基于符号模拟和约束逻辑编程的 RTL 级 Verilog 谓词抽象方法

李 暾 屈婉霞 郭 阳 刘功杰 李思昆
(国防科学技术大学计算机学院 长沙 410073)

摘 要 利用人工智能最新研究成果——约束逻辑编程对 Verilog 描述进行谓词抽象,并与目前基于 SAT 的方法进行了比较.首先通过符号模拟建立 Verilog 的形式化模型,然后结合要抽象的谓词,将谓词抽象问题转化为约束逻辑编程问题并进行求解.该方法的优点是在计算抽象系统时,不需要像基于 SAT 的方法那样将字级约束打散成位级约束,求解效率显著提高;提供了一个统一的框架用于描述各种约束.实验结果表明,与基于 SAT 的抽象技术相比,基于约束逻辑编程的抽象方法的求解速度有显著提高.

关键词 谓词抽象;Verilog 约束逻辑编程;模型检验;符号模拟
中图法分类号 TP306

Predicate Abstraction of RT-Level Verilog Using Symbolic Simulation and Constraint Logic Programming

LI Tun QU Wan-Xia GUO Yang LIU Gong-Jie LI Si-Kun
(*School of Computer Science & Technology, National University of Defense Technology, Changsha* 410073)

Abstract Abstraction is one of the most effective ways to address state explosion problem in model checking, and predicate abstraction has been applied successfully to large software and now to hardware descriptions, such as Verilog. This paper evaluates the state-of-the-art AI techniques, constraint logic programming (CLP), to improve the performance of predicate abstraction of circuits, and compared it with the SAT-based predicate abstraction techniques. With CLP-based techniques, we can model various constraints in a unified framework; we can also model the word-level constraints without flattening them into bit-level ones as SAT-based method dose. Experimental results have showed the promising improvements on the performance of predicate abstraction of hardware designs.

Keywords predicate abstraction; Verilog constraint logic programming; model checking; symbolic simulation

1 引 言

模型检验技术[1]已经在软硬件设计领域得到了广泛的应用.但是,随着系统规模的不断扩大和并发性的引入,模型的可达状态数随着系统组件数的增多而快速增加,模型检验技术的进一步发展受到状态空间爆炸问题的制约.因此,如何对设计进行正确

收稿日期:2005-05-30;修改稿收到日期:2007-01-14.本课题得到国家自然科学基金(60403048,60573173)资助.**李 暾**,1974 年生,博士,副教授,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: tunli@nudt.edu.cn.**屈婉霞**,女,1972 年生,博士研究生,讲师,主要研究方向为计算机体系结构、VLSI 验证技术等.**郭 阳**,男,1971 年生,副研究员,主要研究方向为微处理器设计验证技术.**刘功杰**,男,1977 年生,硕士,主要研究方向为电子 CAD 技术.**李思昆**,男,1941 年生,教授,博士生导师,主要研究领域为 VLSI 设计方法学、虚拟现实技术等.

的化简以减少状态空间需求的抽象技术，是决定模型检验方法能否应用于验证大系统的关键之一，成为当前模型检验领域的研究热点. 抽象的基本思想是将原始模型的多个状态进行分组，映射到一个抽象的、更小的状态集合，同时保证系统的相关行为. 目前，在软件验证领域中，成功用于大规模系统验证的抽象技术是谓词抽象(predicate abstraction)技术[2]. 在硬件验证领域，最常用的局部化简技术(localization reduction)是谓词抽象的特例.

定理证明器[3-4]是最早用于谓词抽象的求解器，代表性研究有文献[2,5-7]等中介绍的工作，开发的SLAM[8]、BLAST[9]和 Magic[10]等主要用于软件验证. 基于 SAT 的谓词抽象方法首先应用于硬件验证[11]，此后，Clarke 等[12]提出了用 SAT 进行谓词抽象的方法，并将该方法用于 ANSI-C 的验证. 此方法的主要思想是为设计代码中每一个基本块中的每一个谓词设置两个符号变量，分别表示基本块执行前后的状态，用谓词作为自由变量产生求解公式，使用位向量逻辑将求解公式转换成合取范式 CNF，然后使用 SAT 求解器，获得符号变量的所有可满足赋值. Clarke[13]等将该方法应用于 RTL 级 Verilog 设计的抽象和验证. 文献[14]将该方法用于 SpecC 的验证，显示了谓词抽象方法在硬件验证领域的广阔前景.

基于定理证明器和 SAT 的谓词抽象方法存在很多问题. 首先，利用定理证明器进行抽象需要成千上万次地调用定理证明器进行求解，并且目前的定理证明器对很多数据类型和运算不提供支持，如位、位向量、除法运算等. 其次，定理证明器使用无界整数对变量进行建模，对 Overflow 或者 bit-wise 操作符并没有建模，而硬件描述语言，如 Verilog 则提供了扩展的 bit-wise 操作符集合，因此限制了待验证系统的代码实现方式. 第三，基于 SAT 的方法虽然能支持各种数据类型和运算，但需要将字级(word-level)谓词综合到位级(bit-level)，使得字级求解的问题复杂度提高，求解时间随设计规模增大成指数级增长.

针对现有谓词抽象技术的不足和缺点，本文提出了一种新的基于逻辑约束编程(Constraint Logic Programming，CLP)[15]的 Verilog 字级谓词抽象方法以及 Verilog 模型到 CLP 约束的转换方法. 该方法首先利用 DD(Decision Diagrams)模型[16]建立 Verilog 描述的形式化模型，然后利用符号模拟方法建立设计的状态机模型，并将用于抽象的谓词转化为 CLP 约束进行求解. 其优点是使用统一的约束描述模型，能统一位、位向量和限界整数等数据类型以及各种逻辑和算术运算；无需将字级约束展开为位级约束，使得约束求解的效率更高，并且能保证抽象的精确性.

2　Verilog 建模

本文方法所支持的 RTL Verilog 设计假设：具有单个时钟 *clk* 的同步 Verilog，*clk* 只使用 *posedge* 或者 *negedge*，两者不能同时出现. 另外，对每个信号的赋值只能在一个地方进行. 对 RTL Verilog 语法的支持是 Verilog 可综合子集.

下面首先介绍 Verilog 建模的基础——DD 模型.

定义 1.　DD 模型是一个有向无圈图 $G=(M,\Gamma,x)$，其中 M 是结点集合，Γ 表示结点间的关系，$\Gamma(m)\subset M$ 表示结点 m 的所有后继结点，结点 $m\in M$ 有一个标记符号 $x(m)$，标记符号可以是变量、代数表达式或常量. 对非终端结点 m 有 $\Gamma(m)\neq\varnothing$，有一个从 $x(m)$ 到 m 的后继结点 $m^e\in\Gamma(m)$ 的映射函数. 用 m^e 表示当 $x(m)=e$ 时 m 可到达的后继结点.

定义 2.　连接结点 m 及其后继结点 m^e 的边 (m,m^e) 称为激活的，当且仅当存在一个赋值 $x(m)=e$. 连接结点 m_i 和 m_j 的所有被激活边组成一条激活路径 $l(m_i,m_j)$. 从初始结点 m^0 到终端结点 m^t 的激活路径 $l(m^0,m^t)$ 称为完全激活路径.

定义 3.　DD 模型 $G_y=(M,\Gamma,x)$ 能用于表示一个函数 $y=f(x)$，当且仅当对每一个输入 x，存在一条到终端结点 m^t 的完全激活路径，并且 $x(m^t)=y$ 成立.

定义 4.　DD 模型的集合 $G(S)=\{G_y\}$ 可以用于表示一个数字系统 $S=(F,N)$，如果对 F 中的每一个函数 $y=f(x)$，都存在一个 DD 模型 $G_y=(M,\Gamma,x)$，则 $G(S)=\{G_y\}$ 称为数字系统 S 的 DD 模型.

根据上述定义，Verilog 设计中的每一个变量(信号)和最终输出对应一个 DD 模型，DD 模型的根结点是该信号或变量，终端结点是所有对该变量或信号的赋值语句. 非终端结点是包围这些赋值语句的控制语句，包括 if、case 和 loop 等语句. 为了便于处理，本文方法对 Verilog 描述中的 case 语句进行预处理，将其转换为一系列“if…else”语句.

图 1 给出了一个 Verilog 例子[13]及其 DD 模型的示例，其中椭圆表示变量，矩形表示判断条件. 该示例将用于本文后面的介绍. DD 模型的提取算法请参考文献[17].

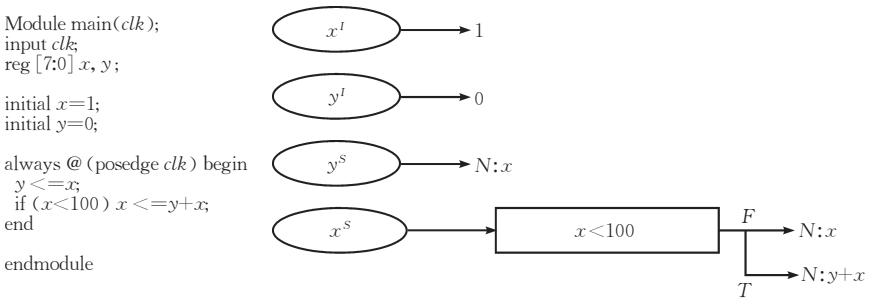

图 1 Verilog 示例及其 DD 模型

Verilog 语言的各种语法采用不同方法进行建模.

① 连续赋值

Verilog 中的每个 wire 类型变量只能在连续赋值语句中被赋值,令 w_i 表示被第 i 个连续赋值语句赋值的变量,e_i 表示赋值语句右边的表达式,用 A 表示设计中所有连续赋值语句的语义,则有 $A := \bigwedge_i w_i = e_i$. 这个形式化语义可以通过搜索表示 w_i 的 DD 模型得到.

② initial 和 always

由时钟信号触发的 always 模块以及 initial 模块,其中的语句或者定义状态的初始值和状态迁移函数,或者定义的是组合电路. 这种不同可以通过检查 always 语句的事件触发条件来判断. 由时钟触发的 always 语句中的变量赋值用一个 latch 来表示. 我们通过在 DD 模型上进行标识来区分组合与时序描述,如图 1 中,上标"I"表示该赋值是在 initial 语句中执行的,而上标"S"表示该赋值是在时序模块中执行的. 类似地,标示"C"表示赋值是在组合模块中执行的.

在模型检验中,通常用有限状态机表示原始设计,本文亦采用此方法,将 Verilog 设计建模为一个有限状态迁移系统 $T=(S,I,R)$,其中 S 为状态集合,I 为初始状态,R 为迁移关系. 令 V 表示给定 Verilog 描述中的变量集合. $L \subseteq R$ 表示真正 latch 的集合. 状态机的状态集 S 定义如下:$S := \{0,1\}^{|L|}$. 对 S 中的状态 s,用 $s(e)$ 表示特定状态中表达式 e 的值,而组合逻辑中的变量记为 $C := V \backslash L$.

用进程状态表示 initial 和 always 中语句的语义. 一个进程状态 φ 将变量 V 中的变量 r 映射到一个表达式对 $(\varphi_c(r), \varphi_f(r))$,其中,$\varphi_c(r)$ 表示 r 当前值,$\varphi_f(r)$ 表示 r 的最终值. 表达式对可以区分非阻塞(non-blocking)赋值和阻塞(blocking)赋值. 非阻塞赋值只更新最终值 $\varphi_f(r)$,不更新当前值 $\varphi_c(r)$,而后者需要同时更新 $\varphi_c(r)$ 和 $\varphi_f(r)$.

对于表达式 e,$\varphi_c(e)$ 表示当前状态 φ_c 中 e 的值,例如,e 中所有变量 v 都被 $\varphi_c(v)$ 替换. 我们通过在 DD 模型上进行标识来区别这些差异,如图 1 所示,赋值前的标识"N:"表示非阻塞赋值,前缀"B:"表示阻塞赋值.

通过遍历 DD 模型中各变量(信号)对应的 DD 模型,可以生成 Verilog 描述的有限状态机模型.

③ 初始状态

所有在 initial 语句中被赋值的变量的值构成状态机的初始状态值集合. 对所有这样的初始状态值,建立其形式化模型为

$$I := \{s \in S \mid \bigwedge_{r \in L} s(r) = \varphi_f^I(r)\}.$$

如图 1 的初始状态为 $x=1 \wedge y=0$.

④ 变量赋值

通过遍历每个变量的 DD 模型,可以生成该变量的赋值形式化模型. 从 DD 模型的根结点出发进行遍历,若为判断结点,则生成一个"$c?t:f$"表达式,若为赋值结点,直接生成赋值表达式. 遍历过程到达终端结点时结束. 具体的遍历算法见文献[17]. 生成变量赋值时,需注意的是对非阻塞赋值和阻塞赋值语义的区别. 例如,对图 1 中的示例,生成的两个赋值语句如下所示(其中 x' 和 y' 表示 x 和 y 在下一状态的赋值):

$$y' = x,$$
$$x' = ((x < 100)?(x+y):x).$$

⑤ 下一状态

首先,根据每个变量的 DD 模型,可以识别出该变量是 L 中的状态变量,还是 C 中的组合变量. 对组合模块中的变量(信号)的赋值,要求每个变量的新值等于赋值后的值,即组合模块赋值的形式化模型为 $C := \bigwedge_{v \in C} s(v) = s(\varphi_f^C(v))$. 对时序模块中的状态变量(信号),除了按上述方法生成状态变量在下一状态的赋值外,还要根据其赋值生成状态迁移关系模型.

状态迁移关系 $R(s, s')$ 定义如下：对于每个 $v \in L$，v 的下一状态值为执行赋值之后 v 的当前值. 同时考虑连续赋值和组合模块赋值的约束 A，状态迁移关系可以表示为

$$R(s,s') := \bigwedge_{v \in L} s'(v) = s(\varphi_{\mathrm{f}}^{S}(v)) \wedge \bigwedge_{v \in C} s(v) = s(\varphi_{\mathrm{f}}^{C}(v)) \wedge s(A).$$

例如，对图 1 中的设计，其状态迁移关系如下所示：

$$R(x,y,x',y') := (x' = ((x < 100)?(x+y):x)) \wedge (y' = x).$$

3　谓词抽象

在谓词抽象中，谓词在状态集合 S 上定义了一个等价关系，这个等价关系对状态集合 S 进行划分，每一个状态属于一个等价类，两个状态等价当且仅当它们在同一个等价类中. 一个等价类用一个抽象状态表示.

给定一组谓词 $\Phi_1, \Phi_2, \cdots, \Phi_N$，定义一组布尔变量 $B_1, B_2, \cdots, B_N$（B_i 与谓词 Φ_i 对应），原始设计的状态用该状态满足的谓词公式 Φ 表示，抽象模型状态集合 A 是变量 $B_1, B_2, \cdots, B_N$ 上的一组正交布尔表达式. 原始状态和抽象状态之间的对应关系由抽象算子 α 和具体算子 γ 确定. 抽象算子 α：$S \rightarrow A$ 把原始状态 Φ 映射到抽象状态，具体算子 γ：$A \rightarrow S$ 是抽象算子的逆运算，把抽象状态映射到原始状态，又称精化算子. 抽象迁移关系由原始迁移关系计算得到. 文献[2]描述了利用定理证明器 PVS 构造完整的抽象状态图的算法.

本文采用的谓词抽象的方法与文献[13]类似，我们做了一些改动. 在谓词抽象中[2]，原始程序的变量被对应于原始程序变量上的谓词的布尔公式替换. 这些谓词将具体状态 $\bar{r} \in S$ 映射为布尔值. 设集合 $B = \{\pi_1, \pi_2, \cdots, \pi_n\}$ 为给定程序上的谓词集合，当对特定原始状态应用所有谓词的时候，我们可以获得布尔值的一个向量，表示一个抽象状态 $\bar{b}$. 抽象模型从抽象状态 $\bar{b}$ 迁移到 $\overline{b'}$ 当且仅当在原始模型中存在从 $\bar{r}$ 到 $\overline{r'}$ 的迁移，其中 $\bar{r}$ 抽象为 $\bar{b}$，$\overline{r'}$ 抽象为 $\overline{b'}$. 符号变量 b_i 同每个谓词 π_i 相关. 如果原始状态机从状态 $\bar{r}$ 到 $\overline{r'}$ 执行一个状态变迁，则抽象状态机也从 $\bar{b}$ 到 $\overline{b'}$ 执行一个变迁，其中 $b'_i = \pi_i(\overline{r'})$. 令得到的抽象状态机为 $\hat{T}$，用 $\hat{R}$ 表示 $\hat{T}$ 的迁移关系：

$$\hat{R} := \{(\bar{b}, \overline{b'}) \mid \exists \bar{r}, \overline{r'} : \bigwedge_{i=1}^{k} b_i = \pi_i(\bar{r}) \wedge R(\bar{r}, \overline{r'}) \wedge \bigwedge_{i=1}^{k} b'_i = \pi_i(\overline{r'})\}.$$

按照该方法，初始状态 $I(\bar{r})$ 的抽象定义为

$$\hat{I} := \{\bar{b} \mid \exists \bar{r} : \bigwedge_{i=1}^{k} b_i = \pi_i(\bar{r}) \wedge I(\bar{r})\}.$$

例如，对图 1 所示示例，其原始状态变迁关系为

$$R(x,y,x',y') := (x' = ((x < 100)?(x+y):x)) \wedge (y' = x).$$

假设要证明其满足 AG$(x < 100)$，谓词集合为 $\{x < 200, x < 100, x + y < 200\}$，用符号变量 b_1, b_2, b_3 分别表示每个谓词，形成如下的式子：

$$(b_1 \Leftrightarrow (x < 200)) \wedge (b_2 \Leftrightarrow (x < 100)) \wedge (b_3 \Leftrightarrow (x + y < 200)) \wedge R(x,y,x',y') \wedge (b'_1 \Leftrightarrow (x' < 200)) \wedge (b'_2 \Leftrightarrow (x' < 100)) \wedge (b'_3 \Leftrightarrow (x' + y' < 200)).$$

初始状态的计算公式为

$$(b_1 \Leftrightarrow (x < 200)) \wedge (b_2 \Leftrightarrow (x < 100)) \wedge (b_3 \Leftrightarrow (x + y < 200)) \wedge (x = 1) \wedge (y = 0).$$

4　CLP 约束生成

我们采用 GNU Prolog[18]（GProlog）作为底层的 CLP 约束求解器，需要将上一节中生成的抽象公式转化为 CLP 约束程序.

HDL 描述中的信号变量包括位、位向量和整型，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.

(a) 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 $\boldsymbol{V}1$ 和 $\boldsymbol{V}2$ 的比较运算“$\boldsymbol{V}1 == \boldsymbol{V}2$”，生成的约束如下所示（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）：

$$V1_0 = V2_0;$$
$$V1_1 = V2_1;$$

$V1_2 = V2_2$;

$V1_3 = V2_3$;

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

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

(b) 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"$\boldsymbol{V}[1:0] == 2'b00$",$\boldsymbol{V}$ 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

$V_0 = 0$;

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

(c) 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.

对 Verilog 算术和逻辑运算,需要考虑位向量和整数运算的溢出问题,即位向量和整数有一个最大宽度问题. 对这样的运算,利用 GProlog 的求余运算来建模. 假设需要转换的表达式为"$a\ op\ b$",其中 a 和 b 为位向量或整数,其宽度为 n,op 为任意的运算,如加减乘除等,则转化成的 CLP 约束为"$(a\ op\ b)\ rem\ 2^n$"(对其余的运算,可以直接进行转换. 详细的转化过程请参考文献[18]):

$$
\begin{aligned}
&Z\#=(X+Y)\ rem\ 256,\\
&Temp1\#<=>((\#\backslash B1)\#\backslash/(X\#<200))\#/\backslash(B1\ \#\backslash/(X\ \#>=200)),\\
&Temp2\#<=>((\#\backslash B2)\#\backslash/(X\#<100))\#/\backslash(B2\ \#\backslash/(X\ \#>=100)),\\
&Temp3\#<=>((\#\backslash B3)\#\backslash/(Z\#<200))\#/\backslash(B3\ \#\backslash/(Z\ \#>=200)),\\
&Y_Ba\ \#=X,\\
&(X\#<100)\#==>(X_Bar\#=Z),\ (X\#>=100)\#==>(X_Bar\#=X),\\
&Z_Bar\#=(X_Bar+Y_Bar)\ rem\ 256,\\
&Temp4\#<=>((\#\backslash B1_Bar)\#\backslash/(X_Bar\#<200))\#/\backslash(B1_Bar\#\backslash/(X_Bar\#>=200)),\\
&Temp5\#<=>((\#\backslash B2_Bar)\#\backslash/(X_Bar\#<100))\#/\backslash(B2_Bar\#\backslash/(X_Bar\#>=100)),\\
&Temp6\#<=>((\#\backslash B3_Bar)\#\backslash/(Z_Bar\#<200))\#/\backslash(B3_Bar\#\backslash/(Z_Bar\#>=200)),\\
&Out\#<=>Temp1\#/\backslash Temp2\#/\backslash Temp3\#/\backslash Temp4\#/\backslash Temp5\#/\backslash Temp6,
\end{aligned}
$$

抽象公式通常是一个合取式,对抽象公式进行转化时,首先为合取式中的每一个等价式生成一个临时变量,然后用所有的临时变量构成一个新的合取式,最后,令该合取式的取值为 1,对生成的约束进行求解,即可得到所有的状态迁移关系,即抽象系统的状态迁移函数. 例如,对图 1 所生成的抽象公式,转化后的 CLP 约束如上所示.

当令变量 Out 取值必须为 1 作为约束目标进行求解时,可以用 GProlog 的 findall 命令将所有的状态迁移关系求出. 抽象系统的状态迁移关系如图 2 所示. 对初始状态抽象公式可用同样的方法进行求解,对图 1 的示例,抽象系统的初始状态为"111".

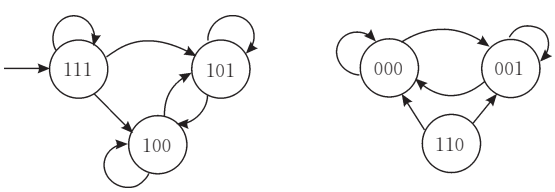

图 2 抽象系统状态迁移关系

将计算所得的抽象系统转换成 SMV 或 NuSMV 设计,并将待验证属性转换成关于抽象系统的属性,即可对设计进行验证. 例如,对图 2 的抽象系统,待验证的属性为 $AG(b_2)$.

5 实验及其分析

基于文献[18]的工作,我们实现了基于 CLP 的 RTL Verilog 谓词抽象原型系统. 为了与基于 SAT 的方法进行比较,我们按照文献[12]中的方法将抽象公式转换为 SAT 求解器可识别的格式. 首先将抽象公式用 VIS 转化成 BLIF 格式,并利用改进后的 BLIF2CNF① 工具将 BLIF 电路转换成 SAT 求解器的 CNF 输入格式. 实验中使用 zChaff[19] 作为 SAT 求解器,并根据文献[20]的算法对 zChaff 进行了改造,以求出所有满足 SAT 问题的解.

表 1 是实验电路的特征数据. 其中 8051 Controller 和 Viper 是两个微处理器的实现. Decoder 设计是一个 32 位微处理器核流水线译码器的 RTL 级设计. SMU 和 DCU 分别是 PicoJava 微处理器② 的数据 Cache 控制部件和栈管理器.

根据要验证的属性,对给定的谓词集合,首先建

① Joao Marques Silva. BLIF2CNF. sat. inesc-id. pt/~jpms/scripts/bin/blif2cnf.

② Sun Microsystems. PicoJava technology. http://www.sun.com/microelectronics/communitysource/picojava,1999.

立设计的状态机模型，然后提取出抽象公式. 对得到的抽象公式，分别用我们的方法和基于 SAT 的方法进行求解. 实验平台配置为 AMD Athlon XP 1.8GHz，1GB 内存，Windows 2000 操作系统. 实验结果如表 2 所示.

表 1　实验电路特性

电路	代码行	输入数	信号数
8051 Controller(8051)	350	18	59
Viper Microprocessor(Viper)	400	24	57
Decoder	2092	14	94
SMU	1476	30	217
DCU	3979	43	385

表 2　实验结果

电路	谓词数	CLP 方法		SAT 方法	
		约束行数	求解时间/s	文字数	求解时间/s
8051 Controller	23	239	10.10	13972	58
Viper Microprocessor	25	247	12.34	24719	204
Decoder	25	253	15.47	17527	177
SMU	39	372	23.73	35836	361
DCU	24	255	17.49	21075	236

表 2 中，CLP 方法中的“约束行数”是根据抽象公式生成的 CLP 约束代码行，SAT 方法中的“文字数”是将抽象公式转换成 CNF 格式后生成的文字数量. 求解时间单位为秒，CLP 和 SAT 的“求解时间”都是求所有解的时间.

从实验结果可以看出，基于 CLP 的方法比基于 SAT 的方法在求解抽象模型时的速度快很多，显示了该方法的优势. 基于 SAT 的方法的求解速度受两个方面因素的影响：将字级约束分解为位级约束导致问题复杂度提高，特别是当抽象中涉及乘法和除法运算时；文献[21]用 SAT 方法求全部可满足解的算法的实现效率不够高. 但是，我们认为将字级约束分解到位级约束是根本原因.

6　结束语

本文提出了一种基于 CLP 的谓词抽象方法，该方法提供一个统一的框架用于描述各种约束，无需将字级约束分解到位级约束，使得抽象系统的计算速度得到大大提高. 实验结果表明，基于 CLP 的方法与 SAT 方法相比，求解速度提高 10 倍之多. 下一步的工作是，进一步研究基于 CLP 的方法在谓词抽象中的应用，如基于 CLP 的抽象和精化、抽象反例验证、不可满足核的抽取等，并进一步完善我们的原型系统，使其更实用.

参　考　文　献

[1] Clarke E, Grumberg O, Peled D. Model Checking. Cambridge, MA: MIT Press, 1999

[2] Graf S, Saidi H. Construction of abstract state graphs with PVS//Proceedings of the 9th International Conference on Computer Aided Verification (CAV′97). Haifa, Isreal, 1997: 72-83

[3] Ball T, Cook B, Lahiri S K, Zhang Lin-Tao. Zapato: Automatic theorem proving for predicate abstraction refinement//Proceedings of the Computer Added Verificatoin (CAV′2004). Boston, MA, USA, 2004: 457-461

[4] Detlefs D, Nelson G, Saxe J B. Simplify: A theorem prover for program checking. Journal of the ACM, 2005, 52(3): 365-473

[5] Ball T, Rajamani S K. Boolean programs: A model and process for software analysis. Cambridge, UK: Microsoft Research, Technical Report 2000-14, 2000

[6] Flanagan C, Qadeer S. Predicate abstraction for software verification//Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL′02). Portland, Oregon, 2002: 191-202

[7] Ball T, Majumdar R, Millstein T D, Rajamani S K. Automatic predicate abstraction of C programs//Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI′2001). Snowbird, Utah, USA, 2001: 203-213

[8] Ball T, Rajamani S K. Automatically validating temporal safety properties of interfaces//Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN′2001). Toronto, Canada, 2001: 103-122

[9] Henzinger T A, Jhala R, Majumdar R, Sutre G. Software verification with Blast//Proceedings of the 10th International Workshop on Model Checking of Software (SPIN′2003). Portland, Oregon, 2003: 235-239

[10] Chaki S, Clarke E, Groce A, Jha S, Veith H. Modular verification of software components in C. IEEE Transactions on Software Engineering, 2004, 30(6): 388-402

[11] Clarke E, Talupur M, Veith H, Wang D. SAT based predicate abstraction for hardware verification//Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT′03). Portofino, Italy, 2003: 78-92

[12] Clarke E, Kroening D, Sharygina N, Yorav K. Predicate abstraction of ANSI-C programs using SAT. Formal Methods in System Design, 2004, 25(2-3): 105-127

[13] Clarke E, Jain H, Kroening D. Predicate abstraction and refinement techniques for verifying Verilog. Pittsburgh, PA: Carnegie Mellon University, Technical Report CMU-CS-04-139, 2004

[14] Clarke E, Jain H, Kroening D. Verification of SpecC using predicate abstraction//Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for

Co-Design (MEMOCODE 2004). San Diego, California, USA, 2004: 7-16

[15] Jaffar J, Maher M J. Constraint logic programming: A survey. The Journal of Logic Programming, 1994, 14(19-20): 503-582

[16] Ubar R. Test synthesis with alternative graphs. IEEE Design & Test of Computers, 1996, 13(1): 48-57

[17] Li Tun. Research on techniques of VLSI RT-Level automatic functional vectors generation[Ph. D. dissertation]. National University of Defense Technology, Changsha, 2003

[18] Li Tun, Guo Yang, Li Si-Kun. Functional vectors generation for RT-Level Verilog descriptions based on path enumeration and constraint logic programming//Proceedings of the 8th Euromicro Conference on Digital System Design. Porto, Portugal, 2005: 17-25

[19] Moskewicz M, Madigan C, Zhao Y, Zhang L, Malik S. Chaff: Engineering an efficient SAT solver//Proceedings of the 38th Design Automation Conference (DAC'01). Las Vegas, 2001: 530-535

[20] McMillan K. Applying SAT methods in unbounded symbolic model checking//Proceedings of the 14th Conference on Computer Aided Verification (CAV'02). Copenhagen, Denmark, 2002: 250-264

LI Tun, born in 1974, Ph. D., associate professor. His main research interests include high confidential computing, microprocessor design & verification, electronic CAD.

QU Wan-Xia, born in 1972, Ph. D. candidate. Her research interests include computer architecture, VLSI design & verification.

GUO Yang, born in 1971, Ph. D., associate researcher fellow. His main research interests include microprocessor design & verificatioin.

LIU Gong-Jie, born in 1977, M. S.. His main research interests focus on electronic CAD.

LI Si-Kun, born in 1941, professor, Ph. D. supervisor. His main research interests include methodology of VLSI design, virtual reality, etc.

Background

Predicate abstraction is a powerful technique for addressing state explosion problem in model checking. Theorem prover is the first solver for computing abstract model, and now more and more attention is paid to predicate abstraction based on SAT. However, there are some limitations when using theorem prover and SAT for predicate abstraction. Firstly, theorem prover based method has to call the theorem prover many times during abstraction, which will make the abstraction process inefficient. Secondly, theorem provers model the variables using unbounded integer numbers. Overflow or bit-wise operators are not modeled. However, hardware description languages like Verilog provide an extensive set of bit-wise operators. Thirdly, although SAT based method can only call the SAT solver one time during abstraction, it has to flatten the word-level constraints into bit-level constraints, which will lose most of the word-level information and the runtime of this process typically grows exponentially with the number of predicates.

This paper focuses on applying constraint logic programming (CLP) to predication abstraction of RTL Verilog descriptions, especially using CLP to solve the abstraction computation constraints obtained from circuit model and predicates. CLP-based method can model bit, bit-vector and bounded integer in a uniform framework, and can support various arithmetic and logic operations. What's more, the word-level constraints are solved with word-level information and without flattening them into bit-level constraints. Owing to the expressiveness of logic programming and the constraints solving techniques, CLP-based method bridges the gap between EDA research and the research progress in constraint satisfaction problem and artificial intelligence area.

This work is supported by the National Science Foundation of China (NSFC) under Grant No. 60403048 and No. 60573173, which focus on SOC system-level function verification and coverage-driven functional verification. Researchers from the two groups have an in-depth study in ASIC design and verification, parallel test generation and VLSI simulation vector generation domains. They have taken part in the design and verification in some microprocessors and ASIC chips during the last few years and obtained a series of significant results in theory, algorithm, design method and environment.

Research on coverage-driven VLSI functional verification and SOC system-level functional verification will speedup SOC design verification progress and improve the quality of SOC design, which is very important to promote the design and verification abilities of integrated circuit in China.