第 24 卷第 1 期　　半 导 体 学 报　　Vol. 24, No. 1
2003 年 1 月　　CHINESE JOURNAL OF SEMICONDUCTORS　　Jan., 2003

面向寄存器的流水线处理器建模及验证方法

何 虎　孙义和

(清华大学微电子学研究所，北京　100084)

摘要：提出了一种新的流水线处理器功能的验证方法，这种方法的主要思想是通过验证流水线处理器中所有寄存器的功能来验证处理器的功能. 流水线处理器绝大部分是由同步电路组成的，同步电路的状态则完全由寄存器的状态决定，因此如果能够保证每个寄存器功能正确就可以保证整个同步电路功能正确. 对于流水线处理器来说，寄存器状态的变迁是由处理器的原始输入和寄存器本身状态决定的. 原始输入包括控制信号(如复位信号)和数据输入(如指令输入). 如果把对每个寄存器的赋值操作转换成对控制信号和数据输入的操作，就可以生成一个验证序列，这个序列包括每个时钟周期控制信号和数据输入的值. 有了这个序列就可以把目标设计和参考模型进行结果比较，从而验证目标设计功能是否正确. 同时这种方法也便于调试.

关键词：面向寄存器；设计验证；流水线处理器；形式验证

EEACC：1265F；1265A　　**CCACC**：6110F

中图分类号：TN431.2　　**文献标识码**：A　　**文章编号**：0253-4177(2003)01-0098-06

1　引言

目前流水线处理器的验证方法包括仿真方法[1,2]和形式验证方法[3~5]. 参考模型的建模方法因不同的验证方法而各不相同. 仿真建模方法包括ISA、RTL、总线功能等. 形式验证的建模方法与形式验证采用的方法是对应的，比如用 BDD[6]，implementation mapping[5] 等等. 仿真方法的优点是方法简单，搭建验证系统快；缺点是可能无法对处理器的所有功能做验证. 形式验证的方法是用数学的方法证明验证对象的功能与参考模型的功能是否一致. 它的优点是能够对处理器做完备的验证；缺点是整个证明过程比较复杂，需要许多辅助软件和较多的人力才能完成较大规模处理器的验证. 目前商业系统中一般采用形式验证的方法. 这样可以保证产品的可靠性.

本文提出的方法介于这两种方法之间，既可以高效地搭建验证系统，同时可以保证验证的可靠性. 在同步电路里，所有寄存器和输入端口的状态惟一决定了电路的当前状态和下一个状态. 因此如果能够验证所有寄存器的功能正确，则可以验证整个同步电路的功能正确. 为了实现这个目标，首先需要一个参考模型详细描述寄存器的功能. 在每个处理器的 SPEC 中都会给出必要的寄存器的描述. 利用这些寄存器，再加上一些必要的辅助寄存器就可以描述出整个处理器的模型. 由于模型中寄存器的种类不多，因此就可以对每个寄存器做功能验证. 功能验证最终生成指令序列，然后将指令序列作为验证向量加载到被测处理器描述上，就可以最终验证处理器功能是否符合模型要求.

本文第 2 节介绍要验证的流水线处理器和一个 SPARC V8[7] 的实现 LEON v1.0. 第 3 节介绍如何搭建参考模型. 第 4 节介绍如何用面向寄存器的方法构造验证向量. 第 5 节给出验证结果. 第 6 节是全文总结，并提出将来工作任务.

何　虎　男，博士研究生，研究方向是集成电路的测试和验证.

孙义和　男，1945 年出生，教授，博士生导师，主要研究方向为 LSI/VLSI 和 SOC 测试方法学和可测性设计、多媒体 VLSI 和 SOC 设计技术、网络和数据安全 VLSI 和 SOC 结构.

2002-04-05 收到，2002-08-14 定稿

2　SPARC V8

SPARC 是一套从精简指令系统(RISC)发展而来的指令集体系结构. 它定义了 32 位的整数和 32、64、128 位的浮点数作为它的基本数据类型. 它同时还定义了一些通用的整数、浮点和处理器状态的寄存器以及 72 条基本指令. 这些指令都是 32 位字长编码. 读取/存取指令可以直接访问 2^{32}字节大小的地址空间. 除了浮点指令以外, SPARC 还定义了支持用户自定义协处理器的指令集. SPARC 最大的特点就是只定义了一种大致的体系结构, 使用者可以根据自己的需要设计出符合 SPARC 结构的各种处理器.

通常 SPARC 包含一个整数单元、一个浮点单元和一个协处理器. 本文用到的验证对象 LEON-lv1.0 是 European Space Agency 实现的一种 SPARC 结构. 它的源代码公布在网络服务器[7]上. LEON 主要是用于嵌入式系统. LEON 的主要部件是整数单元(IU), 还包括指令、数据缓存、存储器接口和外围设备. 由于用途的限制, LEON 不包含浮点处理单元和协处理器单元. 本文的建模和验证对象就是 LEON 的整数单元. LEON 的整数单元是一个有 5 级流水线的指令执行器. 它包含了 SPARC V8 定义的部分通用寄存器. LEON 具有 Fetch、Decode、Execute、Memory 和 Write 五个流水级[1]. 指令从指令缓存 I-cache 进入流水线, 分别在不同的流水级完成不同的工作, 最终数据写入数据缓存 D-cache 或寄存器堆 regfile.

3　整数单元模型构造方法

3.1　建模方法

参考模型是用 C 语言来实现的. 建模的基本思想是基于一些选定的寄存器和原始的输入/输出来实现 ISA(instruction set architecture)中定义的功能. 由于验证方法是面向寄存器的, 因此希望参考模型中的寄存器越少越好, 只要能够描述出功能模型就可以了. 在 3.2 节中会详细描述寄存器的选择.

面向寄存器的验证方法验证的对象是同步电路. 同步电路的当前状态可以由寄存器的当前状态惟一决定, 同时同步电路的下一个时钟周期的状态也由当前寄存器的状态和当前原始输入的状态惟一决定. 定义同步电路下一个状态为 S, 寄存器当前值定义为 V_{r_i}, r_i 表示某个寄存器, V_{p_i}表示原始输入的状态, p_i 表示某个原始输入. 那么下式成立:

$$S = f(V_{r_i}, V_{p_j}) \tag{1}$$

其中　f 是状态转移函数. 如果可以实现验证所有 f 定义的功能, 那么就可以表示已经验证了整个同步电路的功能. 由于f 是一个非常复杂的函数, 因此很难直接用它来验证. 为了降低单次验证的复杂度, 本文验证方法采用以寄存器为对象的分割方法. 定义 $R = \{r_1, r_2, r_3, \cdots, r_n\}$是验证对象所有寄存器的集合. V_{r_1}为寄存器 r_1 的值. 同步电路的当前状态表示为 $S = V_R$, 即所有寄存器的当前值为同步电路的状态. 用算式表示为:

$$\begin{aligned} V_{r_1} &= f_1(V_{r_1}^{-1}, V_{r_2}^{-1}, V_{r_3}^{-1}, \cdots, V_{r_n}^{-1}, V_{S_{PI}}) \\ V_{r_2} &= f_2(V_{r_1}^{-1}, V_{r_2}^{-1}, V_{r_3}^{-1}, \cdots, V_{r_n}^{-1}, V_{S_{PI}}) \\ &\cdots \\ V_{r_n} &= f_n(V_{r_1}^{-1}, V_{r_2}^{-1}, V_{r_3}^{-1}, \cdots, V_{r_n}^{-1}, V_{S_{PI}}) \end{aligned} \tag{2}$$

其中　$V_{S_{PI}}$代表原始输入集合的值. 如果分别验证 $f_1, f_2, \cdots, f_n$ 是正确的, 那么就可以证明整个同步电路功能是正确的.

由于用硬件描述语言实现同步电路的过程中, 为了提高电路的性能, 会定义许多寄存器. 寄存器数量众多导致(2)式非常复杂. 如果可以减少寄存器的种类, 压缩(2)式的空间复杂度, 提高(2)式的时间复杂度, 就可以用较少的寄存器种类来实现状态转移函数. 比如寄存器的值表示为:

$$V_{r_i} = f_i(V_{r_1}^{-1}, V_{r_2}^{-1}, V_{r_3}^{-1}, \cdots, V_{r_n}^{-1}, V_{S_{PI}}) \tag{3}$$

其中　V_{r_j}的值可以表示为:

$$V_{r_j}^{-1} = f_j(V_{r_k}^{-2}, V_{r_l}^{-2}, V_{S_{PI}}^{-1}) \tag{4}$$

其中　$V_{r_k}^{-2}$, $V_{r_l}^{-2}$分别表示寄存器 r_k, r_l 上一时钟周期的值. 那么:

$$V_{r_i} = f_i(V_{r_1}^{-1}, V_{r_2}^{-1}, \cdots, f_j(V_{r_k}^{-2}, V_{r_l}^{-2}, V_{S_{PI}}^{-1}), \cdots, V_{r_n}^{-1}, V_{S_{PI}}) \tag{5}$$

$$= F(V_{r_1}^{-1}, V_{r_2}^{-1}, V_{r_3}^{-1}, \cdots, V_{r_{j-}}^{-1}{}_1, V_{r_{j+}}^{-1}{}_1, \cdots, V_{r_n}^{-1}, V_{r_k}^{-2}, V_{r_l}^{-2}, V_{S_{PI}}, V_{S_{PI}}^{-1}) \tag{6}$$

(6)式中虽然变量增多了, 但是寄存器种类减少了. (2)式中转移函数的种类就减少了. 这种减少是以时间复杂度的增加为代价的. 在建模过程中必须减少参考模型中寄存器的种类, 同时又不会过多地引用寄存器的历史数据. 如果寄存器选择得当, (2)式的

转移函数种类可以非常明显地减少.

当一条指令进入流水线后,根据 ISA 定义的行为需要读取某些寄存器的值,然后修改某些寄存器的值.如果用于读取的寄存器存在于描述中,那么就可以直接获得这个寄存器的值.因为没有选择足够多的寄存器来建模,因此有可能存在这种情况,那就是没有一个寄存器来保存需要读取的值.这个值不能直接从当前时刻的某个寄存器中得到.为了得到这个值,就需要上溯一个时钟周期,从保存的上一个时钟周期的寄存器的值和当前寄存器的值共同计算出这个值.如果用上一个时钟周期的值和当前时刻的值还是不能算出来,那么就需要再上溯一个时钟周期,直到能够计算出这个值为止.这个上溯的次数就是需要保存的历史数据的深度.如果寄存器选择不当,很有可能出现不管上溯多少次,都无法通过已有的寄存器计算出需要的值.因此建模中最重要的一项工作就是选定合适的寄存器.

建模以后寄存器标准形式可以如图 1 所示.每个寄存器的赋值都会表示成如下形式:

$$r_{\mathrm{dest}} = f(r_{\mathrm{src1}}{}^{-x_1}, r_{\mathrm{src2}}{}^{-x_2}, r_{\mathrm{src3}}{}^{-x_3}, \cdots, Vs_{PI}) \quad (7)$$

其中 r_{dest} 是目标寄存器,它表示需要验证的寄存器;r_{src} 是源寄存器,它提供计算 r_{dest} 所需要的数据;f 是布尔表达式.r_{src} 可以分为两种:一种是控制寄存器,一种是数据寄存器.控制寄存器的作用是选择数据通道.数据寄存器的作用是提供数据通道需要的运算数据.在验证过程中,数据寄存器的值并不用关心,需要关心的是控制寄存器的值.如果可以找到控制寄存器所有可能的数据组合,并且把这种组合最终转换成指令序列,那么就可以利用这个指令序列来验证目标设计是否正确.如果可以完全遍历控制寄存器的所有组合,那么从理论上就可以认为目标设计得到完全验证.但是在实际验证中如何分离这两种寄存器并不容易,它取决于 f 的具体表达.为了自动完成控制寄存器的搜索并确定控制寄存器的值,需要一个 C 语言的语法分析程序和一个 BDD (binary decision diagram)[2] 的构造程序.C 语言语法分析程序负责找出参考模型中所有的程序分支.BDD 构造程序负责为每一个程序分支生成 BDD.最后通过 BDD 找出所有可能的控制寄存器和控制寄存器的值.这部分计算在下一节中会详细介绍.

3.2 寄存器的选择

首先选定的寄存器是在 ISA 中定义的.比如在

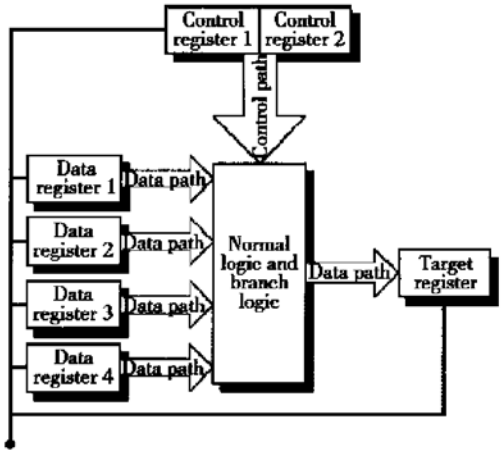

图 1 面向寄存器的建模方法

Fig. 1 Register-oriented modeling method

SPARC V8 Manual 中详细定义了许多整数单元的状态寄存器和 r 寄存器.除去这些寄存器之外还需要定义一些辅助寄存器.辅助寄存器选定的原则是:第一,能够保证辅助寄存器加上 ISA 中提到的寄存器可以完整地描述整个处理器的行为.第二,将历史数据的深度减小到流水线级数以下.比如那些用于多周期指令的计数器.如果不选择这些计数器,那么就需要上溯到多周期指令进入流水线的那一时刻才能计算出当前计数器的值.而在 LEON v1.0 实现中,乘法运算需要 35 个时钟周期才可以完成.这需要占用大量的存储器来保存这么多的历史数据.如果将计数值保存在寄存器中就可以避免这种情况发生.第三,便于生成验证指令序列.参考模型的一个作用就是用它生成验证用的指令序列.指令序列的入口是模型的原始输入,然后指令在流水线中一级一级地被执行.为了清楚每一级执行了什么指令,就需要保存每一级的指令寄存器.根据以上三点就可以找到需要的寄存器,然后用 3.1 节的建模方法构造参考模型.

4 验证指令序列的自动生成

首先要指出的是指令序列并不是单纯的软件指令,它还包括原始输入需要加载的数据.最后加载到目标系统上的验证激励是软件程序和原始输入的组合.这样才能够完整地体现验证激励.

4.1 常用指令序列库的生成

在模型设计中,寄存器是设计的主体.改写一个

寄存器的值有许多方式. 这里的方式指的是数据路径, 比如 SPARC V8 的处理器状态寄存器 PSR 可以被许多指令改写. 但是通常一个寄存器会有一种通过程序控制的最简单的改写方式, 比如 PSR 就可以通过 WRPSR 指令直接改写. 但是单独一条 WRPSR 指令是一条 Supervisor Mode 的指令, 因此在输入这条指令时还需要将处理器设置成 Supervisor Mode. 根据 SPARC V8 定义的内容, 处理器在进入陷阱以后处于 Supervisor Mode, 于是可以用指令先产生一个陷阱, 然后在陷阱里执行 WRPSR 指令就可以改写 PSR 寄存器的内容, 最后从陷阱返回. 改写 PSR 的方式有很多, 如果把这些最简单、最直接的改写方式用库的形式保存起来, 这样当验证指令序列生成过程中需要这些寄存器处于某个值的时候, 就可以从指令序列库中提取这些指令序列.

库的建立方法是将所有的外部原始输入和程序代码用一个带时序性质的表达式表达. 假定某个验证序列是要将 PSR 设置成(xxxxH). 首先执行一条 Ticc 指令, 然后调用 WRPSR 指令设置 PSR 的值, 最后用 RETT 返回. 这一套指令对除 PSR 寄存器以外的寄存器影响最小, 故而可以采用这段代码. 用符号表示为

{pi: Reset, instr: Ticc(trapaddr), instr: WRPSR(xxxxH), instr: RETT}

其中　pi 表示原始输入; instr 表示指令. 括号里的内容是指令参数, 可以根据不同的需要修改这些参数. 对于一个寄存器来说不单单有一个这样的指令序列保存在指令库中. 在实际建库过程中需要为一个寄存器建立几个这样的指令序列, 以保证在验证指令序列生成过程中提供多种可能的选择, 使得最后的指令序列验证效率最高.

4.2　验证指令序列的生成

在 3.1 节中介绍了模型的构建方法. 其中控制寄存器的数据组合是验证指令序列自动生成的一个重要内容. 下面介绍如何找出能够遍历参考模型代码的控制寄存器的所有组合.

在参考模型描述中, 对于分支代码的设计采用的是 if…else…和 switch…case…结构, 没有 while 和 for 循环. 每一个函数的入口点设为第一层逻辑, 每进入一个 if…else…或 switch…case…结构, 逻辑层增加一层. 第一层是确定被执行的, 其他层需要用 BDD 求出逻辑表达式成立的所有控制寄存器的值. 验证程序先用 C 语言语法分析器分析参考模型, 给出进入所有层次的逻辑表达式, 然后调用 CUDD[8] 软件包的工具构造 BDD, 最后给出进入某个逻辑层需要的控制寄存器的值.

控制寄存器可以分成两种: 一种是可以用从指令序列库中提取出的指令序列访问的寄存器, 一种是各个流水线级中的指令寄存器. 当有了某个控制寄存器数据组合以后, 首先区分这两种寄存器, 如果是第一种寄存器, 那么从指令序列库中找出对应的指令序列; 如果是第二种, 判断有没有这样一条指令可以满足要求. 如果可以找到这样的指令就把它插入指令序列中. 搜索寄存器的顺序是从最早的历史记录开始, 然后根据时间先后依次生成指令. 在后面生成的指令注意不要影响到前面寄存器的值. 这也是为什么在指令序列库中要为一个寄存器保存多个指令序列的原因之一.

下面介绍验证指令生成算法:

步骤 1: 选定一个没有验证的目标寄存器. 从还没有验证过的寄存器集合中随机选取一个作为目标寄存器. 如果再没有目标寄存器, 则完成对目标电路的验证指令生成.

步骤 2: 统计在已经生成的验证序列里已经完成了多少控制寄存器数据集合的指令生成. 如果全部完成, 转向步骤 1.

步骤 3: 调用 CUDD 软件包的工具构造 BDD, 列出所有可能的控制寄存器的值组成集合.

步骤 4: 选择集合中一组控制寄存器的值.

步骤 5: 从没有确定指令序列的控制寄存器数据集合中选择一个数据最久远的寄存器(指数绝对值最大者). 判断该寄存器属于第一种还是第二种. 如果是第一种, 则从指令序列库中找出一个序列, 并根据寄存器的值设定其中的参数. 如果是第二种, 则根据寄存器的值确定指令及其参数.

步骤 6: 判断新生成的指令是否与前面的指令发生冲突. 冲突表现为改写了需要保留的寄存器的值. 如果冲突生成, 返回步骤 4 重新确定指令. 如果冲突无法解决, 则把最后确定指令序列的控制寄存器设定为没有确定, 同时从指令堆中删除相应的指令返回步骤 5. 冲突的验证是通过向参考模型输入指令, 然后察看在指令执行完成后寄存器的值是否为规定的值.

步骤 7: 判断是否所有寄存器已经确定完毕. 如果没有, 返回步骤 5. 反之, 则返回步骤 4.

步骤 8: 完成对该目标寄存器的验证指令生成, 返回步骤 1.

4.3 调试

由于参考模型用 C 语言描述, 因此可以用多种调试工具进行调试. 更重要的是在硬件模型仿真和参考模型仿真之间不需要再提供额外的软件进行这两者的同步. 由于参考模型是面向寄存器的, 因此运行的每一条指令(包括对输入) 都与某个寄存器对应. 只要记录寄存器的值, 然后进行比较就可以精确地对错误进行定位.

5 实验结果

本文实验对象是 LEON v1.0 的整数单元. 它是 SPARC V8 定义的一个子集, 不包含 SPARC V8 定义的所有指令. 根据 SPARC V8 的描述, 用 C 语言描述参考模型, 然后生成验证指令序列. 将验证指令分别输入硬件仿真器和参考模型仿真器, 将运行结果进行比对.

在参考模型中总共选定了 8 种寄存器和所有整数单元的输入输出, 其中寄存器包括 SPARC V8 Manual 定义的 5 种控制状态寄存器以及 Windowed r Registers, 再加上每个流水线级的指令寄存器和多周期指令的计数器. 在实现过程中, 参考模型还对整数单元的输入作了一定的简化. LEON v1.0 的指令输入和数据输入都是从缓存读入的. 为了向 LEON v1.0 提供指令输入和数据输入, 参考模型简化了指令输入和数据输入, 不提供指令缓存和数据缓存, 改为直接输入验证序列.

参考模型总共用了 2000 多行 C 语言代码. 最后生成的验证序列有 158210 条指令. 指令包括输入端口操作和汇编指令. VHDL 代码覆盖率达到了 100%. 面向寄存器的验证方法生成的指令序列是可以再压缩的, 在接下来的工作中会提出压缩算法.

6 结论

本文介绍了一种面向寄存器的流水线处理器的验证方法, 其中包括参考模型的构建, 验证序列的生成. 并以基于 SPARC V8 的 LEON v1.0 作为验证对象实现了以上方法. 面向寄存器的验证方法的最大特点是验证对象是寄存器, 它不需要将验证对象作为某种特定的系统, 比如一个有限状态机, 来对待, 而只需要将其作为一个同步电路. 因此这种方法并不一定局限于对流水线处理的功能验证, 所有的同步电路都可以采用这种方法来验证. 只不过最后生成的验证序列可能是 I/O 输入或者各种微码序列. 今后我们的工作包括改进算法, 生成尽量小的验证序列, 并在验证序列生成以后进行压缩等等.

参考文献

[1] Chang You-Sung, Lee Seungjong, Park In-Cheol, et al. Design Automation Conference, 1999 Proceedings, 36th, 1999: 181

[2] Hu A J. Formal hardware verification with BDDs: an introduction. Communications, Computers and Signal Processing, 1997 10 Years PACRIM 1987~ 1997-Networking the Pacific Rim. 1997 IEEE Pacific Rim Conference, 1997, 2: 677

[3] Iwashita H, Kowatari S, Nakata T, et al. Automatic program generator for simulation-based processor verification. Test Symposium, 1994, Proceedings of the Third Asian, 1994: 298

[4] Kaivola R, Narasimhan N. Formal verification of the Pentium (R) 4 multiplier. High-Level Design Validation and Test Workshop, 2001 Proceedings of Sixth IEEE International, 2001: 115

[5] Patankar V A, Jain A, Bryant R E. Formal verification of an ARM processor. VLSI Design, 1999 Proceedings of Twelfth International Conference, 1999: 282

[6] Hazelhurst S, Seger C-J H. A simple theorem prover based on symbolic trajectory evaluation and BDD's. IEEE Trans Comput-Aided Des Integr Circuits and Syst, 1995, 14: 413

[7] http: //www. SPARC. org

[8] http: //vlsi. colorado. edu/~ fabio/cudd

A Register-Oriented Modeling and Verification Method for Pipeline Microprocessor

He Hu and Sun Yihe

(*Institute of Microelectronics, Tsinghua University, Bejing* 100084, *China*)

Abstract: A new function verification method, register-oriented function verification method for pipeline microprocessor is presented. The main idea of the verification method is that verifying the function of each register to prove that the function of the whole design is correct. The pipeline microprocessors are almost made up of synchronous circuits. The current state of the synchronous circuits is decided by the state of registers. If the function of every register is correct, the function of the whole synchronous circuits is also correct. The state transfer of registers lies on the state of registers themselves and the primary input of the synchronous circuits. But in the final analysis, the state transfer of registers is decided by the primary input. This method can find an input data sequence to verify the function of all registers by comparing the response of the circuits under verifying and its reference model. The method is fit for debugging very much.

Key words: register-oriented; design verification; pipeline microprocessor; formal verification
EEACC: 1265F; 1265A　　**CCACC**: 6110F
Article ID: 0253-4177(2003)01-0098-06

He Hu　male, PhD candidate. His main areas of research are testing and verification of VLSI circuits.

Sun Yihe　male, was born in 1945, professor. His research field is the testability for digital VLSI and SOC, design technology for multimedia VLSI and for web network & data security VLSI and SOC.

Received 5 April 2002, revised manuscript received 14 August 2002