基于断言的模拟矢量自动生成方法*

李 暾+, 郭 阳, 李思昆

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Assertion-Based Automatic Generation of Functional Vectors

LI Tun+, GUO Yang, LI Si-Kun

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573673, E-mail: tunli@nudt.edu.cn, http://www.nudt.edu.cn

Received 2003-12-09; Accepted 2004-01-07

Li T, Guo Y, Li SK. Assertion-Based automatic generation of functional vectors. *Journal of Software*, 2004,15(10):1441~1450.

http://www.jos.org.cn/1000-9825/15/1441.htm

Abstract: Simulation-Based verification approaches need a large amount of test vectors for verifying the corner case of designs. This paper proposes a novel assertion-based automatic functional vectors generation method which takes assertion as the functional coverage metric. For the given assertion, the related design part is first extracted, and then the assertion property and the signal propagation process based on the DD model is converted to the CLP constraints; Finally, functional vectors are generated by solving the constraints. The advantage of this method is the combination of the program slicing based design extraction, the word-level SAT engine, and the dynamic search techniques. The method can deal with large designs and handle control and datapath design in a unified framework. Experimental results show that the method is efficient for finding the design errors and vectors generation.

Key words: VLSI; assertion; automatic functional vectors generation

摘 要: VLSI 模拟验证的一个关键问题是需要大量的模拟矢量来验证各种可能情况下设计的正确性.采用断言作为模拟验证的功能模型,提出和实现了一种基于断言的模拟矢量自动生成方法.针对要触发的断言,首先对设计进行化简,通过决策图模型将初始输入传播到断言,并将传播过程和断言条件一起转化成 CLP 约束,最后求解 CLP 约束生成模拟矢量.该方法的优势在于运用了字级(word-level)约束求解技术,能统一处理控制电路和数据通路间的数据传播,求解效率高;基于功能模型的模拟矢量生成技术,模拟矢量生成目标更明确;与动态加速技术相结合,使搜索过程效率更高;设计化简技术的运用使搜索过程计算复杂度只与断言有关.实验结果表明,该方法能快速找到并定位设计中的错误,生成模拟矢量效率更高.

关键词: VLSI;断言;模拟矢量自动生成

* Supported by the National Natural Science Foundation of China under Grant Nos.60303011, 90207019 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA1Z1480 (国家高技术研究发展计划(863))

作者简介: **李暾**(1974－),男,湖南郴州人,博士,讲师,主要研究领域为并行模拟,微处理器设计验证技术,电子 CAD 技术;**郭阳**(1971－),男,博士,副研究员,主要研究领域为微处理器设计验证技术,电子 CAD 技术;**李思昆**(1941－),男,教授,博士生导师,主要研究领域为电子 CAD,VLSI 设计方法学,虚拟现实技术.

中图法分类号: TP302 文献标识码: A

随着现代VLSI设计复杂度越来越高,验证已经成为设计过程的主要任务.由于形式化验证技术还不能处理规模较大的设计,尤其是全系统验证,目前,模拟验证仍然是主要的验证方法.模拟验证的一个主要问题是需要大量的模拟矢量尽可能地验证各种可能情况下设计的正确性.模拟矢量自动生成技术的研究已经取得较多成果,提出了各种模拟矢量自动生成方法[1~5].功能级模拟矢量自动生成的主要问题是没有像逻辑级那样的s-a-0/s-a-1 故障模型,因此需要定义各种模拟矢量生成目标模型.在已有的各种覆盖率模型中,功能模型是最贴近设计属性的模型,其他模型都是基于HDL描述或设计结构的.

近年来,模拟验证方法与形式化验证方法结合得越来越紧密,在模拟验证中加入了以前只用于形式化验证的方法.目前流行的基于断言的验证方法可以通过在设计描述中加入断言(assertion),来实时监测设计描述是否违反了断言属性.比较常用的断言描述方法有开放源码的OVL库[6]、IBM的Sugar语言[7]和Accellera的PSL语言[8]等,商品化工具有 0-in Check[9]等.也有很多工具基于断言对设计进行静态验证[10~12],这些验证技术都是基于形式化验证方法的,主要在设计的逻辑级(Bit-Level)进行分析,对规模较大的设计无法进行静态断言检测.文献[13]采用字级ATPG和模方程组方法进行静态断言验证,通过产生反例生成模拟矢量.虽然该方法能处理较大的设计,但是却需要在不同的求解器间传播约束信息,约束求解性能低.

基于断言的动态模拟验证的一个主要问题是如何找到使断言被触发的输入.在实际设计中,常常使用随机产生的模拟矢量来检验插入的断言是否被触发.往往模拟了很多周期,输入了大量的模拟矢量后仍然无法触发插入的断言.本文提出了一种基于断言的模拟矢量自动生成方法.针对要触发的断言,首先对设计进行化简,通过决策图模型将初始输入传播到断言,并将传播过程和断言条件一起转化成CLP约束,最后求解CLP约束生成模拟矢量.该方法运用了字级(word-level)约束求解技术,能统一处理控制电路和数据通路间的数据传播,字级约束求解效率更高;采用基于功能模型的模拟矢量生成技术,模拟矢量生成目标更明确;与动态加速技术相结合,使搜索过程效率更高;设计化简技术的运用使搜索过程计算复杂度只与断言有关,而不需要搜索整个设计空间.

本文设计实现了基于断言的模拟矢量自动生成原型系统,对一些实际设计进行了实验,并与现有基于路径覆盖的模拟矢量自动生成技术[3]和随机生成方法进行了比较.实验结果表明,本文方法在设计中发现错误的能力上有明显优势,能获得更高的功能覆盖率.本文工作主要针对Verilog语言,当然,该方法同样适用于其他HDL语言.

1 断言定义

采用类似于System Verilog[14]中定义断言的方法,定义了串行断言和并发断言.串行断言用于Verilog的进程语句内,可以看作是一种串行语句,并发断言描述的是设计的时序属性,需要监控一个信号序列以判断是否满足断言要求.

1.1 串行断言

串行断言语法的定义是:

```
squential_assert_statement ::=
//A: seq_assert (expression)
```

其中,“//A:”用于标识该注释语句是一条断言语句,seq_assert 是断言关键字,用于引起一个断言.expression 为Verilog中条件语句所使用的条件表达式.expression的求解结果是一个布尔值,为True或False.

我们定义了两种特殊的串行断言,分别是seq_assert_onehot和seq_assert_amone断言,其格式为

```
//A: seq_assert_onehot([expression])
//A: seq_assert_amone([expression])
```

seq_assert_onehot用于检测表达式或表达式列表中只有一个表达式的求解结果为“1”,通常用于检测one-hot编码的状态机.seq_assert_amone用于检测表达式或表达式列表中最多只有一个表达式的求解结果为“1”.

1.2 并发断言

并发断言描述的是在一个时间段上设计须满足的属性.通常这样的断言是由一个序列构成的,并发断言的语法定义如下所示.其中“->”和“=>”都是蕴涵运算符(implication).前者表示在匹配了蕴涵表达式前件的时刻同时开始求解蕴涵后件表达式,而后者表示将在前件表达式被匹配的下一节拍开始匹配后件表达式.需要有一个时钟信号来触发并发断言的检测,因此,可以将并发断言看作时钟触发的 Verilog 进程语句.

```
concurrent_assert_statement::=
        //A: con_assert(property_expr)
property_expr::=
          sequence_expr
        | sequence_expr |-> [ not ] sequence_expr
        | sequence_expr |=> [ not ] sequence_expr
        | ( property_expr )
sequence_expr ::=
         cycle_delay_range sequence_expr { cycle_delay_range sequence_expr }
        | sequence_expr cycle_delay_range sequence_expr { cycle_delay_range sequence_expr }
        | expression
cycle_delay_range ::=
        ## constant_expression
        | ## [ cycle_delay_const_range_expression ]
cycle_delay_const_range_expression ::=
        constant_expression : constant_expression
```

并发断言中序列表达式表示的是时间序列上的一组事件.例如, “##[0:3]a”表示 a 可在当前时刻开始的连续 3 个时刻内的任意时刻上为“1”.属性表达式“a |-> b ##1 c ##1 d”表示当前时刻 a 为“1”时,如果 b 为“1”,下一时刻 c 为“1”,再下一时刻 d 为“1”,则该属性表达式求解为真.

2 断言目标生成

在基于断言的模拟矢量自动生成方法中,需要将断言条件求反(invert)后转换成等价的 Verilog 语句,再以转换后 Verilog 语句为目标生成模拟矢量.

2.1 串行断言目标生成

串行断言的转化比较简单,可以直接对断言的条件表达式取反(not),再将断言转换成一条 if 语句即可.其转换方法为//A: assert(expression)=>if (not(expression)).

对 seq_assert_onehot 和 seq_assert_amone 断言,其转换方法为

//A: seq_assert_onehot($expr_1$,$expr_2$,…,$expr_n$)

⇒

if(($expr_1$+$expr_2$+…+$expr_n$)!=1);

//A: seq_assert_amone($expr_1$,$expr_2$,…,$expr_n$)

⇒

if(($expr_1$+$expr_2$+…+$expr_n$)>1.

2.2 并发断言目标生成

并发断言的求反需要对序列表达式作特殊处理.可以将事件序列转化成线性时态逻辑(linear temporal logic,简称 LTL)[15]形式的公式,再通过对公式的求反运算得到一个对任意事件序列都有效的求反操作和求反结

果,作为模拟矢量自动生成的目标.线性时态逻辑中,X 算子表示"下一时刻",对逻辑公式 f,X 算子满足$\neg Xf=X\neg f$,定义 $X[n]$为连续 n 个 X 操作,则有$\neg X_{[n]}f=X\neg X_{[n-1]}\neg f=\ldots=X_{[n]}\neg f$ 成立.

对任意形式的事件序列 $f_1\#\#n_1f_2\#\#n_2f_3\ldots\#\#n_mf_{m+1}$,其中 f_i,$1\le i\le m+1$ 为条件表达式,则可将其用 LTL 公式表示为 $f_1\rightarrow\left(x_{[n_1]}f_2\wedge X_{[n_1+n_2]}f_3\ldots\wedge X_{\left[\sum_{i=1}^{m}n_i\right]}f_{m+1}\right)$,其中$\rightarrow$为蕴涵公式.对该 LTL 公式取反的运算过程为

$$\neg\left(f_1\rightarrow\left(X_{[n_1]}f_2\wedge X_{[n_1+n_2]}f_3\ldots\wedge X_{\left[\sum_{i=1}^{m}n_i\right]}f_{m+1}\right)\right)=\left(f_1\wedge X_{[n_1]}\neg f_2\right)\vee\left(f_1\wedge X_{[n_1+n_2]}\neg f_3\right)\vee\left(f_1\wedge X_{\left[\sum_{i=1}^{m}n_i\right]}\neg f_{m+1}\right)\quad(1)$$

其中,得到的公式(1)就是最后进行模拟矢量生成的目标,由公式可以得到,当生成模拟矢量时,如果能生成满足$\left(f_1\wedge X_{[n_1]}\neg f_2\right)$的模拟矢量,则断言监控的事件序列被违反.如果在扩展了设定的周期后不能生成使$\left(f_1\wedge X_{[n_1]}\neg f_2\right)$满足的模拟矢量,则开始考虑生成使$\left(f_1\wedge X_{[n_1+n_2]}\neg f_3\right)$满足的模拟矢量.此时有$\left(f_1\wedge X_{[n_1]}f_2\right)$为真,因此,在考虑$\left(f_1\wedge X_{[n_1+n_2]}\neg f_3\right)$时,$X_{[n_1]}f_2$ 是为真的.依此类推,当在以公式(1)的最后一项为目标生成模拟矢量时,序列的前 m 个事件组成的前缀将成立,而只有最后一个事件 f_{m+1} 不匹配.

由此可得,在为事件序列生成模拟矢量时,尽可能地使事件序列中最左边的事件被违反,能尽早地生成使事件序列不被匹配的模拟矢量,尽早发现设计中的错误.公式(1)保证了这种方法的正确性.

对蕴涵形式的并发断言描述,只有当前件为真,后件为假时,才会违反该断言.其转换方法分为两部分,对蕴涵式前件生成使其事件序列匹配的模拟矢量,再对后件采用上述以事件序列为目标的模拟矢量生成方法,只是此时要在公式(1)中增加一项,即要首先考虑后件事件序列第 1 个事件不匹配的情况.

3 模拟矢量自动生成

约束逻辑编程(constraint logic programming,简称 CLP)在传统的逻辑编程(logic programming)框架中集成了约束求解能力,使得 CLP 程序在具有灵活描述能力的同时还具有很强的约束求解能力.本文选用的 GNU Prolog[19]系统是 CLP(FD)的一个改进版,采用 Prolog 语言作为描述语言,支持布尔变量的约束求解.由于功能模拟矢量自动生成可以转化为有限域的约束求解,因此,支持有限域约束求解的 CLP 平台就能满足本文的要求.GNU Prolog 系统的性能优势主要来自于:1) 集成了上述约束满足问题的求解技术;2) 利用 WAM(warren abstract machine)技术将约束编译成本机代码;3) 使用一种统一的格式定义算术和布尔约束,这种建模方法使约束传播运算更快.GNU Prolog 的另一个优势是开放源码.

对每一个断言,模拟矢量自动生成过程分为以下几步(设定扩展周期为 t):

(1) 设计化简,抽取出与断言所涉及的变量相关的设计描述.详细的设计化简方法请参考文献[16].

(2) 断言目标生成,对串行断言生成对断言取反后的等价 Verilog 语句,对并发断言,将其转换成等价的 Verilog 语句块.

(3) Verilog 描述建模.建立 Verilog 描述中的控制数据流模型,用于将断言处的信号值传播到初始输入以得到模拟矢量.建模主要以决策图(decision diagram)模型[17]为基础,建立信号间的控制和数据依赖关系.

(4) 以当前模拟时刻(假设为 s)为基础,根据 DD 模型为所有断言中使用到的变量生成 CLP 约束,与断言的 CLP 约束一起构成约束程序.

(5) 用 GNU Prolog 求解上述生成的 CLP 约束程序.如果有解,则结束.

(6) 如果无解,并且 $s<t$,则 $s=s+1$,跳至(4).

(7) 如果达到最大扩展周期数或使用的资源(内存)达到限制,结束.

在介绍算法前,我们给出一个 Verilog 描述的 FIFO 设计实例,包括插入的判断 FIFO 上溢和下溢的串行断言.该设计实例将作为后面算法介绍的例子.在设计中,我们修改了原来的代码,引入了一个错误,每次 *cnt* 增幅为 2,使得 FIFO 上溢.图 1(a)是 FIFO 设计错误的 Verilog 描述,图 1(b)是针对上溢断言进行模拟矢量生成时,对设计化简后得到的 Verilog 描述.可以看出,以断言所使用的信号变量为目标化简后,去除了许多无关的描述.

```
always @(posedge clk or negedge fifo_clr_n)
begin
  if (fifo_clr_n == 1'b0) begin
    top <= {3 {1'b0}};
    btm <= {3 {1'b0}};
    cnt <= { 4 {1'b0}};
    for (i = 0; i < 8; i=i+1)
      fifo[i] <= {16{1'b0}};
  end
  else if (fifo_reset_n == 1'b0) begin
    top <= {3 {1'b0}};
    btm <= {3 {1'b0}};
    cnt <= { 4 {1'b0}};
  end
  else begin
    case ({put, get})
      2'b10 : // WRITE
        if (cnt < 8) begin
          fifo[top] <= data_in;
          top <= top + 1;
          cnt <= cnt + 2;
          // Assert that the FIFO cannot overflow
          //A:seq_assert(cnt < 8);
        end
      2'b01 : //READ
        if(cnt>0) begin
          fifo[btm] <= 0;
          btm <= btm + 1;
          cnt <= cnt - 1;
        end
      2'b11 : // WRITE & READ
        begin
          fifo[btm] <= 0;
          fifo[top] <= data_in;
          btm <= btm + 1;
          top <= top + 1;
        end
    endcase
  end
end // always

assign data_out = fifo[btm];

//Assert that the FIFO cannot underflow
//A:seq_assert(!(get && cnt==0));
```

(a)

```
always /* fifo.v:19 */
 @(posedge clk or negedge fifo_clr_n)
  begin
    if ((fifo_clr_n)==(1'b0))
      begin
        cnt <= { 4 {1'b0}}; /* fifo.v:24 */
      end
    else
      if ((fifo_reset_n)==(1'b0))
        begin
          cnt <=  {4{1'b0}};  /* fifo.v:31 */
        end
      else
        begin
          case ({put, get}) /* fifo.v:34 */
            2'b10:
              if (cnt<8)
                begin
                  cnt <= cnt+2;  /* fifo.v:39 */
                  // Assert that the FIFO cannot overflow
                        //A:seq_assert(cnt < 8);
                end
            2'b01:
              if (cnt>0)
                begin
                  cnt <=  cnt-1;  /* fifo.v:47 */
                end
          endcase
        end
  end
```

(b)

Fig.1 FIFO verilog description

图 1 FIFO 设计描述

图 2 是对变量 *cnt* 建立的 DD 模型的示意图,DD 模型最早是由 Ubar[17]提出的,它不但能表示位级逻辑关系,而且能表示字级逻辑关系,能很好地用于表示 RTL 级和行为级的信号依赖关系.从结构上来说,DD 模型是二叉树结构,根结点是某个信号名称,如图 2 所示的“*cnt*”变量.椭圆结点表示叶结点,是对变量进行赋值的结点.“*cnt*+2”结点表示对 *cnt* 信号赋值为“*cnt*+2”.矩形结点表示控制结点,控制着变量赋值的执行.“fifo_clr_n==0”结点表示条件成立时,*cnt* 信号赋值为 0.

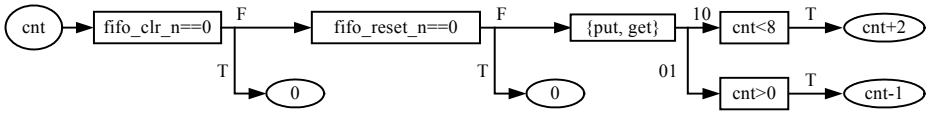

Fig.2 DD model for *cnt* variable

图 2 *cnt* 变量的 DD 模型

3.1 约束生成方法

模拟矢量生成时,由于要试探 DD 模型中的所有路径,为变量的各种可能赋值建立约束模型,因此需要为条件语句的所有求值结果生成约束.

在 Verilog 中,条件语句(if-else,case 等)可被视为多路选择器,其中 if-else 语句是 2-1 选择器,而 case 语句是多选一选择器.当蕴涵运算符的前件表达式条件不全时,例如,为条件表达式 condexpr 生成了约束“condexpr #==> result1”,而没有对应的“#\ condexpr #==> result2”时(表达式中的符号含义见文献[19]),求解的结果会包含大量的错误结果.通常称这种条件不全的条件语句为不完全多路选择器.

为不完全多路选择器生成约束模型时,如果为 2-1 选择器,则只要为缺失的分支再生成一条蕴涵运算语句即可,在附加的蕴涵语句中,变量的赋值将保持不变.如在图 2 中,为“*cnt*<8”生成的约束语句为

(Cnt_0_0#<8)#==>Cnt_0_1#=Cnt_0_0+2,

(Cnt_0_0#>8)#==>Cnt_0_1#=Cnt_0_0.

该语句假设在第 1 个模拟时刻 *cnt* 变量的赋值情况,其中 *cnt* 变量所带的后缀“0_0”和“0_1”分别表示时间化身和空间化身[3],模拟周期从 0 开始计数.

对 case 语句构成的不完全多路选择器,解决方法是为 case 条件表达式的所有入口构造一个新的整数,而为缺失的入口构造一个新布尔变量.例如,对图 2 中 case 判断结点,假定模拟时刻是 0,其建模方法如图 3 所示.图 3(a)为 RTL 描述,图 3(b)为建模后的 GNU Prolog 程序.

```
case ({put, get}) /* fifo.v:34 */
 2'b10:
    if (cnt<8)
      begin
        cnt <= cnt+2;  /* fifo.v:39 */
      end
 2'b01:
    if (cnt>0)
      begin
        cnt <=  cnt-1;  /* fifo.v:47 */
      end
endcase
```

(a)

```
S_0 #= Put_0 * 2 + Get_0,
#\((S_0 #= 1) #\/ (S_0 #= 2)) #<=> Def_0,
(Def_0 #= 1) #==> Cnt_0_1 #= Cnt_0_0
(S_0 #= 2) #==> ((Cnt_0_0 #< 8) #==> Cnt_0_1 #= Cnt_0_0 + 2),
(S_0 #= 2) #==> ((Cnt_0_0 #>= 8) #==> Cnt_0_1 #= Cnt_0_0),
(S_0 #= 1) #==> ((Cnt_0_0 #> 0) #==> Cnt_0_1 #= Cnt_0_0 - 1),
(S_0 #= 1) #==> ((Cnt_0_0 #=< 0) #==> Cnt_0_1 #= Cnt_0_0),
```

(b)

Fig.3 Constraints for incomplete case statement

图 3 Case 语句不完全多路选择器约束建模

另一种特殊的多路选择器是 one-hot 选择器,每一时刻选择信号中只有一位是活跃的.对 one-hot 多路选择器,解决方法是为选择信号的每一位进行建模.例如,假设一个 n 位信号构成的 one-hot 多路选择器,信号的各位为 $\{s_1,s_2,\ldots,s_n\}$,n 个数据输入为 $\{D_1,D_2,\ldots,D_n\}$,输出为 Z,则生成的 CLP 约束为 $\{(s_i\#=1)\#==>Z\#=D_i,\forall 0\le i\le n\}$.

对循环语句,则需要指定循环展开的次数.根据循环展开次数,先将 Verilog 循环语句转化成等价的条件语句组,再根据转化后的 Verilog 程序生成信号变量的 DD 模型.循环语句的展开分为两种类型:精确展开和模糊展开.精确展开指的是将循环语句恰好展开指定的次数,而模糊展开指的是至多展开指定的次数.Verilog 循环语句的两种等价转化如图 4 所示.

3.2 DD模型约束建模方法

为了将初始输入赋值的影响传播到断言所使用的变量上,使断言被违反(串行断言或并发断言蕴涵式后件)或产生符合断言要求(并发断言蕴涵式前件)的事件序列,需要使用包含控制和数据依赖信息的变量的 DD 模型.根据 DD 模型生成约束模型时,为了满足对不完全多路选择器的考虑,从 DD 模型的根结点开始,逐步生成 CLP 约束,最后形成变量的约束描述.算法如图 5 所示.

```
while (expression) // Expanded n times
begin
  loop-body;
end
```

⟶

```
if(expression)  //1
begin
  loop-body;
  if(expression)//2
  begin
    loop-body;
      ...
      if(expression)//n
      begin
        loop-body;
      end
      ...
  end
end
```

Fig.4 Equivalent transform for Verilog loop statement

图 4 Verilog 循环语句的等价转换方法

```
GenCLP()
{
 if(is leaf node)
 {
  generate CLP statements for node;
  return;
 }
 else if(is if-else node)
 {
  call GenCLP of child nodes;
  save returned CLP statements;
  if(child nodes < 2)
   generate CLP statement for default else;
 }
 else if(is case node)
 {
  if(child nodes < desired number)
  {
   generate CLP for incomplete case node;
   call GenCLP of all the child nodes;
   save the CLP statements of all the child nodes;
  }
  else
  {
    call GenCLP of all the child nodes;
    save the CLP statements of all the child nodes;
  }
 }
 generate CLP statements for current node;
 combine CLP statements of child nodes;
 return;
}
```

Fig.5 Constraints generation based on DD model

图 5 基于 DD 模型的约束生成方法

为了将中间信号变量的值传播到初始输入信号,在分析 Verilog 设计描述时,将为设计中所有的中间信号、初始输入建立 DD 模型.在基于中间信号的 DD 模型生成 CLP 约束时,如果其 DD 模型中各种结点所使用的信号变量都为初始输入信号,则生成过程结束.例如,为图 2 中 *cnt* 信号的 DD 模型生成 CLP 约束时,所有涉及到的信号都为初始输入,此时只要生成 *cnt* 信号的 DD 模型的 CLP 约束即可.

如果信号的 DD 模型赋值结点和控制节点所使用的信号变量名有中间信号变量,假设所使用的中间信号集合为 V.对 V 中的每一个中间信号 v,遍历为信号 v 生成 CLP 约束.如果在该过程中,信号 v 的 DD 模型结点中仍有中间信号变量,则将这些中间信号变量添加到集合 V.该过程将持续直到集合 V 为空.

3.3 约束求解和模拟矢量生成

按照前述模拟矢量自动生成步骤,对待测断言进行迭代展开,直到找到模拟矢量,或达到预设限制停止.对本文前述例子,扩展了前 3 个周期后,GNU Prolog 没有返回解,当扩展到第 4 个周期时,GNU Prolog 返回一个解:[1,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0].CLP 程序中各变量都是与时间化身和空间化身关联的,前 4 个结果表示第 1 个周期 fifo_clr_n,fifo_reset_n 和 put 信号输入为"1",get 信号输入为"0",后面周期的输入值依此类推.

我们的方法在生成模拟矢量时,假定设计的初始状态各信号为"*x*",则上述为 fifo 设计生成的模拟矢量没有包含设计初始化的模拟矢量.初始化模拟矢量可由设计者提供,或由设计者给出提示(例如采用文献[18]的基于路径的生成方法),为初始化语句生成模拟矢量,再与本文方法生成的模拟矢量一起构成最后的模拟矢量.底层的约束求解能力受变量数和约束数影响,能求解的约束规模受所采用的约束求解算法的能力制约[20],GNU Prolog 实现了大部分优化后的约束求解算法,求解能力较强.通过基于程序切片的设计化简技术,使本文的方法能处理大规模的设计,如 Decoder 设计.

3.4 加速技术

在实际设计中,为了生成满足断言的模拟矢量,需要将上述过程迭代很多次才能获得断言所监控的信号值.为了加速模拟矢量生成过程,每次生成过程不需要从设计的初始状态开始.可由设计者指定一些信号的初始值,以加快迭代过程.或与动态随机模拟相结合,先对设计进行模拟,此时模拟矢量可以是随机生成或手工生成的,在达到一定模拟程度以后,以模拟结束时的电路状态为初始状态,以未覆盖的断言为目标,应用本文的方法生成模拟矢量.

4 实验结果

我们已经初步实现了基于断言的模拟矢量自动生成的原型系统.在该系统上,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.实验电路的属性见表 1.表中 Line 表示设计的 Verilog 描述行数,PI 为初始输入数,Version 为该设计的版本数,不同的版本是对正确设计的一个修改,针对设计需满足的属性,在正确设计中引入了错误,使属性被违反.对每个版本的错误,我们都增加了一条断言用于监测错误.

Table 1 Design characteristic

表 1 实验电路属性

Name	Line	PI	Version
Address decoder	52	7	2
Traffic light controller	71	3	3
Arbiter	303	69	2
Clock	719	7	3
Decoder	2 092	14	1

表 1 中地址译码器需要监控的两个属性是:1) 对任意选择的内存地址,能成功地写入数据;2) 在同一时刻不可能有两个地址被选中.仲裁器设计需满足的属性是:1) 总线选择信号是 one-hot 的;2) 对总线的请求总能在一定等待时间内获得总线访问权.闹钟设计的 3 个属性是:1) 时钟在到达"23:59"后会重置为"00:00";2) 在时钟被开启后,一定会走到一个时刻,使小时字段显示"1";3) 小时字段不会变成"24".交通控制灯的 3 个属性是:1) 不会同时有两个绿灯;2) 等待在红灯上的车辆不会无限期等下去;3) 其他一些正确操作.Decoder 设计是一个 32 位微处理器核流水线译码器的 RTL 级设计,在设计过程中,发现该模块在处理中断时与体系结构描述不符,我们直接使用存在问题的描述进行实验.在 Decoder 设计中,直接加入了监测中断处理的断言.

针对上述 4 个实验电路的不同版本中引入的错误和增加的断言,以断言为目标进行了模拟矢量生成,以激活监测引入的错误断言.为了与基于路径的方法进行查错能力和效率比较,我们实现了文献[3]中的方法[18],该方法将路径信息转化为 CLP 约束,通过求解 CLP 约束生成模拟矢量来执行所选路径.实验结果见表 2.两种方法的实验平台配置是一样的.表 2 中第 2~6 列分别是本文方法在生成约束时的扩展周期数和检测到的设计错误数以及生成模拟矢量所用的时间,时间是生成各断言的模拟矢量的时间总和.第 7、8 列是基于路径的方法的扩展

周期数和检测设计错误数.第 9 列为基于路径覆盖方法的约束求解时间.在实验中,两种方法的扩展周期数相同,基于路径的方法中枚举的路径是随机生成和用户指定相结合产生的.

Table 2 Experimental results compared with path based test generation method

表 2 与基于路径方法的比较

Name	Assertion based					Path based		
	Cycle	Find all		Find one		Cycle	Detected	Time (s)
		Detected	Time (s)	Detected	Time (s)			
Address decoder	10	2	0.02	2	0.01	10	1	1.02
Traffic light controller	30	3	3.01	3	0.01	30	1	1.55
Arbiter	11	2	0.01	2	0.01	11	0	0.98
Clock	60	3	4.17	3	0.03	60	1	2.71
Decoder	4	1	0.03	1	0.01	4	0	1.40

在实验中,Decoder 设计在切片化简后得到的与处理中断相关的设计描述只有 200 多行.对闹钟设计,为了加速模拟矢量生成过程,我们采用了加速技术.例如,对第 1 个属性,通过指定闹钟当前时刻为“23:59”,从该状态开始搜索,能在 60 个模拟周期内找到使断言违反的模拟矢量,而不需要从“00:00”开始搜索.

产生 CLP 约束程序时分为两种类型,一种是只找一组解,另一种是找出所有的解.在实验中,当不断增大扩展周期以检测更多的错误时,只找到一组解的 CLP 约束程序都返回了结果,而没有达到资源限制(GNU Prolog 缺省的 8MB 内存).而对找到所有解的 CLP 约束程序,内存为 32MB 时,都找到了所有解.

从表 2 的实验结果可以看出,基于断言的方法在查错效率上比基于路径的方法要高.这是由于基于断言的方法生成模拟矢量目标更明确,在模拟矢量生成时更多地考虑了设计的功能.而基于路径的方法更多地考虑了设计描述的程序结构.同时,设计化简能获得更快的约束求解和模拟矢量生成速度.

我们将本文的方法与随机生成模拟矢量方法进行了比较,实验结果见表 3.实验中,在正确的设计中插入检测属性的断言后,用本文方法生成的模拟矢量(表 2 的结果)与随机生成的模拟矢量比较对插入断言的触发能力.表中 Length 为生成的模拟矢量长度,Trigger 表示模拟时被触发的断言.对不同设计设置了不同的模拟矢量长度上限.从表 3 的实验结果可以看出,本文的方法比随机生成方法目标更明确,生成的模拟矢量长度短,但是能触发的断言多(功能覆盖率高).

Table 3 Experimental results compared with random test generation method

表 3 与随机方法的实验结果比较

Name	Assertion based		Random	
	Length	Triggered	Length	Triggered
Address decoder	10	2	500	2
Traffic light controller	30	3	1 000	2
Arbiter	11	2	1 000	1
Clock	60	3	2 000	2
Decoder	4	1	1 000	0

5 结束语

基于断言的模拟矢量自动生成方法显示了高效的查错能力.该方法生成模拟矢量时采用静态分析技术,对 Verilog 中的模块实例化语句和函数调用的语句无法处理,底层的 GNU Prolog 求解器受约束求解算法自身的影响,在采用了切片化简方法以后,对大规模设计仍存在内存和求解时间爆炸的问题.

将来的研究将主要集中在如何决定扩展周期数,如何与现有模拟矢量自动生成方法结合使用,提高系统的实用性,支持更多的断言描述.结合问题领域的约束求解算法,研究更优的约束求解算法,也是我们将来的主要研究方向.

References:

[1] Cheng K, Krishnakumar AS. Automatic generation of functional vectors using extended finite state machine model. ACM Trans. on Design Automation of Electronic Systems, 1996,1(1):57~79.

[2] Fallah F, Devadas S, Keutzer K. Functional vector generation for HDL models using linear programming and Boolean Satisfiability. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2001,20(8):994~1002.

[3] Venuri R, Kalyanaraman R. Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming. IEEE Trans. on Very Large Scale Integration Systems, 1995,3(2):201~214.

[4] Zeng ZH, Ciesielski M, Rouzeyre B. Functional test generation using constraint logic programming. In: Robert M, Rouzeyre B, Piguet C, Flottes M, eds. Proc. of the 11th Int'l Conf. on Very Large Scale Integration of Systems-on/Chip, SOC Design Methodologies, IFIP TC10/WG10.5. Montpellier: Kluwer, 2001. 133~138.

[5] Ferrandi F, Rendine M, Sciuto D. Functional verification for system C descriptions using constraint solving. In: Kloos CD, ed. Proc. of the 2002 Design, Automation, and Test in Europe. Washington: IEEE Computer Society, 2002. 744~751.

[6] Open Verification Library. 2003. http://www.verificationlib.org/

[7] Beer I, Ben-David S, Eisner C, Fisman D, Gringauze A, Rodeh Y. The temporal logic Sugar. In: Berry G, Comon H, Finkel A, eds. Proc. of the 13th Int'l Conf. on Computer Aided Verification. Paris: Springer-Verlag, 2001. 363~367.

[8] Accellera Organization. Accellera property specification language reference manual, Version 1.01. 2003. http://www.eda.org/vfv/docs/psl_lrm-1.01.pdf

[9] 0-in DA Ltd., 0-in Check. 2003. http://www.0-in.com/products_check.html

[10] @HDL Ltd., @Verifier/@Designer.2003. http://www.athdl.com/products.html

[11] IBM Ltd, RuleBase.2003. http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/index.html

[12] Verplex Ltd., BlackTie. 2003. http://www.verplex.com/products/bt_brochure_PDC.html

[13] Huang CY, Cheng KT. Assertion checking by combined word-level ATPG and modular arithmetic constraint-solving techniques. In: Micheli GD, ed. Proc. of the 37th Design Automation Conf. New York: ACM Press, 2000. 118~123.

[14] Accellera Organization. SystemVerilog 3.1. 2003. http://www.eda.org/sv/SystemVerilog_3.1_final.pdf

[15] Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: MIT Press, 2000. 30~32.

[16] Li T, Guo Y, Li SK. An automatic circuit extractor for HDL description using program slicing. Journal of Computer Science and Technology, 2004,19(5):718~728

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

[18] Li T. Research on techniques of VLSI RT-Level automatic functional vectors generation [Ph.D. Thesis]. ChangSha: National University of Defense Technology, 2003 (in Chinese with English abstract).

[19] Diaz D, Codognet P. The GNU prolog system and its implementation. In: Carroll J, Daminani E, Haddad H, Oppenheim D, eds. Proc. of the 2000 ACM Symp. on Applied Computing. New York: ACM Press, 2000. 728~732.

[20] Kumar V. Algorithms for constraint-satisfaction problems: A survey. AI Magazine, 1992,13(1):32~44.

附中文参考文献:

[18] 李暾.VLSI RTL 级模拟矢量自动生成技术研究[博士学位论文].长沙:国防科学技术大学,2003.