第33卷 第12期 计 算 机 工 程 2007年6月
Vol.33 No.12 Computer Engineering June 2007
·开发研究与设计技术· 文章编号：1000—3428(2007)12—0259—03 文献标识码：A 中图分类号：TP391

# 面向TTA结构的可重定向周期精确模拟器的设计与实现

岳 虹，王志英，戴 葵，赵学秘
（国防科学技术大学计算机学院，长沙 410073）

**摘 要：**给出了一种面向传输触发体系结构的可重定向周期精确模拟器的设计与实现。该模拟器能够在不修改的情况下，对不同的 TTA 硬件体系结构设计进行高效的模拟。同时提供了方便的用户自定义扩展指令的添加接口。为了加快模拟速度，提出了一种预解释模拟机制。
**关键词：**周期精确模拟器；可重定向；传输触发体系结构；嵌入式片上系统

# Design and Implementation of Retargetable Cycle-accurate Simulator for TTA

YUE Hong, WANG Zhiying, DAI Kui, ZHAO Xuemi
(School of Computer, National University of Defense Technology, Changsha 410073)

**【Abstract】**This paper presents the design and implementation of a retargetable cycle-accurate simulator for transport triggered architecture (TTA). Different TTA hardware architecture can be simulated without modification of the simulator itself. Flexible interface to add user-custom instructions is also provided. To increase the simulation speed, a pre-interpretation mechanism is presented.
**【Key words】**Cycle-accurate simulator; Retargetable; Transport triggered architecture(TTA); Embedded SoC

在嵌入式处理器设计过程中，为了减少开发时间和设计开销，软件模拟器被广泛采用。作为一种验证和模拟体系结构功能和性能的方式，它减小了物理设计和实现处理器的代价和风险。

对于当前的嵌入式SoC设计领域中的处理器体系结构模拟，对每一种体系结构都手工开发不同的体系结构模拟器的方法不再有效。随着应用需求的增强和细化，处理器的指令集结构(ISA)和微体系结构必须面向特定的应用进行设计。为了寻求最好解决方案，必须对所有可行的体系结构方案进行筛选和优化，相应的软件开发工具，包括模拟器必须能面向所有这些体系结构。这样的需求必然导致可重定向模拟器的产生，它必须根据不同的体系结构来自动生成与之对应的模拟器。

本文给出的面向传输触发体系结构(transport triggered architecture, TTA)的可重定向周期精确模拟器，是基于TTA进行嵌入式ASIP(application specific instruction-set processor)设计过程中所需的一个非常重要的工具[1]。通过读取体系结构描述文件并进行解析，在已有的硬件功能单元描述库中选取相应的功能单元描述，搭建成所对应的体系结构，并在此体系结构上对不同的应用程序进行周期精确的模拟。对于用户自定义的指令及功能单元，也可以通过简单的方式加入到模拟器中。

## 1 TTA及其设计框架

由Henk Corporaal等人提出的TTA[5]，可以看成传统VILW体系结构的一个超集。其结构如图1所示，数据通路段由功能单元(function unit, FU)和寄存器文件通过Socket互联在多条总线上组成。每个功能单元包含一个或者多个操作数寄存器(O)、触发寄存器(T)和结果寄存器(R)。与传统的操作触发方式不同的是，TTA中的运算是由数据传输来触发的。当数据传输到触发寄存器时，就会触发该功能单元将操作数寄存器和触发寄存器中的值作为源操作数，来完成相应的操作，并将结果放入结果寄存器。同一个功能单元通常可以执行多种操作(比如加法和减法)，通过不同的控制信号来区分。

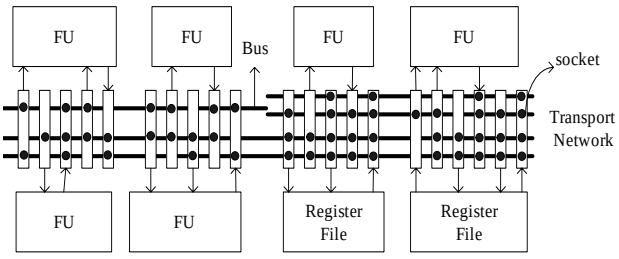

**图1 TTA典型结构**

因为是数据传输触发，所以所有的操作，包括load/store、分支、跳转等，都通过 move 指令来完成。例如，将一个加法操作变换成3个move操作：

ADD r3, r2, r1=>r1 -> $O_{ADD}$;
r2 -> $T_{ADD}$;
$R_{ADD}$ -> r3;

基于TTA的ASIP设计流程在文献0中有详细的说明。从文献[1]中可以看出，软件模拟器是整个ASIP处理器设计过程中非常重要的一环。体系结构优化器必须通过模拟器运行得到的统计信息对体系结构进行筛选以获取最优解。模拟器必须对所有可行的体系结构进行模拟，必须做到可重定向。

**基金项目：**国家自然科学基金资助项目(90407022)
**作者简介：**岳 虹(1980－)，女，博士生，主研方向：计算机体系结构，高性能微处理器设计；王志英，博导、教授；戴 葵，博士、副教授；赵学秘，博士生
**收稿日期：**2006-07-30 **E-mail：**yuehong@chiplight.com.cn

## 2 设计方案

### 2.1 总体结构

如图 2 所示，模拟器的软件结构主要分为 3 个部分：用户界面，模拟器引擎和基本部件库。

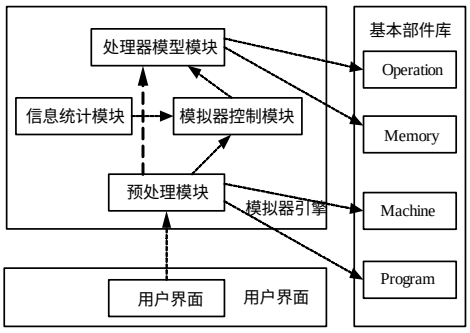


**图 2 模拟器总体结构**

基于文本的用户界面用于接收用户输入，包括体系结构描述文件及应用程序代码文件。用户界面也接收脚本输入。同时，用户界面也负责输出模拟结果和统计信息。

模拟器引擎分为 4 个部分：预处理模块，模拟器控制模块，信息统计模块和处理器模型模块。预处理模块负责对体系结构描述文件和程序代码文件进行解析和预解释。对体系结构描述文件的解析将建立起对应的处理器模型；程序代码文件进行预解释之后，将以已译码的方式存放在指令队列里。模拟器控制模块负责在已建立起的处理器模型上运行指令队列里的指令，从而完成整个模拟过程。信息统计模块收集模拟过程及模拟结果中的信息，将用户所需输出传递到用户界面。处理器模型模块是整个模拟器运行的基础。

基本部件库是对 TTA 硬件结构和 TTA 程序进行表示的模块的集合。其中，Operation 表示 TTA 处理器中包含的所有已定义的操作，即能够被执行的指令。Program 是对应用程序所建立的一个对象模型，用来描述将被模拟的程序。Machine 中则包含 TTA 硬件结构中的一些模块，诸如总线、寄存器、功能单元，它们均用机器模型库中的模块来表示。Memory 则是对存储器建立的模型。模拟器引擎在其工作的各个阶段，均需使用基本部件库中的不同模型来完成整个模拟。不同的体系结构描述文件，将对应于基本部件库中的不同部件，构建不同的处理器模型。

### 2.2 处理器模型的建立

从被模拟程序的角度来看，处理器模型就是能够被读取和写入的处理器部件的集合。为了做到可重定向，模拟器引擎中的预处理模块对体系结构描述文件进行解析时，将处理器的模型的建立分成 2 个步骤：

(1)从基本部件库 Machine 中选取部件模型，搭建与具体体系结构无关的处理器模型框架，即 TTA 结构中一定具有的硬件架构，包括基本的控制通路和数据通路，固定的流水线结构，基本的部件接口等。具体信息，对处理器模型框架进行完善。此时需要从基本部件库 Operation 中选取体系结构描述文件中给出的所有操作，即该体系结构所支持的指令集，加入到处理器模型中。同时，对于其它参数，比如：总线数目，寄存器文件数目，存储器接口类型等，都需要根据体系结构描述文件给出的数目来确定。

(2)从 Memory 中选取相应的存储器部件，确定完整的处理器模型。在生成处理器模型的同时，还会生成一个译码映射信息文件，用于指导应用程序的预解释。

### 2.3 应用程序的预解释

为了提高周期精确模拟器的模拟速度，在本模拟器的设计中采用了应用程序的预解释机制。如图 3 所示，模拟器引擎的预处理模块读取应用程序代码文件，然后逐条对代码文件中的指令进行译码，译码后的指令被放入应用程序模型的指令队列里。模拟器控制模块启动之后，将从指令队列里读取相应的已译码指令，进行模拟。

预解释机制的采用，使得在模拟程序运行期间译码段的操作，提前到程序运行之前完成，减少了实际程序运行的时间，提高了模拟器的性能。

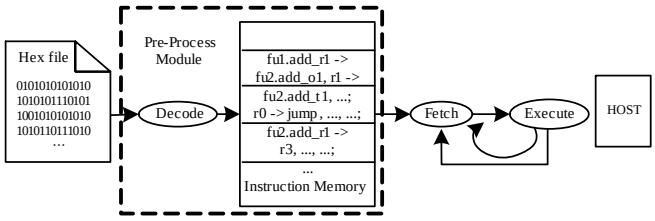


**图 3 应用程序的预解释**

### 2.4 并发操作的模拟

在TTA结构的处理器中，每一个周期运行着多个move操作，即在不同的总线上，都有数据从这个功能单元或者寄存器传输到另一个功能单元或者寄存器中[1,5]。同时，用户需要在某个周期知道当前的寄存器内容或者处理器状态等信息，需要在指定周期向用户输入处理器的相关信息。另外，除了模拟处理器的数据通路和控制通路以外，处理器的其它部件也在同步工作，需要对这些并行操作进行正确的模拟。在本模拟器中，模拟器控制模块将按照下面的步骤进行一个周期以内的并发操作的模拟。

(1)源操作数的动作。依次将同一条指令中多个 move 操作的源操作数通过相应的 socket 放置到对应的总线上，根据源操作数的类型(短立即数，长立即数，功能单元寄存器或者通用寄存器)将进行不同的处理。

(2)数据的传输。依次检查同一条指令中多个 move 操作的传输类型。如果是传输到操作数寄存器或者通用寄存器，那么将数据放入相应的寄存器即可。如果是传输到触发寄存器，那么将触发相应的功能单元进行相应操作。如果该传输操作表示跳转、函数调用或者自陷处理，那么将设定跳转、函数调用或者自陷处理的延迟槽。

(3)处理器中各部件状态的更新。根据当前指令的运行，对处理器中功能单元、定时器、中断控制器等部件的状态进行更新。如果跳转、函数调用或者自陷操作的延迟周期已到，那么将重新设定 PC 的值。

(4)统计信息。在每一个周期，信息统计模块都将对该周期的相关信息进行统计，一直持续到程序完成为止。

### 2.5 自定义扩展指令的加入

本文设计的模拟器不仅能够对不同结构的 TTA 处理器进行模拟，同时还提供用户自定义指令的加入，这样更进一步拓宽了可重定向的范围，使得该模拟器更加灵活，方便用户使用。

用户自定义指令的加入分为以下几个步骤：

(1)在模拟器源代码中加入自定义指令的名称、汇编指令格式、该指令所表示的操作以及对该指令操作的具体模拟过程；

(2)修改体系结构描述文件，将用户自定义指令加入到体系结构描述文件之中；

(3)在用户编写的高级语言程序里使用该自定义指令的名称；

(4)对模拟器源代码进行重新编译。

当然，基于 TTA 的嵌入式 ASIP 设计框架中的编译器也应该进行相应的修改，使得高级语言经过其编译器编译，能够将用户自定义指令编译成相应的二进制代码。此问题不在本文的讨论范围之内。

## 3 模拟流程

如图 4 所示，用高级语言或者汇编语言编写的应用程序经过可重定向编译器或者汇编器进行编译后，生成二进制代码。二进制代码和体系结构描述文件一起，作为模拟器的输入。模拟器收到输入后，首先由预处理模块对体系结构描述文件进行解释，生成对应的处理器模型，同时生成一个译码映射信息文件。预处理模块根据处理器模型和译码映射信息文件对二进制代码进行预处理，将译码后的指令放入程序模型的指令队列里。接下来，模拟器控制模块从指令队列里读取指令进行模拟执行，直到整个程序正确结束。在模拟期间和结束之后，信息统计模块将对模拟过程中的一些信息进行统计，并按需求反馈给用户。

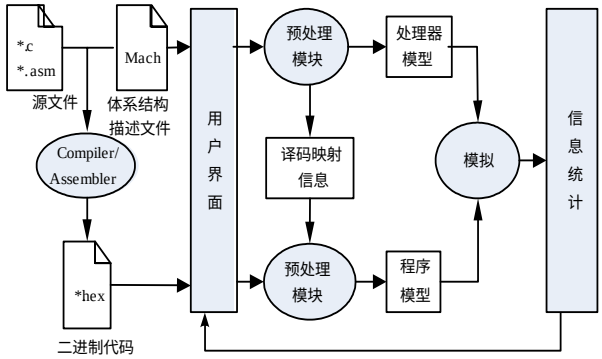


**图 4 模拟流程**

## 4 实现和测试结果

该模拟器在 Linux 操作系统下用标准 C 语言进行实现，并采用 GNU 编译器“GCC”进行编译。在对体系结构描述文件进行解释时，使用了 Python 语言。模拟器实现完成之后，采用了不同的程序在模拟器上模拟运行。功能性验证通过比较模拟结果和已知的正确结果来进行；模拟性能则通过应用程序被模拟的时间来评测。模拟器运行平台拥有 1.8GHz 主频的 Intel Pentium 4 处理器和 1GB 内存，操作系统为 Linux 2.4.27。

模拟程序采用Viterbi算法[6]、IDCT算法和FFT算法[7]。目标处理器体系结构由连接在 8 条全相连总线上的 15 个功能单元所构成。通用寄存器分布在 4 个寄存器文件中。Viterbi程序用C语言编写，后两个用汇编语言编写。为了获取更加精确的模拟时间，每个程序都运行了多遍。模拟结果如表 1 和表 2 所示。表 1 给出了指令模拟速度，表 2 给出了对move的模拟速度。从结果可以看出，每条指令里面能同时执行的move数大概是 5.2。

**表 1 指令模拟速度**

| | Instructions/s | Host cycles/instruction | Total Instructions |
|---|---|---|---|
| Viterbi | 668 900 | 4 190 | 1 799 953 000 |
| IDCT | 543 200 | 4 180 | 1 233 453 000 |
| FFT | 889 300 | 5 030 | 2 300 403 000 |

**表 2 move 模拟速度**

| | Moves/s | Host cycles/move | Total Moves |
|---|---|---|---|
| Viterbi | 3 478 280 | 838 | 6 961 272 000 |
| IDCT | 2 820 450 | 819 | 6 233 683 000 |
| FFT | 4 443 200 | 986 | 10 400 320 000 |

## 5 结论

本文给出了一种面向 TTA 结构的可重定向周期精确模拟器的详细设计和实现。本模拟器的设计与实现是整个 TTA 设计框架实现中的一环。在本模拟器的设计与实现中，采用了对应用程序的预解释机制从而提高了模拟速度；同时，提供了对用户自定义指令的添加接口，增强了整个模拟器的灵活性。采用标准 C 语言和 Python 语言进行模拟器的实现，而不是选择一种新的建模语言，是源于 C 语言的广泛使用和对其进行支持的大量稳定的附加工具。

在今后的工作中，将致力于开发更友好的图形用户界面，同时将该软件模拟器和硬件模拟验证工具结合起来，构建一个完整的系统建模环境。

**致谢** 感谢荷兰 Delft 科技大学的 Andrea Cilio 及其同事对本研究工作的进展提供的大力支持和帮助。

### 参考文献

1 岳 虹, 沈 立, 戴 葵. 基于 TTA 的嵌入式 ASIP 设计[J]. 计算机研究与发展, 2006, 43(4): 752-758.

2 Sykes D A, Molloy B A. The Design of an Efficient Simulator for the Pentium Pro Processor[C]//Proceedings of the 28th Winter Conference on Simulation Table of Contents, Coronado. 1996: 840-847.

3 Molloy B. The Validation of a Multiprocessor Simulator[C]// Proceedings of the Winter Simulation Conference. 1993: 625-631.

4 Zivojnovic V. DSP Processor/Compiler Co-design: A Quantitative Approach[D]. Aachen University of Technology, 1998.

5 Corporaal H. Transport Triggered Architectures: Design and Evaluation[D]. Delft Univ. of Technology, 1995-09.

6 Viterbi A J. Error Bounds for Convolutional Coding and an Asymptotically Optimum Decoding Algorithm[J]. IEEE Transactions on Information Theory, 1967, 13(2): 260-269.

7 Texas Instruments Inc. TMS320C64x DSP Library Programmer's Reference[Z]. 2002-04.

(上接第 258 页)

### 参考文献

1 法国电信北京研究院. 移动增值业务 IVR[J]. 当代通信, 2004, (24): 54.

2 阿与. 语音增值业务(IVR)基础知识[J]. 中国数据通信, 2004, (7): 63-65.

3 熊桂林, 邹志敏. 基于 SMS 的增值服务平台的设计与实现[J]. 系统工程, 2003, 21(1): 59-62.

4 Bray T, Paoli J, Sperberg-McQueen C M, et al. Extensible Markup Language (XML) 1.0[Z]. 2004-02-04.http://www.w3.org/TR/REC-xml/.

5 McGlashan S, Burnett D C, CarterVoice J, et al. Extensible Markup Language (VoiceXML)[Z]. 2004-03-16. http://www.w3c.org/TR/voicexml20.