2023（第五届）集成电路 EDA 设计精英挑战赛

赛题指南

一、赛题名称

面向 SystemVerilog Constraints 的通用约束求解器

二、命题单位

上海合见工业软件集团有限公司

三、赛题 Chair

邸志雄(西南交通大学)

四、赛题背景

在数字前端验证阶段，验证工程师需要搭建测试平台（testbench）。测试平台负责产生信号激励，信号激励则驱动待测试设计（DUT，design under test）进行仿真。现在最为流行的产生激励的方法是约束化随机激励。在 SystemVerilog[1]中验证工程师可以根据待测试设计，描写信号间的约束

（Constraints），验证工具中的约束求解器(Constraint Solver)根据这些约束产生信号的随机激励向量。

约束求解器是验证工具中较为复杂的一个部件。它需要支持各种运算符、操作数以及特殊的约束类型。在保证正确性的基本前提下，它需要能够在很短的时间内求解出每个待随机信号的激励向量。

SystemVerilog constraints 定义的问题复杂，对求解器的性能和随机值分布要求很高。当前支持 SystemVerilog constraints 的产品有 Synopsys VCS，Cadence Xcelium，Simens Questa 和合见工软 UVS 等。

在学术领域和开源项目中，还没有直接用于 System constraints 的求解器，但是，通常有两种技术途径可以实现该问题的求解：（1）基于 SAT、SMT、CSP 相关的求解算法，如 MiniSat、Z3、CGecode 等[2,3,4]。（2）基于 BDD（Binary Decision Diagram）[5,6,7,8]求解器，这类求解器在产生均匀随机分布的解上具有优势。

BDD[9,10]是一种可以用来表达布尔函数的数据结构。在形式化验证、电路验证中，许多任务都涉及大型命题逻辑公式的运算 BDD 在这些领域中有着广泛的应用。在约束问题求解中，因为约束

本身就是一系列逻辑、位运算和算术运算组成的表达式，所以也可以通过基于 BDD 的运算来实现。

该题目要求设计实现面向 SystemVerilog constraints 的 BDD 求解器，该求解器根据每条约束生成各自的 BDD，通过交集运算求解得到所有 BDD 的交集，得到最终的 BDD，这个 BDD 对应了整个约束问题的解空间。在这个 BDD 上进行随机采样即可得到符合约束的一组随机值。

本赛题适合计算机、集成电路、数学等相关专业的学生，对于已有 SAT/SMT/CSP 编程经验的学生也更加适合。

五、赛题描述

这里按照 SystemVerilog 的约束描述语法对本次比赛需要支持的问题类型说明如下。

（一）约束问题的定义

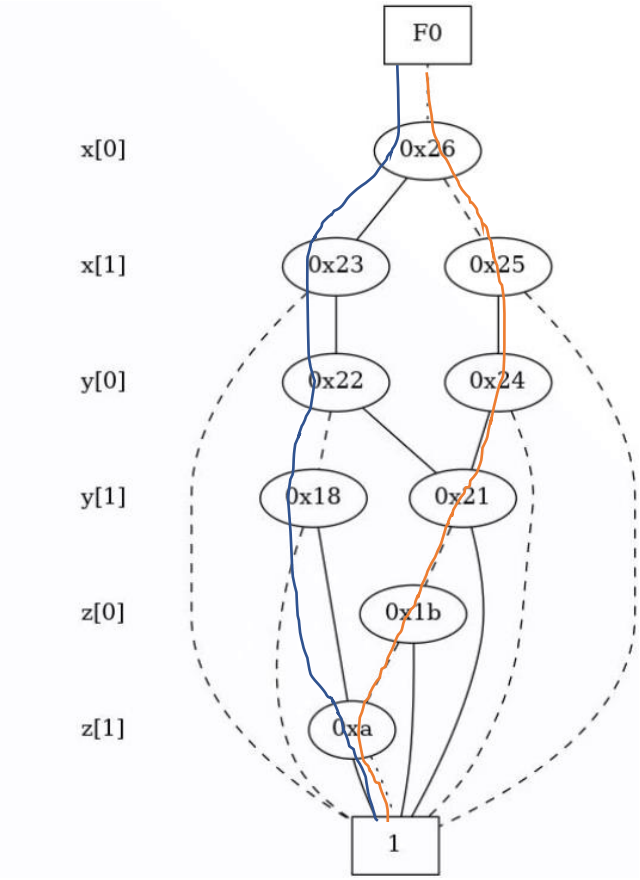
约束问题，由变量、常量和表达式组成，其中

- 变量是有符号(signed)或无符号(unsigned)的、具有固定位宽(bit-width)的位向量(bit-vector)
  - 比如 bit[7:0] x;定义了一个 unsigned 的 8bit 位宽的变量。
- 常量是有符号或无符号、具有固定位宽的整型数
- 表达式由操作数和运算符构成，操作数可以常量或变量，运算符有以下 6 类：
  - 逻辑运算：&&, ||, !
  - 位运算：&, |, ^, ~ …
  - 算术运算：+, -, *, /, %
  - 关系运算: >, <, >=, <=, ==, !=
  - If-then-else 运算

（二）给定一个约束问题，我们可以计算得到代表解空间的 BDD

如，假设给定如下示例约束：

```
bit[1:0] x,y,z;
x> y;
y> z;
```



在上图中，存在 3 种边，实线是 Then 边，表示相应的 bit 取 1，短划线是 Else 边，表示相应的 bit 取 0。点虚线是 complement 边，如果从根节点到叶节点的路径上有奇数个 complement 边，那么这个路径取值为 0，否则取值为 1。叶节点取值为 1 的边对应的每个 bit 的值，即是符合约束的一组随机赋值，该条路径可以称之为合法路径（如上图中的红蓝两条路径）。因此得到 BDD 之后挑选随机值的过程就是从 BDD 的根节点到叶节点自顶向下挑选路径

的过程。为了使产生的随机值能够均匀地覆盖解空间，我们需要保证每组随机值具有同样的选中概率。参考方法为：

计算并标注每个节点的 Then/Else/Complement 边通过的合法路径的数目

计算每个节点的 Then/Else/Complement 边被选中的概率，计算方法是

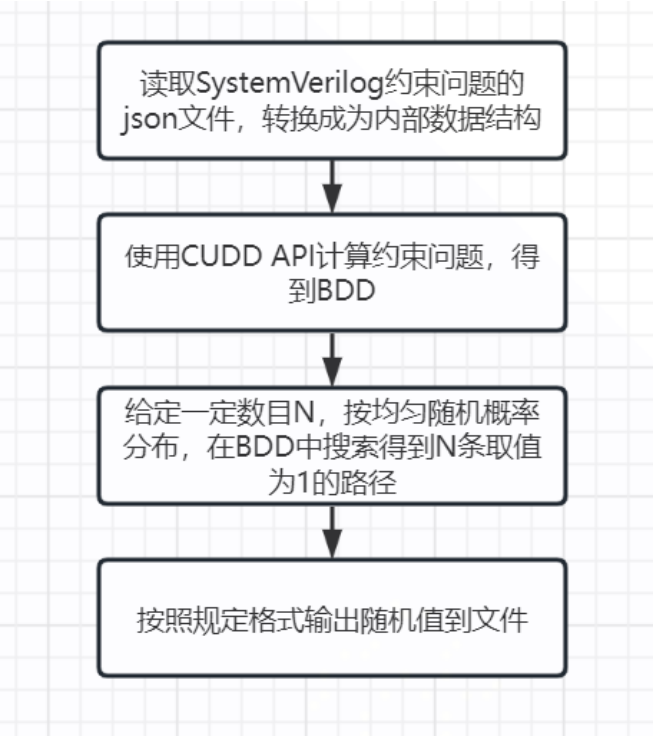
Then 概率 = 通过 Then 边的合法路径数 / 所有通过该节点的路径数

Else 概率 = 通过 Else 边的合法路径数 / 所有通过该节点的路径数

Complement 概率 = 通过 Complement 边的合法路径数 / 所有通过该节点的路径数

每次到达一个节点后，根据计算好的概率选边。

综上所述，整体 BDD 求解器的工作流程如下图：



其中，CUDD 是一个用于计算和处理 BDD/ADD/ZDD 的开源库。使用 CUDD 提供的 API 可以比较方便地从表达式计算得到 BDD，具体的计算过程可以参阅相关的文档。除了 BDD，参赛者也可以选用 ADD/ZDD 完成题目。

由于 SystemVerilog constraints 需要专门的 parser 进行处理才能得到结构化信息，为了避免过多的工作量，我们将会提供已经 parse 过的 json 格式的约束问题。Json 格式会在附件中说明。

六、评分标准

（一）评分说明

本赛题将提供一定数量的测试用例，其中包含了从简单到复杂不同难度的问题。所有的测试用例赛前不对参赛者公开。每个测试用例分配相同的分值，最后根据总分评判参赛者排名。赛前会提供若干测试用例和输出结果的样本，供参赛者练习参考使用。

评分标准包括 3 个部分：

1. 输出结果必须保证正确性；
2. 性能通过程序的执行时间来评判；
3. 输出结果需要符合均匀联合随机分布；

根据以上标准每个测试用例的评分步骤如下，具体细则另行说明：

1. 检查结果正确性，任意一组值错误，本用例得分为 0；
2. 检查结果分布，如随机值分布偏差过大，本用例得分为 0；
3. 通过正确性和分布检查后，根据性能排名评分：假设参赛人数为 N，名次为 M，得分根据以下公式计算

   Score=10*（N-M+1）/N

第一名得 10 分，第二名得(N-1)*10/N……

最后将每个测试用例的分数相加得到总分。

（二）提交说明

本次比赛需要提供二进制文件和脚本文件。所依赖的第三方库也要一并提供。

本赛题要求使用 CUDD3.0.0（源码下载网址：https://davidkebo.com/source/cudd_versions/cudd-3.0.0.tar.gz 教程介绍：The CUDD package, BDD, ADD Tutorial and examples | David Kebo，User's Manual (mit.edu)）.考虑到参赛者可能在编译 CUDD 时可能加入不同的编译参数和选项，CUDD 库也要提供。

输入和输出文件格式见附件。

七、参考资料：

[1] IEEE Std 1800™-2017, IEEE Standard for SystemVerilog— Unified Hardware Design, Specification, and Verification Language, IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group

[2] N. E´en and N. S¨orensson. An extensible SAT-solver [ver 1.2]. In Theory and Applications of Satisfiability Testing, volume 2919 of LNCS, pages 512–518. Springer, 2003.

[3] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference (TACAS), volume 4963 of LNCS, pages 337–340. Springer, 2008.

[4] Gecode: Generic Constraint Development Environment, http://www.gecode.org

[5] C. Barrett, D. Dill, and J. Levitt. Validity checking for combinations of theories with equality. In M. K. Srivas and A. J. Camilleri, editors, Formal Methods in Computer-Aided Design, First International Conference (FMCAD), volume 1166 of LNCS, pages 187–201. Springer, 1996

[6] J. Filliatre, S. Owre, H. Ruess, and N. Shankar. ICS: Integrated canonizer and solver. In 13th International Conference on Computer Aided Verification (CAV), volume 2102 of LNCS, pages 246–249. Springer, 2001

[7] J. Yuan, K. Shultz, C. Pixley, H. Miller and A. Aziz, Modeling design constraints and biasing in simulation using BDDs, 1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051), San Jose, CA, USA, 1999, pp. 584-589, doi: 10.1109/ICCAD.1999.810715.

[8] Jun Yuan, Carl Pixley, Adnan Aziz, Constraint-Based Verification, 2006

[9] R. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, C-35(12):1035–1044, 1986.

[10] Henrik Reif Andersen. An Introduction to Binary Decision Diagrams. Fall, 1999

附件：赛题相关文件说明

比赛程序的输入是 json 格式描述的约束问题文件，输出是N组随机值（N按照每个测试用例的具体情况可以是100到1000不等）。

1、输入文件的格式如下：

```
"variable_list":
[
    {       // variable
            "id": <number>,
            "name": <string>,
            "signed": <true_of_false>,
            "bit_width": <number>
    },
    ...
]

"constraint_list":
[
    {       // expression
            "op": <string> // operator type
            "id": <number> // variable id if op is VAR
            "value": <number> // hexical number is op is CONST
            "lhs_expression": // if op is another type
            {       // lhs expression
                    // same as above expression format
            }
            "rhs_expression": // if op is another type and rhs exists
            {       // rhs expression
                    // same as above expression format
            }
            "if_expression": // if op is MUX(If-Then-Else), if: if_expression, then:
lhs_expression, else: rhs_expression
            {       // pred expression
                    // same as above expression format
            }
    }
]
```

2、输出文件格式如下：

```
"assignment_list":
[
        [       // assignment 1
                {"value": <number>} // hexical number, sorted in asending
order of variable ids
                {"value": <number>}
                ...
        ]
        [       // assignment 2
                ...
        ]
        ...
]
```