一种形式化验证方法:模型检验

杨 军, 葛海通, 郑飞君, 严晓浪
(浙江大学 超大规模集成电路设计研究所, 浙江 杭州 310027)

摘 要:模型检验作为一种形式化验证方法,近年来在各种硬件、软件设计中得到了广泛应用.文中首先介绍了描述系统行为的Kripke结构和描述系统性质的CTL逻辑,然后介绍了模型检验中常用的两种算法:标记算法和基于固定点的算法,最后介绍了为避免内存爆炸而引入的符号模型检验技术.

关 键 词:模型检验; Kripke结构; CTL逻辑; 标记; 固定点; 符号模型检验
中图分类号: TP301 文献标识码: A 文章编号: 1008-9497(2006)04-403-05

YANG Jun, GE Hai-tong, ZHENG Fei-jun, YAN Xiao-lang(*Institute of VLSI Design, Zhejiang University, Hangzhou* 310027, *China*)

A formal verification method: model checking. Journal of Zhejiang University(Science Edition), 2006, 33(4): 403~407

Abstract: Being a formal verification method, model checking is used frequently in hardware and software design. Firstey the Kripke structure which is used to describe the behavior of a system and the CTL logic which is used to describe the property of a system were introduced. Then two model checking algorithms were introduced: one is the labeling algorithm, the other is the algorithm based on fixpoint. In the end, the symbolic model checking which can reduce the possibility of memory explosion was introduced.

Key words: model checking; Kripke structure; CTL logic; labeling; fixpoint; symbolic model checking

0 引 言

集成电路设计中传统的验证方法有模拟和仿真,除非模拟或仿真了所有的测试向量,否则它们不能保证完全的错误覆盖,同时这两种方法所耗费的时间也是惊人的.随着电路规模的不断增大,它们的不足也日益突出,严重影响到整个产品的面市时间.形式化验证技术用数学的方法验证一个设计的功能,能保证对所有可能的情况进行验证,在集成电路设计过程中得到了越来越多的应用[1~3].

形式化验证技术在集成电路设计的各个层次都得到了运用,主要的方法有定理证明(theorem proving)、模型检验(model checking)和等价性验证(equivalence checking)[4].它们的侧重点各有不同,等价性验证虽然可以保证设计实现(implementation)与设计原型(specification)的功能一致,但它却不能确保设计原型的正确性,为此还必须对设计原型进行验证,这就需要定理证明和模型检验.表现在具体的设计过程中,设计者根据产品要求写出系统的行为级描述后,需要用定理证明或模型检验技术验证其正确性,通过这一关之后,设计者需要将行为级描述逐步转化为RTL级、门级和版图级描述,每一步转化之后的描述都需要用等价性验证来确保功能的一致性.

定理证明根据公理和形式推演规则来验证设计实现是否满足要求,由于需要人工干预,能处理的设计规模较小,发展比较缓慢.而模型检验由于能实现机器自动处理,处理的问题规模较大,在实际设计中的运用也渐渐增多.模型检验的基本思想是用有限状态机(FSM)表示系统的状态转移结构,用时序逻辑(如CTL、LTL等)表示设计的性质,通过遍历有限状态机来检验系统是否具有时序逻辑所表达的性质,如果不符合,模型检验工具可以给出一个反

收稿日期: 2005-04-19.
基金项目: 国家自然科学基金资助项目(No. 90207002).
作者简介: 杨 军(1981-),男,博士研究生,主要从事集成电路的形式化验证.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

例[3, 5, 6].

最初发展起来的模型检验工具都显式表示系统的状态和状态转移结构,随之而来的问题就是占用过多内存,不能处理较大规模的设计,伴随着 ROBDD(Reduced Ordered Binary Decision Diagram)[7]的发展,出现了符号模型检验技术[8, 9],它用布尔公式表示系统的状态、状态之间的转移关系以及 CTL 公式,所有模型检验所涉及的运算都通过布尔运算来实现,这就是隐式表示. 布尔公式的表达和运算都可以通过 BDD 很高效地实现,避免了内存爆炸问题,处理问题的规模也随之增大.

1 Kripke 结构

Kripke 结构是一个五元组 $K =< S, S_0, R, AP, L >$,其中 S 表示系统的状态空间集合,S_0 表示系统的初始状态空间集合,它是 S 的一个子集,$R \subseteq S \times S$ 表示系统的状态转移关系,AP 表示所有的原子命题和它们否定命题的集合,$L: S \to 2^{AP}$ 是标记函数,它把系统的每个状态映射为在此状态中成立的原子命题的集合,这个原子命题的集合是 AP 的子集.

2 分支时态逻辑 CTL

分支时态逻辑 CTL(Computation Tree Logic)是一种在模型检验中运用最多的时序逻辑,它由两个基本部分构成,一部分是路径量词,包括 A(对所有的路径)和 E(存在一个路径);另一部分是时态运算符,包括 G(Global)、F(Final)、X(neXt)和 U(Until). 在 CTL 逻辑中,时态运算符前必须有路径量词,这也是 CTL 逻辑区别于 CTL * 逻辑的地方. 举例来说,假设 f 是一个原子命题,Gf 在 CTL * 逻辑中是合乎规定的,而在 CTL 逻辑中则是不允许的,必须以 AGf 或 EGf 的形式存在.

CTL 逻辑公式的语法定义如下:

(1) 每个原子命题都是 CTL 公式;

(2) 如果 f, g 是 CTL 公式,则 $\neg f$、$f \wedge g$、AXf、EXf、$A(fUg)$ 和 $E(fUg)$ 也是 CTL 公式;

(3) 只有有限次的运用规则(2)得到的公式才是 CTL 公式.

其他一些常用的 CTL 公式可由这些基本公式得到:

$$f \vee g = \neg(\neg f \wedge \neg g),$$

$$AFf = A(\text{true}Uf),$$

$$EFf = E(\text{true}Uf),$$

$$AGf = \neg E(\text{true}U \neg f),$$

$$EGf = \neg A(\text{true}U \neg f).$$

CTL 公式跟时序相关,也就是说 CTL 公式伴随着状态的变化,为此引入表达式 $K, s \models f$,它表示公式 f 在 Kripke 结构 K 的状态 s 为真(在不致引起误解的情况下,可以简略为 $s \models f$). 用 $(s_0, s_1, s_2, \cdots)$ 表示一条从状态 s_0 开始的无限路径,其中 $(s_i, s_{i+1}) \in R$ 对于 $\forall i \geqslant 0$.

CTL 公式的语义定义如下:

(1) $s \models p$ 表示原子命题 p 在状态 s 成立;

(2) $s \models \neg f$ 表示公式 f 在状态 s 不成立;

(3) $s \models f \wedge g$ 表示公式 f 和公式 g 在状态 s 都成立;

(4) $s \models AXf$ 表示在以状态 s 开始的所有路径上,f 在 s 的下一个状态成立;

(5) $s \models EXf$ 表示存在一条以状态 s 开始的路径,在这条路径上,f 在 s 的下一个状态成立;

(6) $s \models A(fUg)$ 表示在以状态 s 开始的所有路径上,存在一个状态 s_k,在 s 到 s_k 的所有状态上,f 成立,在 s_k 之后的所有状态上,g 成立;

(7) $s \models E(fUg)$ 表示存在一条以状态 s 开始的路径,在这条路径上,存在一个状态 s_k,在 s 到 s_k 的所有状态上,f 成立,在 s_k 之后的所有状态上,g 成立;

用 CTL 逻辑可以表达系统的很多性质,如:

$AG(\text{req} \to AF\text{ack})$ 表示对所有状态,如果在当前状态发出请求信号 req,那么系统总可以在未来的某个状态产生回应 ack.

3 模型检验算法 ——labeling

如果 CTL 公式 f 在 Kripke 结构 K 的所有初始状态 s_0 都成立,则称此结构 K 为公式 f 的一个模型,记为 $K \models f$. 要检查系统 K 是否具有某性质 p,就要检查 p 是否对系统 K 的所有初始状态都成立,如果 p 在系统 K 的某一个初始状态不成立,那就认为系统 K 不具有性质 p.

labeling 算法的基本思想是对系统 K 中的每一个状态 s 标记以 label(s),它是一个集合,这个集合中的元素都是在状态 s 中成立的公式 f 的子公式. 初始时,label(s) 就是 $L(s)$,也就是说此时 label(s) 中的元素是在状态 s 中成立的原子命题;然后对公式 f 从里向外逐步处理,如果某个子公式在状态 s 成立,将此子公式加入到 label(s) 中. 在处理第 i 层的子公

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

式时, 所有小于 i 层的子公式都必须已经处理完毕; 对于第 i 层的子公式来说, 第 $i-1$ 层的子公式又可以被看成是在某些状态 s 中成立的原子命题.

例如对于公式 $AG(p \rightarrow AFq)$, 应该按以下几步由里向外逐步处理:

(1) p, q;

(2) AFq;

(3) $p \rightarrow AFq$;

(4) $AG(p \rightarrow AFq)$.

因为低一级的子公式在处理完毕之后可以被看成是原子公式, 所以对复杂公式的处理可以化简为对诸如 $\neg f$、$f \wedge g$、$f \vee g$、AXf、EXf、AFf、EFf、AGf、EGf、$A(fUg)$ 和 $E(fUg)$ 等简单形式的处理 (其中 f 和 g 都是原子公式).

$\neg$, $\wedge$, AF, EX 和 EU 是 CTL 逻辑的一组完备集, 因此 labeling 算法只须对这几种简单的运算符处理即可, 其他的运算都可以通过上述几种运算的不同组合得到. 首先, 需要对原子命题进行处理, 也就是说将在各个状态中成立的原子命题分别加入到状态的 label(s) 中. 接下去就可以对合式公式进行处理了:

(1) 对于公式 $\neg f$: 如果公式 f 在状态 s 不成立, 将 $\neg f$ 加入到状态 s 的 label(s) 中.

(2) 对于公式 $f \wedge g$: 如果公式 f 和公式 g 都在状态 s 中成立, 则将 $f \wedge g$ 加入到状态 s 的 label(s) 中.

(3) 对于公式 AFf: 如果公式 f 在状态 s 成立, 则将 AFf 加入到状态 s 的 label(s) 中; 对结构中的所有状态逐个进行处理, 如果对某个状态 s', 它的所有直接后继都有公式 AFf 成立, 则将 AFf 也加入到状态 s' 的 label(s') 中.

(4) 对于公式 EXf: 对结构中的所有状态逐个进行处理, 如果对某个状态 s, 它有一个直接后继使公式 f 成立, 则将 EXf 加入到状态 s 的 label(s) 中.

(5) 对于公式 $E(fUg)$: 如果公式 g 在状态 s 成立, 则将 $E(fUg)$ 加入到状态 s 的 label(s) 中; 对结构中的所有状态逐个进行处理, 如果对某个状态 s', 它使公式 f 成立, 且状态 s' 的某个直接后继使公式 $E(fUg)$ 成立, 则将 $E(fUg)$ 也加入到状态 s' 的 label(s') 中.

如此由里向外直到程序终止, 如果对要检验的性质 f, 有 $f \in$ label(s), 则有 $K, s \models f$, 如果对所有初始状态 s_0, 都有 $K, s_0 \models f$ 成立, 则称公式 f 在结构 K 中成立, 结构 K 满足性质 f. 此算法对每一个状态逐一进行处理, 判断子公式在此状态中是否成立, 其算法复杂度为 $O(|f| * |S| * (|S| + |R|))$, 其中 $|f|$ 表示公式 f 的子公式的数目, $|S|$ 表示结构 K 中状态的数目, $|R|$ 表示结构 K 中状态转移的数目.

另一种改进的 labeling 算法[6] 并不是对所有的状态进行逐一的判断, 算法复杂度可以降低到 $O(|f| * (|S| + |R|))$. 例如对 $E(fUg)$, 改进的算法如下:

```
procedure CheckEU(f, g)
  T = {s | g ∈ label(s)};
  for all s ∈ T do label(s) = label(s) ∪ {E[fUg]};
  while T ≠ Φ do
    choose s ∈ T;
    T = T - {s};
    for all t such that R(t, s) do
      if E[fUg] ∉ label(t) and f ∈ label(t) then
        label(t) = label(t) ∪ {E[fUg]};
        T = T ∪ {t};
      end if;
    end for all;
  end while;
end procedure
```

算法首先将公式 $E(fUg)$ 加入到使 g 成立的状态 s 的 label(s) 中, 并初始化集合 T, 它是所有使公式 $E(fUg)$ 成立的状态的集合. 对集合 T 中的任一状态 s, 如果它的某一个前继 t 使公式 f 成立, 则将 t 加入到集合 T 中, 并将 $E(fUg)$ 加入到状态 t 的 $label(t)$ 中, 当对状态 s 的所有前继都处理完毕时, 将其从集合 T 中删除. 如此对集合 T 中的所有元素逐个进行处理直到集合为空, 判断过程结束.

$\neg$, $\wedge$, AF, EX 和 EU 只是 CTL 逻辑的一种完备集, 不是唯一的完备集, labeling 算法也可以只对其他完备集中的运算符进行处理, 例如 $\neg$, $\vee$, EX, EU 和 EG 也是一种完备集.

4　模型检验算法 ——fixpoint

对 CTL 公式的检验还可以通过固定点的算法来完成, 首先引入函数单调性(monotonic)、并连续($\cup$- continuous)、交连续($\cap$- continuous)、固定点(fixpoint)、最小固定点(least fixpoint) 和最大固定点(greatest fixpoint) 等概念. 记 τ 为定义在集合 S 的幂集 2^S 上的函数, 即 $\tau: 2^S \rightarrow 2^S$.

(1) 如果对 $P \subseteq Q$, 有 $\tau(P) \subseteq \tau(Q)$ 成立, 则称

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

函数 τ 是单调的.

(2) 如果对 $P_1 \subseteq P_2 \subseteq \cdots$, 有 $\tau(\bigcup_i P_i) = \bigcup_i \tau(P_i)$ 成立, 则称函数 τ 是并连续的.

(3) 如果对 $P_1 \supseteq P_2 \supseteq \cdots$, 有 $\tau(\bigcap_i P_i) = \bigcap_i \tau(P_i)$ 成立, 则称函数 τ 是交连续的.

(4) 如果对于集合 P, 有 $\tau(P) = P$ 成立, 则称 P 为函数 τ 的一个固定点.

(5) 函数 τ 的所有固定点的交集称为它的最小固定点, 记为 $\mu Z.\ \tau(Z)$.

(6) 函数 τ 的所有固定点的并集称为它的最大固定点, 记为 $\nu Z.\ \tau(Z)$.

Tarski 在 1955 年已经证明对于单调函数 τ, 它存在最小固定点和最大固定点, 如果 τ 还是并连续的, 则有 $\mu Z.\ \tau(Z) = \bigcup_i \tau^i(\mathrm{False})$; 如果 τ 还是交连续的, 则有 $\nu Z.\ \tau(Z) = \bigcap_i \tau^i(\mathrm{True})$.

为了能用关于固定点的方法来检验 CTL 公式, 将公式用满足它的状态的集合表示, 即将公式 f 用集合 $\{s \mid s \models f\}$ 表示. 因为结构 K 中的状态 s 是有限的, Emerson 和 Clarke[10] 证明了对于定义在状态 s 上的函数 τ, 有下列公式成立:

$$EFp = \mu Z.\ (p \vee EX(Z)),$$

$$EGp = \nu Z.\ (p \wedge EX(Z)),$$

$$E(pUq) = \mu Z.\ (q \vee (p \wedge EX(Z))).$$

下面举一个例子来说明如何通过固定点判断 CTL 公式. 对于如图 1 所示结构 K 判断 EFp,

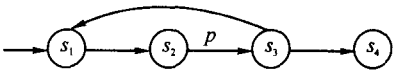

图 1 K 结构

Fig. 1 K structure

第 1 步: $\tau^1(\mathrm{False}) = p \vee EX(\mathrm{False}) = p$, 用状态集合表示就是 $\{s_2\}$;

第 2 步: $\tau^2(\mathrm{False}) = p \vee EX(\tau^1(\mathrm{False})) = p \vee EX(p)$, 用状态集合表示就是 $\{s_1, s_2\}$;

第 3 步: $\tau^3(\mathrm{False}) = p \vee EX(\tau^2(\mathrm{False})) = p \vee EX(p \vee EX(p))$, 用状态集合表示就是 $\{s_1, s_2, s_3\}$;

第 4 步: $\tau^4(\mathrm{False}) = p \vee EX(\tau^3(\mathrm{False})) = p \vee EX(p \vee EX(p \vee EXp))$, 用状态集合表示仍然是 $\{s_1, s_2, s_3\}$, 达到了固定点, 计算结束.

所以, 满足公式 EFp 的状态集合为 $\{s_1, s_2, s_3\}$, 这其中包含了初始状态 s_1, 所以结构 K 满足公式 EFp.

5 符号模型检验

上述方法中, 结构 K 的状态以及状态转移关系是显式的存储在内存中的, 当状态数目很大时会不可避免地出现内存爆炸, 阻碍了模型检验技术在现实设计中的应用. 经过众多研究者的努力, 符号模型检验技术得到了较快发展, 它的主要思想是将状态以及状态转移关系用布尔公式表示, 通过布尔运算完成状态集合之间的各种运算. 符号模型检验分为以下几步:

(1) 用布尔公式表示状态: 假设某个设计中有 n 个存储元件, 对应每个状态这 n 个存储元件都有一组独特的赋值, 也就是说每个状态都可以由这 n 个存储元件的一组赋值表示. 假设某设计中有 2 个存储元件 $V = \{v_1, v_2\}$, 对这两个存储元件有 4 种赋值: 00、01、10 和 11, 用布尔公式表示就是 $\neg v_1 \wedge \neg v_2$、$\neg v_1 \wedge v_2$、$v_1 \wedge \neg v_2$ 和 $v_1 \wedge v_2$, 每个布尔公式代表设计中的一个状态.

(2) 用布尔公式表示状态转移: 假设用布尔向量 $V = \{v_1, v_2, \cdots, v_n\}$ 表示现状态, 引入另一组布尔向量 $V' = \{v_1', v_2', \cdots, v_n'\}$ 表示次态. 对于结构中的每个状态转移 (s_i, s_j), 都有一系列 (v_i, v_i') 与之对应, 其中 $v_i \in V, v_i' \in V', i = 1, 3 \cdots, n$. 对于同步时序电路, 如果知道每一个存储元件的特征函数 $v_i' = f_i(V)$, 整个电路的转移关系可以用各个特征函数的逻辑与表示: $R(V, V') = \prod_1^n R_i(V, V')$, 其中 $R_i(V, V') = (v_i' \equiv f_i(V))$.

(3) 用布尔公式表示 CTL 原子命题逻辑: 因为可以将 CTL 原子命题用使之成立的状态集合表示, 而状态集合又可以用布尔公式表示, 所以可以将 CTL 原子命题用与之对应的布尔公式表示.

(4) 用布尔公式表示 CTL 逻辑运算: 假设用布尔公式 $B(f)$ 表示对应的 CTL 公式 f, 则

$$B(\neg f) = \neg B(f),$$

$$B(f \wedge g) = B(f) \wedge B(g).$$

$B(EXf) = \exists V'[R(V, V') \wedge B(f')]$, 其中 $B(f')$ 是将布尔公式 $B(f)$ 中的现态存储元件符号 v 用对应的次态存储元件符号 v' 代替, $R(V, V')$ 是用布尔公式表示的状态转移关系.

$$B(EGf) = \nu Z.\ [B(f) \wedge B(EXZ)],$$

$$B[E(fUg)] = \mu Z.\ [B(g) \vee [B(f) \wedge B(EXZ)]].$$

对其他诸如 EFf、AXf、AGf、AFf 和 $A(fUg)$ 等 CTL 逻辑公式的运算都可以通过对 EXf、EGf 和 $E(fUg)$ 等基本公式的运算来实现. 当然它们也可以直接实现:

$$B(EFf) = \mu Z.\ [B(f) \vee B(EXZ)],$$

$$B(AXf) = \forall V'[R(V, V') \Rightarrow B(f')],$$

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

$$B(AFf) = \mu Z.[B(f) \vee B(AXZ)],$$
$$B(AGf) = \nu Z.[B(f) \wedge B(AXZ)],$$
$$B[A(fUg)] = \mu Z.[B(g) \vee [B(f) \wedge B(AXZ)]].$$

用布尔公式表示状态和状态转移关系, 不再需要显式的构建状态图, 避免了内存爆炸. 对布尔公式的运算可以有多种方法, 而其中又以 BDD[7] 为简捷高效, 关于如何用 BDD 完成布尔运算, 本文不再做详细述说, 可参考文献[7, 11].

6 总　结

本文对模型检验进行了介绍, 首先介绍了模型检验的两个基本要素: 描述系统的 Kripke 结构和描述系统性质的 CTL 逻辑, 接着介绍了模型检验中常见的两种算法: labeling 算法和基于 fixpoint 的算法. 传统的模型检验由于显式的表示状态和状态转移关系, 常引起内存爆炸, 使处理的问题规模有所限制, 随着符号模型检验技术的兴起以及 BDD 的应用, 模型检验能处理问题的规模急剧增加. 现在模型检验不但应用在包括微处理器、存储器阵列、浮点运算器等各种硬件设计的验证中, 在包括 IEEE 协议和标准等各种软件设计的验证中也得到了广泛的应用.

参考文献(**References**):

[1] KERN C, GREENSTREET M R. Formal verification in hardware design: a survey[J]. **ACM Transactions on Design Automation of Electronic Systems**, 1999, 4(2): 123-128.

[2] HU ALAN J. Formal hardware verification with BDDs: an introduction[C] // **1997 IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing**. Victoria, Canada: IEEE Press, 1997: 677-682.

[3] 韩俊刚, 杜慧敏. 数字硬件的形式化验证[M]. 北京: 北京大学出版社, 2001.
HAN Jun-gang, DU Hui-min. **Formal Verification of Digital Hardware**[M]. Beijing: Peking University Press, 2001.

[4] HUANG Shi-yu, CHENG Kwang-ting. **Formal Equivalence Checking and Design Debugging**[M]. Boston: Kluwer Academic Publishers, 1998.

[5] CLARKE E M, GRUMBERG O, PELED D A. **Model Checking**[M]. Cambridge, Massachusetts: the MIT Press, 2001.

[6] CLARKE E M, EMERSON E A, SISTLA A P. Automatic verification of finite state concurrent systems using temporal logic specification[J]. **ACM Transactions on Programming Languages and Systems**, 1986, 8(2): 244-263.

[7] BRAYANT R E. Graph-based algorithms for boolean function manipulation[J]. **IEEE Transactions on Computer**, 1986, C-35: 667-691.

[8] BURCH J R, CLARKE E M, MCMILLAN K L, et al. Symbolic model checking: 10^{20} states and beyond[C] // **Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science**. Philadelphia: IEEE Press, 1990: 428-439.

[9] MCMILLAN K L. **Symbolic Model Checking: An Approach to the State Explosion Problem**[D]. Pittsburgh: Carnegie Mellon University, 1993.

[10] EMERSON E A, CLARKE E M. Characterizing correctness properties of parallel programs using fixpoints[C] // **Proceedings of the 7th Colloquium on Automata, Languages and Programming**. London: Springer-Verlag Press, 1980: 169-181.

[11] BRACE K S, RUDELL R L, BRAYANT R E. Efficient implementation of a BDD package[C] // **Proceedings of the 27th ACM/IEEE Design Automation Conference**. Orlando, Florida: IEEE Press, 1990: 40-45.

(责任编辑　涂　红)

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net