第37卷 第14期 计 算 机 工 程 2011年7月
Vol.37 No.14 Computer Engineering July 2011

·工程应用技术与实现· 文章编号：1000—3428(2011)14—0248—03 文献标识码：A 中图分类号：N945

基于 TLM2.0 的 SPARC 事务级建模

周海洋[1,2]，葛 宁[1]，于立新[2]，李玉红[2]

(1. 清华大学电子工程系，北京 100084；2. 北京微电子技术研究所，北京 100076)

摘 要：为提高可扩展处理器体系结构(SPARC)的设计抽象层次和仿真速度，设计一种符合第8版SPARC(SPARC V8)的事务级模型。该模型基于TLM2.0标准，采用解释型指令集仿真方法实现程序执行。通过构建验证环境，证明该事务级模型能够正确运行并跟踪SPARC V8程序，仿真速度比寄存器传输级提高2个数量级。

关键词：可扩展处理器体系结构；事务级模型；指令集仿真；仿真速度；寄存器传输级

Transaction Level Modeling of SPARC Based on TLM2.0

ZHOU Hai-yang[1,2], GE Ning[1], YU Li-xin[2], LI Yu-hong[2]

(1. Department of Electronic Engineering, Tsinghua University, Beijing 100084, China;
2. Beijing Microelectronics Technology Institute, Beijing 100076, China)

【Abstract】In order to raise the level of design abstraction and simulation speed of Scalable Processor Architecture(SPARC) processor, a SPARC V8 processor Transaction Level Model(TLM) is designed in this paper. This model is based on TLM2.0 standard and interpretive instruction set simulation technology. Results of the test show that the model can execute and trace SPARC V8 programs exactly and the simulation speed of it is improved two orders of magnitude than Register Transfer Level(RTL) design.

【Key words】Scalable Processor Architecture(SPARC); Transaction Level Model(TLM); instruction set simulation; simulation speed; Register Transfer Level(RTL)

DOI: 10.3969/j.issn.1000-3428.2011.14.084

1 概述

随着片上系统(System on a Chip, SoC)功能的不断增加，从RTL开始设计高性能的SoC将花费大量时间，这极大地延迟了产品的面世日期。为了缩短SoC设计周期，事务级建模应运而生[1-2]。

TLM2.0(Transaction Level Modeling standard, version 2.0)是OSCI(Open System C Initiative)为统一事务级建模方法，以SystemC和TLM1.0(Transaction Level Modeling, version 1.0)为基础制定的事务级建模标准[2]。符合TLM2.0的事务级模型，因为使用了统一的编码风格和接口，所以具有良好的通用性，使得不同设计者设计的事务级模型之间能够便捷、快速地完成互联。

本文针对在嵌入式系统中应用越来越广泛的SPARC V8处理器，基于TLM2.0标准设计了该处理器的一种事务级模型，并建立该模型的验证环境，给出了功能和速度的仿真结果。

2 SPARC V8事务级模型设计

2.1 SPARC V8处理器简介

SPARC是一种指令集体系结构(Instruction Set Architecture, ISA)，V8表示版本，其显著特点是寄存器窗口。根据SPARC V8 ISA手册[3]，SPARC V8处理器由执行单元(Execute Unit, EU)和通用寄存器堆(regfile)组成，如图1所示。其中，EU负责取指、译码、执行、访存和写回；regfile负责保存指令执行用的数据和执行结果。同时，EU具有指令缓存接口和数据缓存接口，EU执行的指令从指令缓存接口读入，指令执行中用到的数据从数据缓存接口读入，需要写入外部存储器的指令执行结果通过数据缓存接口输出。

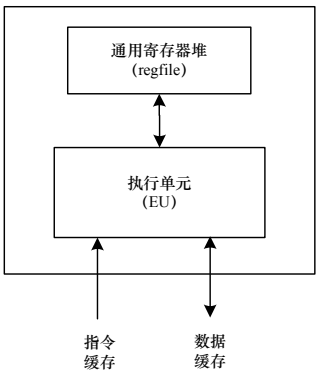


图1 SPARC V8处理器的基本组成

2.2 事务级模型设计

在设计事务级模型时，需要分别对目标的通信功能和计算功能建模，通信模型负责事务的传输，计算模型负责事务的处理[4]。

这种建模方法在TLM2.0标准中通过发起者(Initiator)和目标(Target)实现。事务的计算发生在Initiator和Target的内部，而事务的通信发生在Initiator和Target之间。因此，基于TLM2.0标准的事务级模型基本结构如图2所示，其中通用负载(payload)是TLM2.0中定义的承载事务的对象，Initiator和Target之间通过payload交换事务；而socket则用于连接Initiator和Target。

作者简介：周海洋(1981－)，男，工程师、硕士，主研方向：微处理器仿真，嵌入式系统设计；葛 宁，教授、博士；于立新，研究员；李玉红，工程师、硕士

收稿日期：2011-01-17 **E-mail：**zhysnail@126.com

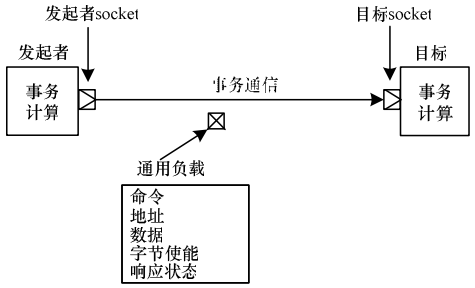


图 2　基于 TLM2.0 的事务级模型基本结构

根据上述事务级建模方法，建立 SPARC V8 处理器事务级模型应首先分析处理器内部的计算和通信，从而分离它们。之后，按照 TLM2.0 分别对通信和计算进行建模。

SPARC V8 处理器内部的计算和通信分析结果见表 1。其中，EU 的通信包括对 regfile 的读/写、读指令缓存接口和读/写数据缓存接口 3 个部分，计算包括指令译码、执行和更新处理器状态 3 个部分；regfile 的通信是响应 EU 的通信要求，返回寄存器值，计算部分包括定位当前工作的寄存器窗口和读/写当前寄存器值 2 个部分。

表 1　SPARC V8 处理器内通信和计算分析

建模对象	通信	计算
执行单元 (EU)	读/写 regfile	译码
	读指令缓存接口	执行
	读/写数据缓存接口	更新处理器状态
通用寄存器堆 (regfile)	响应 EU 的请求，返回寄存器值	当前工作窗口定位 读写当前工作窗口中的寄存器

根据 SPARC V8 处理器计算、通信分析结果，按照 TLM2.0 标准，SPARC V8 处理器事务级模型的总体设计结构如图 3 所示。

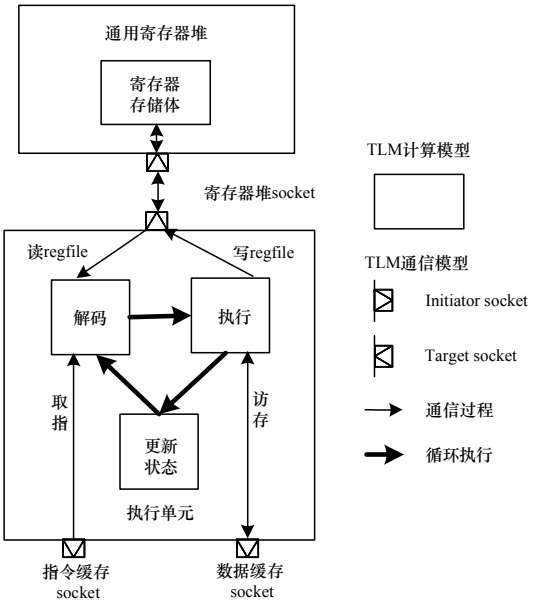


图 3　SPARC V8 事务级模型的总体设计结构

EU 是对指令缓存、数据缓存和 regfile 发起通信的一方，因此 EU 模块定义为 Initiator，并具有 3 个 initiator socket 作为通信接口。EU 从指令缓存 socket 中读取的是 32 位长的指令，通过数据缓存 socket 传输的是 32 位宽的数据；EU 内部包含 3 个循环执行的方法，分别完成指令的解码、执行和处理器状态的更新。

regfile 响应 EU 的操作要求，是接收通信事务的一方，因此定义为 Target，并具有一个 Target socket 作为通信接口。regfile 内部包含完成对寄存器读写的方法。

下面分别介绍 EU 和 regfile 的详细建模过程。

2.2.1　执行单元建模

从通信和计算 2 个方面建模 EU。

(1)EU 的通信建模

EU 具有 regfile、指令缓存和数据缓存 3 个通信接口，每个接口需要不同的处理方法，因此采用 TLM2.0 tlm_utils 中定义的具有 callback 功能的 simple_initiator_socket 实现 3 个 Initiator socket。当产生通信时，3 个 socket 调用绑定好的 callback 函数进行处理，保证用正确的方法对通信数据进行处理。

(2)EU 的计算建模

EU 中解码、执行和状态更新用 SystemC 的 SC_THREAD 实现。解码方法 decode_inst_method()完成 SPARC V8 二进制指令的分析，得到当前指令操作及操作数；执行方法 execute_inst_method()根据解码方法的结果读取 regfile，完成指令执行，并将结果写入到 regfile 和/或外部数据缓存；状态更新方法 status_updated_method()根据执行方法的结果更新 SPARC V8 处理器中的状态寄存器，如当前程序计数器 PC、处理器状态寄存器 PSR 等。3 个方法循环执行，直到程序退出。

3 个方法中解码方法是 EU 计算建模的重点，考虑到译码过程的灵活性和可观测性，本文选用解释型译码方法[5-7]。

译码方法根据图 4 所示的 3 种 SPARC V8 指令格式[3]，按照如下流程解析指令。

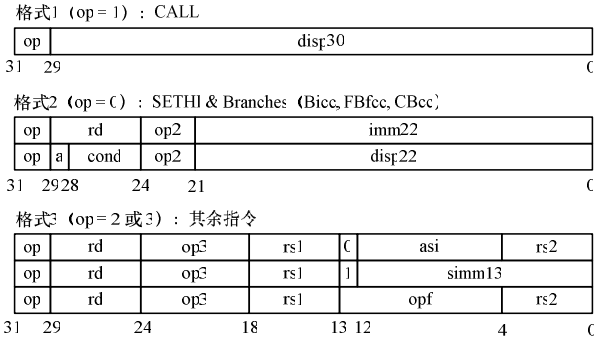


图 4　SPARC V8 的 3 种指令格式

SPARC V8 程序译码流程如下：

(1)指令字段分解赋值到对应变量。

(2)使用 switch-case 结构，逐字段解释指令，得到当前指令并记录在变量 current_instruction 中，过程如下：

```
switch(op)
{
 case 1: //调用指令
        current_instruction = CALL; break;
 case 0：//SETHI 和分支指令
        switch(op2)：
        {
              case 2：//Bicc
                    switch(cond){ … }
              case 4: //SETHI
                    switch(rd){ … }
              …
}
        case 2: //算术、位移等指令
              switch(op3)：{ … }
        case 3: //访存指令
              switch(op3)：{ … }
```

}

(3)结束解码，进入执行方法。

2.2.2 通用寄存器堆建模

regfile 的建模也从通信和计算 2 个方面进行。

(1)regfile 的通信建模

regfile 作为与 EU 通信的 target，采用 tlm_target_socket 实现通信接口，处理通信时直接调用方法函数。

EU 与 regfile 通信中使用的 payload 是需要关注的环节，这是由于TLM2.0的payload设计用来完成基于地址映射的存储器操作，只有一个地址属性，一次传输只能对地址属性记录的地址或以该地址为起始的一组连续的地址操作。而在 EU 操作 regfile 时用寄存器名称作为索引，同时 SPARC V8 指令多数是三操作数指令(2 个源操作数、1 个目的操作数)，保存源操作数的寄存器不一定相邻。所以，如果直接使用 payload 中的地址属性保存寄存器名称，EU 与 regfile 之间一次通信只能读/写一个寄存器，通信效率低，也与实际不符。因此，需要改进 payload，使其可以用寄存器名称作为索引，在一次传输中读/写多个寄存器。

考虑到 payload 的数据属性由一个指针实现，并具有数据长度属性，可以灵活地控制发送数据的长度和内容，所以定义 regfile 数据帧，作为 payload 的数据部分。

regfile 数据帧由寄存器名和数据组成，如图 5 所示。一个寄存器名和数据为一组，一次通信中操作几个寄存器就有几个数据帧。多个 regfile 数据帧连接起来组成 payload 的数据段，数据帧的总长保存在 payload 的数据长度属性中。由于 EU 对 regfile 的通信发生在解码方法和执行方法，前一次是读，后一次是写，不存在一次通信中即读又写，因此直接使用 payload 本身的操作属性就可以标示出所有 regfile 数据帧是读操作还是写操作。

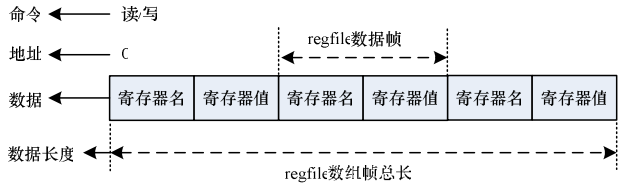


图 5 执行单元与 regfile 通信的 payload

(2)regfile 的计算建模

regfile 中采用一维数组实现寄存器窗口，并设置一个指针指向当前工作窗口。由于 SPARC V8 寄存器窗口间具有输入寄存器和输出寄存器交叠的特性，并使模型具有可以配置寄存器窗口数量的能力，因此存储体窗口数组长度和寄存器定位分别按式(1)、式(2)进行：

存储体窗口数组长度=窗口数×16 (1)

寄存器定位=(窗口指针×2+1)×8±偏移 (2)

其中，偏移是当前工作寄存器窗口中的寄存器名称。

3 SPARC V8 事务级模型的验证

3.1 验证环境设计

为验证 SPARC V8 事务级模型的功能，验证环境需要模拟程序存储器。由于程序存储器一般是单口 RAM，而本文设计的模型具有指令缓存和数据缓存 2 个接口，因此还需要 1 个同步模块将 2 个缓存接口与单端口存储器接口连接在一起。同时，验证环境也要符合 TLM2.0。据此，验证环境的结构如图 6 所示。程序存储器(memory)是通信的 Target，其中内部负责模拟存储体行为。存储体有一个大小可设的数组模拟，数组索引作为存储体的地址。该模块中内建 ELF 文件加载器，通过解析 ELF 格式的程序，将指令、数据等写入数组的相应位置，供仿真事务级模型时读/写。同步模块(memory wrap)在与处理器事务级模型通信过程中是通信的接收方，在与 memory 的通信过程中是通信的发起方，所以，memory wrap 具有 2 个 Target socket 和 1 个 Initiator socket。为保证从 memory 读出的数据可以正确返回到读取数据的 socket，2 个 Target socket 之间采用 1 个信号量进行同步。

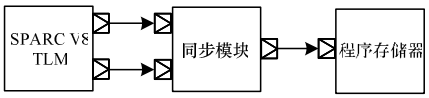


图 6 SPARC V8 处理器 TLM 验证环境

3.2 验证方法及结果

在 CPU 为 Intel P4 2.93 GHz、2 GB 内存、使用 WindowsXP 为操作系统的宿主机上用 VMware6 建立 CPU 为 Intel P4 2.93 GHz、512 MB 内存、使用 Redhat9 为操作系统的虚拟机。SPARC V8 事务级模型用 GCC 编译并运行于虚拟机上，仿真 Stanford baby benchmark suite，模型执行结果与 RTL 相同，验证模型能够正确执行 SPARC V8 处理器的程序，功能正确。

该事务级模型可以详细报告每条 SPARC V8 指令的执行情况，如下所示，包括指令地址、指令名、源操作数、指令执行过程和结果、处理器当前状态等信息。这一特点对调试程序，特别是需要充分利用 SPARC 寄存器窗口特性的程序有很大帮助。

```
0x40001f70: save %o6, 0xffffff88, %o6 ！  %o6 =
0x400ffff0 %o6 = 0x400fff78 now window No. = 0x7
0x40001f74: or %g0, 0x1, %i0 ！  %g0 = 0 %i0 = 0x1
0x40001f8: st %i0, [%i6 + 0xffffff4] ！  address =
0x400ffff0 + 0xffffffc4 = 0x400fffe4 %i0 = 0x1
```

经过多次仿真，并记录每次仿真时间，SPARC V8 事务级模型平均每秒执行 17 647.934 条指令，与使用 QuestaSim6.3d 在宿主机上仿真的 RTL 模型执行相同程序的速度比较结果如图 7 所示，事务级模型的仿真速度比 RTL 快约 123 倍。

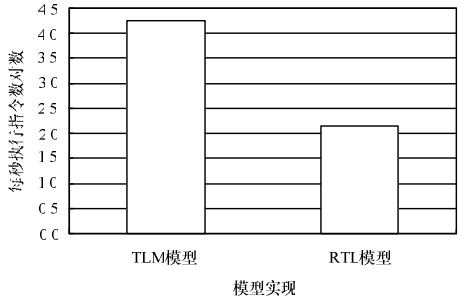


图 7 SPARC V8 TLM 与 RTL 模型的速度比较

4 结束语

本文基于 TLM2.0 标准实现了一种符合 SPARC V8 指令集体系结构的事务级模型，将 SPARC V8 处理器的设计抽象层次从 RTL 提高到事务级，仿真速度提高了 123 倍，并可以报告指令的详细执行过程。由于采用了 SystemC 和 TLM2.0，该 SPARC V8 事务级模型具有良好的平台无关性，既可以嵌入 EDA 工具进行 SoC 结构设计和验证，也可以作为 SPARC V8 指令集仿真器进行软件开发，该模型有利于加速以 SPARC V8 处理器为核心的 SoC 的研究、设计和应用。

(下转第 253 页)