5	0.8	0.0
Sbbc	26.3	320.94	20	0.5	0.0
Sbca	80.3	37.26	20	2.7	0.1
Sbcb	50.1	96.83	20	1.8	0.0
Scaa	81.1	37.31	80	3.1	0.2
Scca	80.4	37.26	80	2.6	0.2
Sccb	50.2	96.83	80	1.0	0.0

同时，本文使用 Xilinx EDA 工具编译综合各组合设计方案，得出面积和延迟值，并以此为基准计算组合生成规则的误差，参见表 2 的 Ea 和 $E\tau$. 需要注意的是，因各任务模块运行频率不同，所以使用不同的时钟源驱动不同的模块. 吞吐率 β 与时钟相关，不

在本讨论之列.从结果可知,延迟误差基本上可以控制在 0.2%以内,这是因为我们针对表 1 和表 2 在综合时使用了同样的优化速度的选项,EDA 工具生成了同样的关键路径.经分析,误差主要源自流水级间插入的寄存器和不同的路由路线.相对而言,因为 EDA 在速度优化时可能会改变某些布局布线方式和逻辑单元的选用,尤其当面积占用率较高时(如 *Scaa*),布局布线的随机性更大,引起误差升高,因此面积的误差较大.总而言之,4%的实验最高误差说明了本模型具有相当高的准确度.如果器件支持动态部分可重构,那么误差将更小,因为部分重构直接使用了子任务的配置数据流文件,而不需重新综合编译.

4 结论

考虑到可重构计算应用设计的复杂性,本文提出了一种系统级任务抽象模型 TaCoM.在本模型中,系统由运算任务组成,任务之间通过数据端口通讯.本模型抽象出任务的计算和通讯两个基本要素,同时以数值参数的形式描述任务的性能和面积属性等.本文提出组合生成规则,以解决从门级到算法级等不同粒度下的任务组合设计问题.根据该规则快速计算高层任务的性能参数,从而能够对应用系统进行设计空间搜索,以寻找最佳的实现方案.

本模型主要在空域上进行应用设计,仍未考虑时域划分等问题.我们将在后续工作中拓展该模型.同时我们将设计一种体系结构描述语言,以计算机可读的形式实现该模型;并编写 EDA 工具和任务库,自动完成组合设计任务.

参考文献(References)

[1] Dehon A, Wawrzynek J. Reconfigurable computing: what, why, and implications for design automation [C] // Proc. 36th ACM/IEEE Conf. on Design Automation. NY: ACM Press, 1999: 610-615.

[2] Makimoto T. The rising wave of field-programmability [C] // Proc. FPL2000, LNCS 1896. London: Springer-Verlag, 2000:1-6.

[3] Wulf W, McKee S. Hitting the memory wall: implications of the obvious [J]. Computer Architecture News, 1995, 23(1):20-24.

[4] Patterson D A. The future of computer architecture [EB/OL]. Berkeley EECS Annual Research Symposium 2006, http://www.eecs.berkeley.edu/BEARS/.

[5] Diessel O, Milne G. Field- programmable logic and applications-the roadmap to reconfigurable computing [C]//10th International Conference, FPL 2000. Heidelberg: Springer-Verlag, 2000, 1896:707-717.

[6] El-Araby E, El-Ghazawi T, Gaj K. A system-level design methodology for reconfigurable computing applications [C] // IEEE Int'l Conf. on Field-Programmable Technology. Washington: IEEE Press, 2005:311-312.

[7] Le T, Donohoe G, Buehler D, et al. Graphical design environment for a reconfigurable processor [C] // 7th annual MAPLD International Conference. Washington: NASA, 2004.

[8] Cret O, Pusztai K, Vancea C, et al. CREC: a novel reconfigurable computing design methodology [C] // Proceedings of the International Parallel and Distributed Processing Symposium. Washington: IEEE Press, 2003: 22-26.

[9] Bondalapati K, Prasanna V. Reconfigurable Computing Systems [J], Proc. IEEE, 2002, 90(7):1 201-1 217.

[10] ZOU Yi, ZHUANG Zhen-quan, YANG Jun-an. HW-SW partitioning based on genetic algorithms [J]. Journal of University of Science and Technology of China, 2004,34(6): 724-731.
邹谊,庄镇泉,杨俊安. 基于遗传算法的嵌入式系统软硬件划分算法[J],中国科学技术大学学报,2004,34(6): 724-731.

[11] LUO Sai. Architectural research and implementation on reconfigurable computing system[D]. PhD Thesis, University of Science and Technology of China,2006.

[12] Bezem M, Klop J W, de Vrijer R, et al. Term Rewriting Systems [M]. Cambridge: Cambridge University Press, 2003.

[13] LUO Sai,ZHOU Xue-hai. A modeling and simulation framework for dynamic reconfigurable computing system using term-rewriting [J]. Mini-Micro Systems, 2007,28(9):1 652-1 659.

[14] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series [J]. Mathematics of Computation, 1965, 19(90) :297-301.

[15] Mohanty S, Prasanna V K, Neema S, et al. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multigranular simulation[C]// Language Compilers and Tools for Embedded Systems. Berlin: ACM Press, 2002:18-27.