

面向 HDL 描述基于路径覆盖的 模拟矢量自动生成方法研究

李 噢¹⁾ 李思昆¹⁾ 郭 阳¹⁾ 万 海²⁾ 冷 彪¹⁾

¹⁾(国防科学技术大学计算机学院 长沙 410073)

²⁾(清华大学计算机科学与技术系 北京 100084)

摘要 提出和实现了一种面向 HDL 描述基于路径覆盖的模拟矢量自动生成方法。该方法在约束生成时只考虑控制语句的条件表达式,可有效避免生成冗余约束;利用扩展的决策图模型解决了中间信号到初始输入的传播问题和信号依赖关系问题,以及处理各种 HDL 描述风格的问题;采用约束逻辑编程方法解决了由位、位向量和整型变量组成的约束系统的统一处理问题。实验结果表明该方法能加快模拟矢量生成速度,提高路径覆盖率。生成的模拟矢量也能用于低层次设计验证和故障模拟,加快了设计进度。将该方法的原型系统用于一个 32 位微处理器核 RTL 级验证,发现了 RTL 级设计描述中的错误。

关键词 VLSI; 模拟矢量自动生成; 决策图模型; 路径覆盖; 约束逻辑求解

中图法分类号 TP338

A New Method for Automatic Generating Simulation Vectors from HDL Descriptions Based-on Path Coverage

LI Tun¹⁾ LI Si-Kun¹⁾ GUO Yang¹⁾ WAN Hai²⁾ LENG Biao¹⁾

¹⁾(School of Computer Science and Technology, National University of Defense Technology, Changsha 410073)

²⁾(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

Abstract A novel method for automatic generating simulation vectors from HDL descriptions based on path coverage and constraint solving is presented. The method only generates constraints for condition expression of the control statements, which can reduce the costs on constraint solving. It can deal with all constraints involving bits, bit-vectors and integers. It can deal with various HDL description styles, and various types of designs. Experimental results on several practical designs show that our method can efficiently improve the simulation vector generation process, which in turn accelerates the design process. The vectors generated by our method can also be used in low-level verification and fault simulation. The prototype system has been applied to verify RTL description of a real 32-bit microprocessor design and complex bugs remained hidden in the RTL descriptions are detected.

Keywords VLSI; automatic simulation vectors generation; DD model; path coverage; constraint solving

收稿日期: 2003-07-08; 修改稿收到日期: 2004-02-10. 本课题得到国家自然科学基金(60303011)、国家自然科学基金重点项目基金(90207019)和国家“八六三”高技术研究发展计划项目基金(2002AA1Z1480)资助。李 噢,男,1974 年生,博士,讲师,主要研究方向为并行模拟、微处理器设计验证技术、电子 CAD 技术等。E-mail:tunlee@sina.com. 李思昆,男,1941 年生,教授,博士生导师,主要研究方向为电子 CAD、VLSI 设计方法学、虚拟现实技术等。郭 阳,男,1971 年生,博士,副教授,主要研究方向为微处理器设计验证技术、电子 CAD 技术等。万 海,1981 年生,硕士研究生,主要研究方向为 CAD 技术。冷 彪,1982 年生,本科生,主要研究方向为电子 CAD 技术。

1 引言

VLSI 中设计的功能验证主要验证 HDL(Hardware Description Language) 描述的 RTL 级设计是否满足设计的功能描述的要求。主要的功能验证方法是形式化方法和模拟验证方法。目前,形式化方法还不能处理规模较大的设计。模拟验证仍然是功能验证的主要手段,模拟验证中的主要问题是设计生成所需的模拟矢量。近几年来,模拟矢量自动生成的研究成为模拟验证方法的研究重点。针对 RTL 级设计,各种模拟矢量自动生成方法采用不同的模型和算法进行模拟矢量生成。文献[1]采用扩展有限状态机模型进行模拟矢量生成;文献[2]利用遗传算法生成模拟矢量。文献[3, 4]利用二叉决策图模型生成模拟矢量,文献[5]利用赋值决策图模型进行模拟矢量生成。文献[6]基于设计描述的可控制性和可观察性,利用 SAT 求解方法进行模拟矢量生成,但是无法统一处理由位、位向量和整型变量组成的约束系统。这些方法主要使用了形式化验证技术进行模拟矢量自动生成,模型和算法处理的是设计的位级描述,因此能处理的设计规模有限。

HDL 描述也是一种软件描述,可以借鉴软件测试数据自动生成方法。文献[7]针对 VHDL 描述,首先枚举进程语句的路径,再将路径上的所有语句转化成约束表达式,最后利用约束求解系统 CLP(R) 求解生成的约束获得模拟矢量。文献[8]对该方法的改进是能根据软件测试中的 McCabe 测试标准提取每个进程语句的基本覆盖路径。文献[9]在商用符号模拟(symbolic simulation)系统上运行 Verilog 描述的设计,将执行轨迹转化为 CLP 描述,再进行约束求解得到模拟矢量。文献[10]利用约束求解为 SystemC 描述生成模拟矢量,该方法主要针对 SystemC 描述的 FSM,以状态覆盖率和状态变迁覆盖率为测度生成模拟矢量。

但是上述方法还存在不足。首先,对路径中的所有语句都进行约束生成,将产生大量的冗余约束,影响约束求解效率。其次,已有方法生成的模拟矢量只是各进程语句的输入信号,现有方法都没有讨论如何获得初始输入的模拟矢量的问题。第三,文献[9]的方法采用的符号模拟方法能处理的设计规模有限,并且无法判断生成的模拟矢量对语句、分支和路径等覆盖率测度的效果。第四,文献[10]针对 SystemC 描述生成模拟矢量只能处理特定书写格式的

FSM 描述,方法能处理的设计类型有限。

针对现有方法的不足,我们提出了一种新的面向 HDL 描述基于路径覆盖率的模拟矢量自动生成方法。该方法利用决策图模型解决了如何将生成的中间变量和信号的模拟矢量传播到初始输入的问题;约束生成时只考虑控制语句的约束生成,而不需要为所有的语句生成约束,提高了约束求解效率;能统一处理由位、位向量和整型变量构成的约束系统;能处理各种描述风格和各种控制和数据通路组成的设计。实验结果表明该方法能加快模拟矢量生成速度,提高路径覆盖率。生成的模拟矢量也能用于低层次设计验证和故障模拟,加快了设计进度。本文实现的原型系统主要针对 Verilog 语言,后续讨论提到的 HDL 指的是 Verilog 语言。

2 Verilog 描述建模

对设计描述的控制部分,我们用控制流图(Control Flow Graph, CFG) 进行建模。控制流图是由结点和边组成的有向图,结点表示 HDL 程序中的语句(赋值语句和控制语句),有向边表示结点间程序执行的顺序。设计的组合部分通常就是一个数据通路,起到传播信号值的作用,我们采用决策图(Decision Diagram, DD) 模型^[11] 来表示信号之间的依赖关系,将每个模拟周期求得的中间信号的值传播到初始输入,形成每个模拟周期应输入的模拟矢量。文献[11]中定义的 DD 模型不能满足当两个控制语句间有对某变量赋值的描述,我们扩展了现有 DD 模型的定义,解决了对各种描述风格的描述问题。

定义 1. 扩展的 DD 模型是一个完全二叉树,在二叉树中,只有两种类型的结点:控制结点和赋值结点。控制结点和赋值结点交替出现。控制结点用于控制程序(数据流)的走向,而赋值结点用于描述赋值信息。

图 1 是扩展的 DD 模型的示意图,图中椭圆结点表示 HDL 描述中的控制语句,矩形结点表示赋

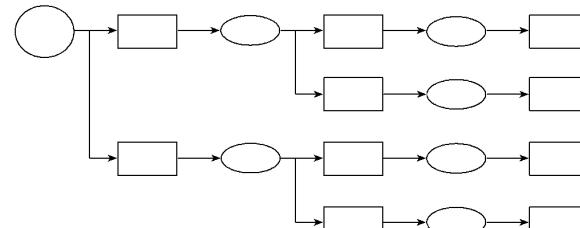


图 1 扩展 DD 模型示意

值语句,根结点是 DD 模型表示的变量。图 2 是求最大公约数(GCD)设计的 Verilog RTL 描述,图 3 给

出了对应的 CFG 和 DD 模型。该示例将用于后续算法的介绍。

```

Module GCD (Clock, Reset, Load, A, B, Done, Y);
parameter Width=8;
input Clock, Reset, Load;
input [Width-1:0] A, B;
output Done;
output [Width-1:0] Y;
reg A_lessthan_B, Done;
reg [Width-1:0] A_New, A_Hold, B_Hold, Y;

always@(posedge Clock)
begin
    if(Reset)           1
    begin
        A_Hold=0;      2
        B_Hold=0;      3
    end
    else if(Load)       4
    begin
        A_Hold=A;      5
        B_Hold=B;      6
    end
    else if(A_lessthan_B) 7
    begin
        A_Hold=B_Hold; 8
        B_Hold=A_New;  9
    end
    else
        A_Hold=A_New; 10
end

always@(A_Hold or B_Hold)
begin
    if(A_Hold>=B_Hold) 1
    begin
        A_lessthan_B=0; 2
        A_New=A_Hold-B_Hold; 3
    end
    else
        begin
            A_lessthan_B=1; 4
            A_New=A_Hold; 5
        end
    end

always@(A_Hold or B_Hold)
begin
    if(B_Hold==0) 1
    begin
        Done=1;          2
        Y=A_Hold;        3
    end
    else
        begin
            Done=0;          4
            Y=0;              5
        end
    end
endmodule

```

图 2 GCD 的 Verilog 描述

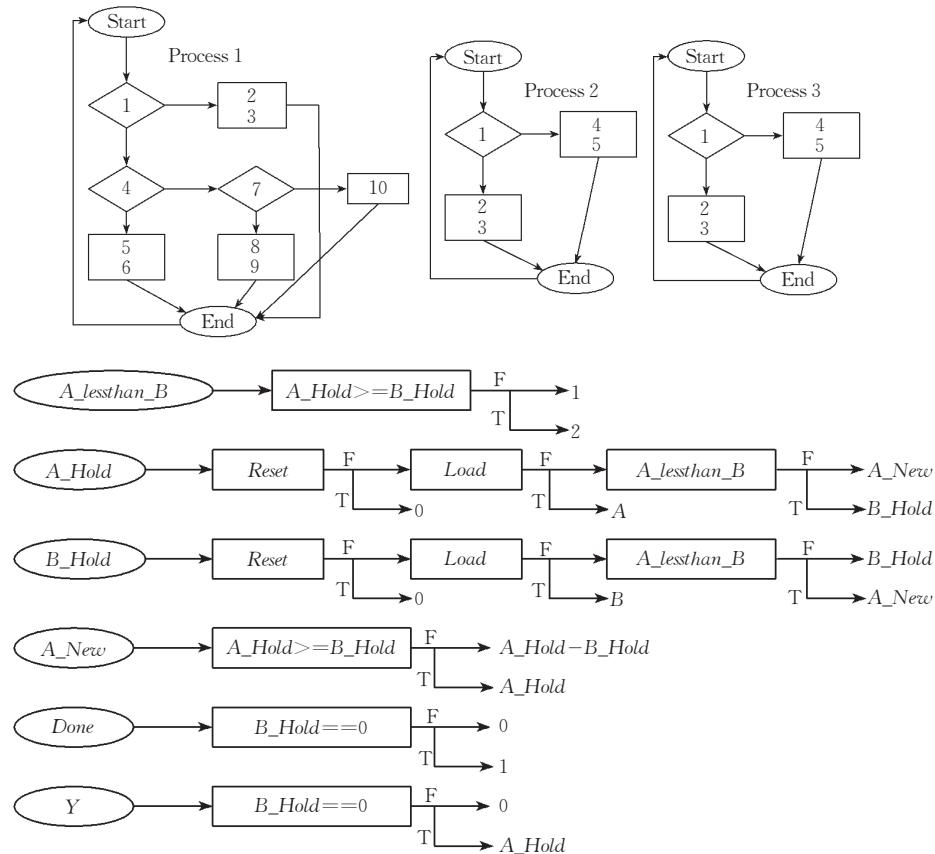


图 3 GCD 描述的 CFG 和 DD 模型

3 基于路径的模拟矢量生成方法

3.1 数据获取

对待验证的 Verilog 描述,首先由 Verilog 编译器分析设计描述,获得所需的设计数据,包括五个列表:

(1)端口列表:设计的所有输入端口的列表,包括输入端口的宽度等信息.

(2)信号列表:所有中间信号和输出信号的列表.

(3)表达式列表:包括条件表达式和赋值表达式.赋值表达式是赋值语句右边部分的表达式.

(4)CFG 列表:设计描述中每个进程语句的 CFG 的列表.每个 CFG 结点保存该结点对应的语句的表达式在表达式列表中的编号,以及该结点定义的变量(出现在赋值符号左边的变量).

(5)DD 模型列表:所有中间变量和输出信号的 DD 模型的列表.

3.2 路径枚举和排序

待测路径集合的生成过程是通过深度优先的图遍历算法实现的.该算法从进程语句 CFG 的 START 结点开始,每碰到一个结点,如果该结点为顺序结点,直接将该结点压栈;如果为控制结点,则记录下该控制结点在 CFG 中的后续结点数,将该结点压栈,按结点序号次序取后续结点作为新结点进行处理.该过程持续到碰到 END 结点,表示找到一条路径.此后,算法将从栈中弹出结点作为新结点,如果新结点是控制结点,并且有未处理的后续结点,则顺序选择后续结点重复上述运算.该过程将一直进行下去,直到栈为空.此时找到的所有路径就是该进程语句路径覆盖测度的依据.算法复杂度为 $O(n^2)$,其中 n 为进程语句 CFG 的结点数.以图 3 的 CFG 为例,进程语句 1 的所有路径为:(1)1—2—3;(2)1—4—5—6;(3)1—4—7—8—9;(4)1—4—7—10.

由于 HDL 描述是多个进程语句的集合,各进程间存在依赖关系,导致所选择的路径间也存在着依赖关系.根据 HDL 模拟语义,这些存在依赖关系的路径必须按照依赖关系顺序执行.我们定义路径间的依赖关系如下.

定义 2. 对路径 p_1 和 p_2 ,如果 p_2 中某个控制结点的控制表达式使用的信号变量在路径 p_1 的某个结点上被赋值,则称路径 p_2 依赖于路径 p_1 , p_1 为 p_2 的前继路径.

根据路径依赖定义,每个待测路径都有一个前继路径编号,只有前继路径执行完后,该路径才能被

执行.不存在依赖关系的路径间的执行顺序是任意的.为了解决路径间循环依赖的问题,每个模拟周期中,来自时钟信号驱动的进程语句(posedge 和 negedge)的路径在排序时排在最前面,是每个模拟周期中最先执行的路径.实际模拟验证中,对存在循环依赖关系的路径组合,模拟激励的加载是通过随机选择一个进程语句进行赋值实现的.因此,为循环依赖路径组合生成约束时,将随机选择一条路径作为最先执行的路径,这与实际模拟验证是一致的,不会违反模拟语义.

以图 3 的 CFG 为例,假设某个模拟周期各进程语句要执行的路径分别为 $P_1:1—4—7—8—9$; $P_2:1—2—3$; $P_3:1—2—3$.字母 P 后的编号表示该路径来自于哪个进程语句.由于 P_2 中的控制语句 1 使用的变量 A_Hold 和 B_Hold 在 P_1 中的语句 8 和 9 上被赋值,因此, P_1 将在 P_2 前执行,而在 $A_lessthan_B$ 信号上,路径 P_1 依赖于 P_2 .按照对时钟驱动进程的特殊处理,解决了 P_1 和 P_2 间的循环依赖关系.最后的路径执行顺序为 $P_1 \rightarrow (P_2, P_3)$.

3.3 路径分析和约束生成

读入自动生成的或用户选择生成的待测路径组合文件,首先为条件语句的条件表达式生成约束,再将条件表达式用到的中间信号变量传播到初始输入上,以避免冗余约束.本节使用的待测路径示例是 3.2 节的路径 P_1 , P_2 和 P_3 .

3.3.1 时间化身和空间化身

HDL 描述中用到的信号变量带有空间信息和时间信息,空间信息表示的是该信号变量在描述中不同位置的出现;而时间信息表示的是该变量在不同模拟时刻的出现.约束生成时,用空间化身编号(spatial number)和时间化身编号(temporal number)来表示信号变量的这种特性.信号变量在被声明和初始化时有一个初始空间化身编号,每次该变量出现在赋值语句的左边时,空间化身编号在原空间化身编号基础上加 1.在每一个模拟周期,变量的时间化身编号应是上一模拟时刻时间化身编号加 1.用 $Signal_name^{temporal_number}_{spatial_number}$ 来表示信号变量在生成的约束中的出现.例如,假设当前模拟时刻为 t , a 已经被赋值 m 次,则为表达式 " $a = a + 1$ " 生成的约束形式为 $a_m^t = a_{m+1}^t + 1$.

3.3.2 路径分析和条件表达式约束生成

激活所选路径的基本前提是其进程语句必须被激活,因此,为路径生成的约束所满足的条件是:待测路径使用的信号变量集合中任一信号当前模拟时

刻的值不等于上一模拟时刻的值。路径中使用的信号变量集合是路径中所有条件语句表达式使用的变量和出现在路径中赋值语句右边的变量。对路径中的条件语句,与该路径所属的进程语句的 CFG 一起,分析条件语句的跳转方向,判断条件语句的条件求解取值。例如,对路径 P2,其条件语句编号为 1,后续语句编号为 2,在进程语句 2 的 CFG 上可知语句 1 的条件求解结果为 True。

以路径 P2 为例,假设当前模拟时刻为 t , A_Hold 上一模拟周期的空间化身编号为 m , B_Hold 上一模拟周期的空间化身编号为 n 。则生成的约束为

$$\begin{aligned} & (A_Hold_m^{t-1} \neq A_Hold_0^t) \vee \\ & (B_Hold_n^{t-1} \neq B_Hold_0^t) \end{aligned} \quad (1)$$

$$A_Hold_0^t \geq B_Hold_0^t$$

3.3.3 信号传播约束生成

为待测路径的控制语句的表达式生成约束时,其中使用的信号变量有些是初始输入信号,例如对待测路径 P1,生成控制语句的表达式约束方程为

$$\begin{aligned} & (Reset_0^t == 0) \wedge (Load_0^t == 0) \wedge \\ & (A_lessthan_B_1^{t-1} == 1) \end{aligned} \quad (2)$$

其中使用的信号变量 $Reset$ 和 $Load$ 都为初始输入,求解该约束方程可以直接得到模拟矢量,而 $A_less-than_B$ 则不是初始输入,需要用传播算法将其值传播到初始输入。

传播算法利用中间信号变量对应的 DD 模型进行。传播类型有两种:前向传播和反向传播。前向传播从信号变量 DD 模型的根开始,解析碰到的条件表达式,根据解析结果决定搜索方向,直到碰到一个对该变量赋值的终端结点。反向传播是指在知道变量取值时,从 DD 模型的终端结点开始向根结点方向搜索。为了到达根结点,搜索时确定所碰到的条件表达式的求值结果。在 DD 模型中条件表达式中使用的信号变量也是带有时间化身编号和空间化身编号的。

以约束方程(1)和(2)为例。对方程(2)中的信号 $A_lessthan_B$,在模拟时刻 t 取值为 1,反向搜索对应的 DD 模型,可以得到约束方程 $A_Hold_m^{t-1} \geq B_Hold_n^{t-1}$ 。对方程(1)中模拟时刻 t 的 A_Hold 和 B_Hold 变量,前向搜索对应的 DD 模型直到碰到赋值结点。假设 $t-1$ 时刻 $Reset=1$,遍历 A_Hold 的 DD 模型,可以立即得到 $A_Hold_0^t=0$,同样可以得到 $B_Hold_0^t=0$ 。

为了正确地搜索信号变量的 DD 模型,需要在碰到 DD 模型中的条件表达式时能选取信号变量正

确的空间化身编号和时间化身编号,编号的选取原则是

1. 对条件表达式中的初始输入信号变量,其时间化身编号为当前模拟时刻编号,空间化身总是为 0。这是因为在设计中不会为初始输入赋值,其值一旦在每个模拟周期被确定后,在该模拟周期永远不会改变。

2. 对表达式中使用的中间信号变量,如果其赋值语句为阻塞赋值,则其时间化身为当前模拟时刻编号,空间化身为当前模拟时刻的最新值。

3. 如果表达式中使用的中间信号变量的赋值语句为非阻塞赋值,则其时间化身为上一模拟时刻编号,空间化身为上一模拟时刻的最新值。

3.3.4 约束生成方法

HDL 描述中的信号变量包括位、位向量和整型。需要为不同类型的信号变量采用不同的约束生成方法。

1. 位。位类型的信号变量的运算比较简单,GProlog^[12]提供了各种运算操作符支持位变量的运算,并且只会为位变量构成的表达式生成一个约束方程,位变量的取值范围为 {0, 1}。

2. 位向量。对位向量将分为几种情况进行考虑:

(1) 整体。当位向量变量作为一个整体参加运算时,如果表达式中还有整型变量,则将位向量变量看成整型变量参加运算。而如果参加运算的变量都为位向量,则需要对位向量进行分解。例如对两个 4 位的位向量 V1 和 V2 的比较运算“ $V1 == V2$ ”,生成的约束如式(3)所示。此处省略了变量的空间化身编号和时间化身编号,约束方程中 V1 和 V2 表示位向量对应的整数值,V1 和 V2 后的编号表示位向量的第几位参加。

$$\begin{aligned} & V1_0 = V2_0 \\ & V1_1 = V2_1 \\ & V1_2 = V2_2 \\ & V1_3 = V2_3 \end{aligned} \quad (3)$$

$$2^3 * V1_3 + 2^2 * V1_2 + 2 * V1_1 + V1_0 = V1$$

$$2^3 * V2_3 + 2^2 * V2_2 + 2 * V2_1 + V2_0 = V2$$

(2) 位选。位选是指位向量的某一位或某几位参加表达式运算,此时除了要为参加运算的位生成约束方程,还将为对应的位向量生成约束方程,对参与运算的位进行约束。例如考虑表达式 “ $V[1:0] == 2'b00$ ”, V 为 4 位的位向量,则生成的约束方程为

$$V_1 = 0,$$

$$V_0 = 0,$$

$$2^3 * V_3 + 2^2 * V_2 + 2 * V_1 + V_0 = V.$$

对表达式“ $V[0]==0$ ”，生成的约束方程为

$$V_0 = 0,$$

$$2^3 * V_3 + 2^2 * V_2 + 2 * V_1 + V_0 = V.$$

(3) 整型. GProlog 为整型变量提供了丰富的操作符支持,同样的,对整型变量也要定义其取值范围.

3.4 约束求解

生成约束文件时,以 HDL 描述的初始输入作为约束求解的输入,以待测路径条件语句的条件表达式求解结果作为输出,以使约束输出满足路径运行要求为目标,求解约束方程. 如果无解,则说明待测路径组合在实际运行时不会被执行. 如果有解,则求解器会返回一个结果. 对得到的约束求解结果,按照结果的空间化身编号,将结果组织成能用于 Verilog 模拟器的格式,形成最终的测试输入. 在生成的约束文件中定义了一个 findall 操作,能返回满足约束的所有输入,这是为了用尽可能多的模拟矢量执行待测路径,提高故障覆盖率.

4 实现和实验

4.1 原型实现

我们已经实现了面向 HDL 基于路径覆盖的模拟矢量自动生成原型系统,该系统包括以下几个部分:

(1) Verilog 编译器. 分析读入的 Verilog 描述,生成 CFG、DD 模型、端口列表、表达式列表和变量列表.

(2) 路径枚举器. 从 Verilog 编译器获得各进程语句的 CFG 结构,枚举各进程语句的所有可能执行路径.

(3) 约束生成器. 从 Verilog 编译器得到 CFG、DD 模型等数据,分析待测路径组合,生成约束文件.

(4) 约束求解和输入生成. 用 GProlog 求解约束文件,分析输出结果,判断有解还是无解. 有解时将输出组织成最终的 Testbench.

4.2 实验结果

在我们实现的原型系统上,用几个实际设计进行了实验. 实验平台配置为 AMD Athlon XP 1.8G, 256MB 内存, Windows 2000 操作系统. 实验电路的属性如表 1 所示. 表中 Line 表示设计的 Verilog 描述行数, PI 为初始输入数, Sig 为中间信号和最终输出数之和, Pr 为设计描述中的进程数.

表 1 实验电路属性

名称	线(Line)	PI	Sig	Pr
8 位加法器	46	8	9	2
交通灯控制器	71	3	7	1
8051 控制器	360	18	59	2
Viper 微处理器核	400	24	57	2

表 2 为实验结果. 表中 τ 表示模拟周期数, T. P 表示理论可行路径组合数, F. P 表示在模拟矢量生成中实际可行的路径组合数, Time 表示模拟矢量生成时间, 包括 Verilog 编译、约束生成和约束求解的时间, Random 为用随机模拟矢量生成方法进行比较时, 随机生成的模拟矢量数, Cov. P 为模拟随机生成模拟矢量时, 所覆盖的路径组合数. 从实验结果可知, 虽然随机模拟矢量生成的模拟矢量数远远多于我们基于路径覆盖生成的模拟矢量, 但是覆盖率效果却较差. F. Size 为按照本文方法生成的约束文件大小, “[7]F. Size” 为按照参考文献 [7], 将路径上所有语句都转化为约束方程后得到的约束文件大小, 可以看出本文方法在保持路径覆盖率高的同时, 能有效地避免产生冗余约束, 提高约束求解效率.

表 2 实验结果

名称	τ	T. P	F. P	Time	Random	Cov. P	F. Size	[7]F. Size
8 位加法器	4	14	4	8s	100	4	40KB	98KB
交通灯控制器	4	21	21	39s	100	21	57KB	101KB
8051 控制器	9	21190	12474	1h20m	40000	7901	41.3MB	85.9MB
Viper 微处理器核	7	47800	7106	2h10m	40000	5927	70.9MB	103.5MB

在进行约束求解时,对同一个路径覆盖要求, GProlog 求解器可以生成尽可能多的模拟矢量,这些模拟矢量能在执行待测路径的同时,尽可能地用信号的各种可能值对设计进行模拟验证,提高信号跳变覆盖率.

4.3 实际应用

我们将该原型系统用于自行设计的一款 32 位

微处理器的流水线的模拟验证. 对流水线各级的 RTL 描述, 枚举了描述中进程语句的所有可执行路径, 对自动生成的路径组合生成模拟矢量进行模拟验证. 下面以译码器模块的验证为例进行说明: 在为译码器生成模拟矢量时, 发现了译码处理中断异常信号的一个错误.

微处理器核中断分为 6 种, 优先级从高到低分

为 *Reset*, *Data Abort*, *FIQ*, *IRQ*, *Prefetch Abort* 和软件中断。译码器在处理中断信号时,需要根据当前处理器状态和中断信号,判断是否能执行中断以及执行哪一个中断异常。译码器 RTL 描述是一个 2000 多行的模块,与中断异常处理相关的代码如下所示。

```
assignCanProcessInterrupt=
  (in_TrueFiq | in_TrueIrq | in_MatchBreakPoint)
  & ~ExistState;
always @(...)
if(CanProcessInterrupt) //1
begin
...
end
else if(in_IAbt) //2
begin
...
end
```

其中 *ExistState* 信号为当前处理器状态,由其控制是否能处理中断异常。*In_TrueFiq*,*in_TrueIrq*,*in_Iabt* 分别为 *FIQ*, *IRQ* 和 *ABORT* 中断信号。根据体系结构描述,当能处理中断信号时,如果 *in_TrueIrq* 和 *in_Iabt* 信号同时为“1”时,则语句块 1 被执行,语句块 2 不会被执行;而不能处理中断时,两个语句块都不能被执行。在生成路径组合时,我们得到一条这样的路径组合:当前时刻执行语句块 1,下一周期执行语句块 2。将上述路径组合的判断语句条件转化为 GProlog 程序,编译执行后,约束求解结果中包含一些使译码器不能处理中断,并且 *in_TrueIrq* 和 *In_Iabt* 同时为高,而执行了语句块 2 的模拟矢量。将模拟矢量与译码器 RTL 描述在 Verilog 模拟器上运行后,发现该模拟矢量的模拟输出违反了上述体系结构描述。经过检查代码,发现对 *in_Iabt* 信号的判断没有进行是否能处理中断的检测。

5 结束语

在现有方法和系统基础上,今后将主要研究如何指导生成各种所需的模拟矢量,例如,为某些信号生成取边界值的输入;生成使某些算术操作溢出或产生异常的模拟矢量等。另一个工作是现有系统的实用化。

参 考 文 献

- Cheng K., Krishnakumar A. S.. Automatic generation of functional vectors using extended finite state machine model. ACM Transactions on Design Automation of Electronic Systems, 1996, 1(1): 57~79
- Ferrandi F., Sciuto D., Fin A., Fummi F.. An application of genetic algorithms and BDDs to functional testing. In: Proceedings of IEEE International Conference on Computer Design, Austin, Texas, 2000, 48~58
- Ganai M. K., Aziz A., Kuehlmann A.. Enhancing simulation with BDDs and ATPG. In: Proceedings of the 36th Design Automation Conference, New Orleans, US, 1999, 385~390
- Li Tun, Guo Yang, Li Si-Kun. Automatic simulation vector generation using interacting FSM model. Journal of Software, 2003, 14(3): 628~634(in Chinese)
(李 峰, 郭 阳, 李思昆. 交互状态机模型模拟矢量自动生成方法. 软件学报, 2003, 14(3): 628~634)
- Ghosh I., Fujita M.. Automatic test pattern generation for functional RTL circuits using Assignment Decision Diagrams. In: Proceedings of the 37th Design Automation Conference, Los Angeles, US, 2000, 43~48
- Farzan Fallah, Srinivas Devadas, Kurt Keutzer. Functional vector generation for HDL models using linear programming and Boolean satisfiability. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2001, 20(8): 994~1002
- Venuri R., Kalyanaraman R.. Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming. IEEE Transactions on Very Large Scale Integration Systems, 1995, 3(2): 201~214
- Pauli C., Nivet M. L., Santucci J. S.. Use of constraint solving in order to generate test vectors for behavioral validation. In: Proceedings of IEEE International High-Level Validation and Test Workshop, Berkeley, US, 2000, 15~20
- Zeng Zhi-Hong, Maciej Ciesielski, Bruno Rouzeyre. Functional test generation using constraint logic programming. In: Proceedings of IFIP TC10/WG10.5 Eleventh International Conference on Very Large Scale Integration of Systems-on/Chip, Montpellier, France, 2001, 133~138
- Ferrandi F., Rendine F., Sciuto F.. Functional verification for System C descriptions using constraint solving. In: Proceedings of the Design, Automation, and Test in Europe, Paris, France, 2002, 744~751
- Ubar R.. Test synthesis with alternative graphs. IEEE Design & Test of Computers, 1996, 13(1): 48~57
- Daniel Diaz, Philippe Codognet. The GNU prolog system and its implementation. In: Proceedings of the ACM symposium on Applied Computing, Como, Italy, 2000, 728~732



LI Tun, born in 1974, received the B. E. degree in 1996, M. S. degree in 1999 and Ph. D. degree in 2003 from National University of Defense Technology. His research interests include parallel logic simulation, microprocessor design verification.

LI Si-Kun, born in 1941, professor and Ph. D. supervisor of Department of Computer Science and Technology of National University of Defense Technology. His research interests include microprocessor design verification, virtual reality, and VLSI design methodology, etc.

GUO Yang, born in 1971, received the B. E. degree in 1993, M. S. degree in 1996 and Ph. D. degree in 1999 from

National University of Defense Technology. He is an Assistant professor of Department of Computer Science and Technology of National University of Defense Technology. His areas of research interests include microprocessor design verification.

WAN Hai, born in 1981, received the B. E. degree in 2003 from National University of Defense Technology. Now he is working toward the M. S. degree at the Tsinghua University. His research interests focus on computer-aided design.

LENG Biao, born in 1982. He is now working toward the B. E. degree at the National University of Defense Technology. His research interests focus on computer-aided design.

Background

Functional verification is one of the most important techniques used to assure the correctness of VLSI design. The coverage-driven approach, which combines the ideas of simulation and formal verification, is proposed and rapidly getting popular. With the coverage reports, the verification engineers can focus their efforts on the un-verified areas and generate more test vectors by the help of formal techniques to achieve better functional coverage. Among the techniques used in coverage-driven verification framework, the automatic functional vectors generation technique is the most important one.

The project “Research on Techniques of VLSI RT-Level Automatic Functional Vectors Generation” is supported by the National Natural Science Foundation of China under Grant No. 60303011, 90207019; and the National High Technology Development 863 Program of China under Grant Nos. 2002AA1Z1480. In this project, our research targets at the key techniques of the automatic simulation vectors generation method, and building a coverage-driven verification prototype system. The work of this paper deals with the automatic simulation vectors generation techniques based on path-

cover of HDL description in the coverage-driven verification framework.

Other works of the project include:

1. Program slicing based circuit extraction method, and a paper describes this work has been accepted to be appear in Journal of Computer Science & Technology..
2. Automatic simulation vectors generation based on interacting FSM, the work has been published in Journal of Software in 2003.
3. A novel VCD file based coverage analysis method is proposed and to be appeared in Journal of Computer Research & Development.
4. A novel assertion based automatic simulation vectors generation method is also proposed and is accepted to be appeared in Journal of Software.
5. The related works are also published in the 12th Asian Test Symposium, 17th International Conference on VLSI Design, Proceedings of 2004 Asia South Pacific Design Automation Conference, and the 14th ACM Great Lakes Symposium on VLSI.