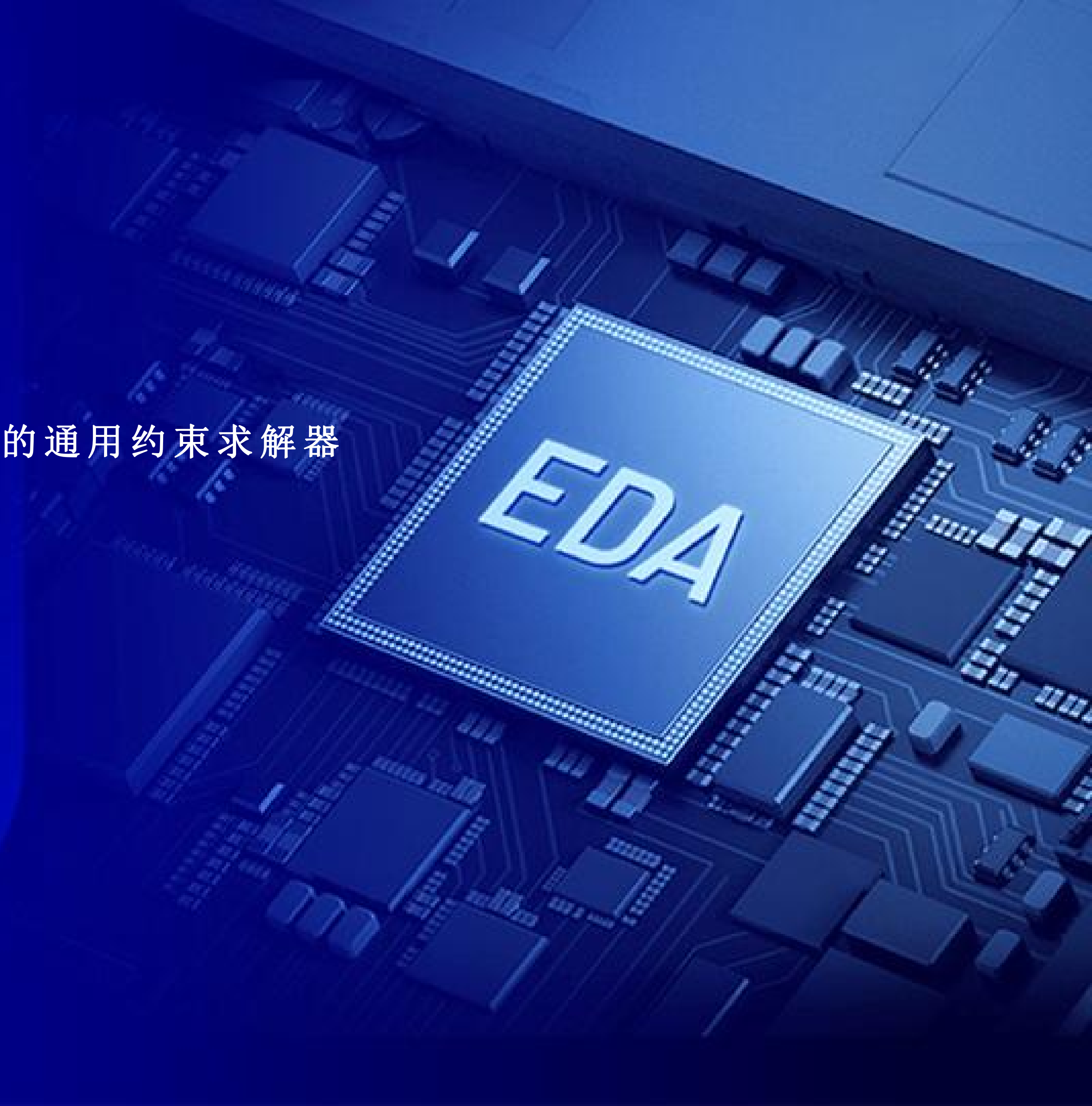
面向SystemVerilog Constraints的通用约束求解器

集成电路 EDA 设计精英挑战赛（合见工软）赛题讲解

倪恩志



上海合见工业软件集团有限公司
Shanghai UniVista Industrial Software Group Co.,Ltd.

CONTENT

目录

背景介绍 – 验证方法

- Formal verification v.s. simulation based verification
- Directed test, constrained random test and UVM

背景介绍 – 求解器简介

- Solver简介
- 类型：SAT, SMT, CSP and BDD-based
- 特性：正确性，随机性和可重现，概率分布，性能要求

赛题详解 – 约束问题定义

约束问题，由变量、常量和表达式组成，其中

- 变量是无符号(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: >, <, >=, <=, ==, !=

赛题详解 – 约束问题示例

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

随机解1：

x = 2’b11

y = 2’b10

z = 2’b01

随机解2：

x = 2’b10

y = 2’b01

z = 2’b00

…

赛题详解 – BDD & CUDD

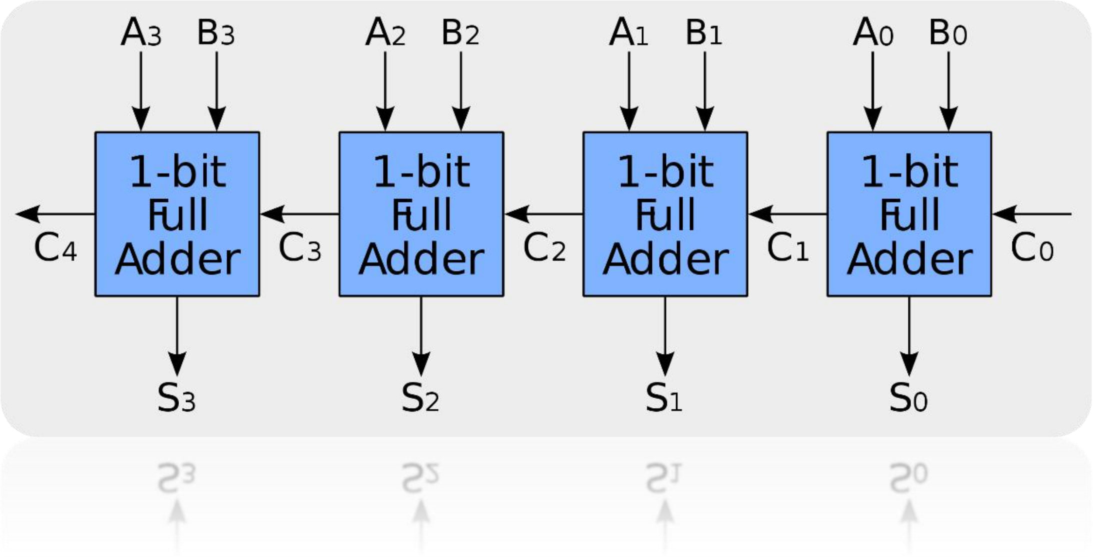
BDD[9,10]是一种可以用来表达布尔函数的数据结构。在形式化验证、电路验证中，许多任务都涉及大型命题逻辑公式的运算，BDD在这些领域中有着广泛的应用。在约束问题求解中，因为约束本身就是一系列逻辑、位运算和算术运算组成的表达式，所以也可以通过基于BDD的运算来实现。

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

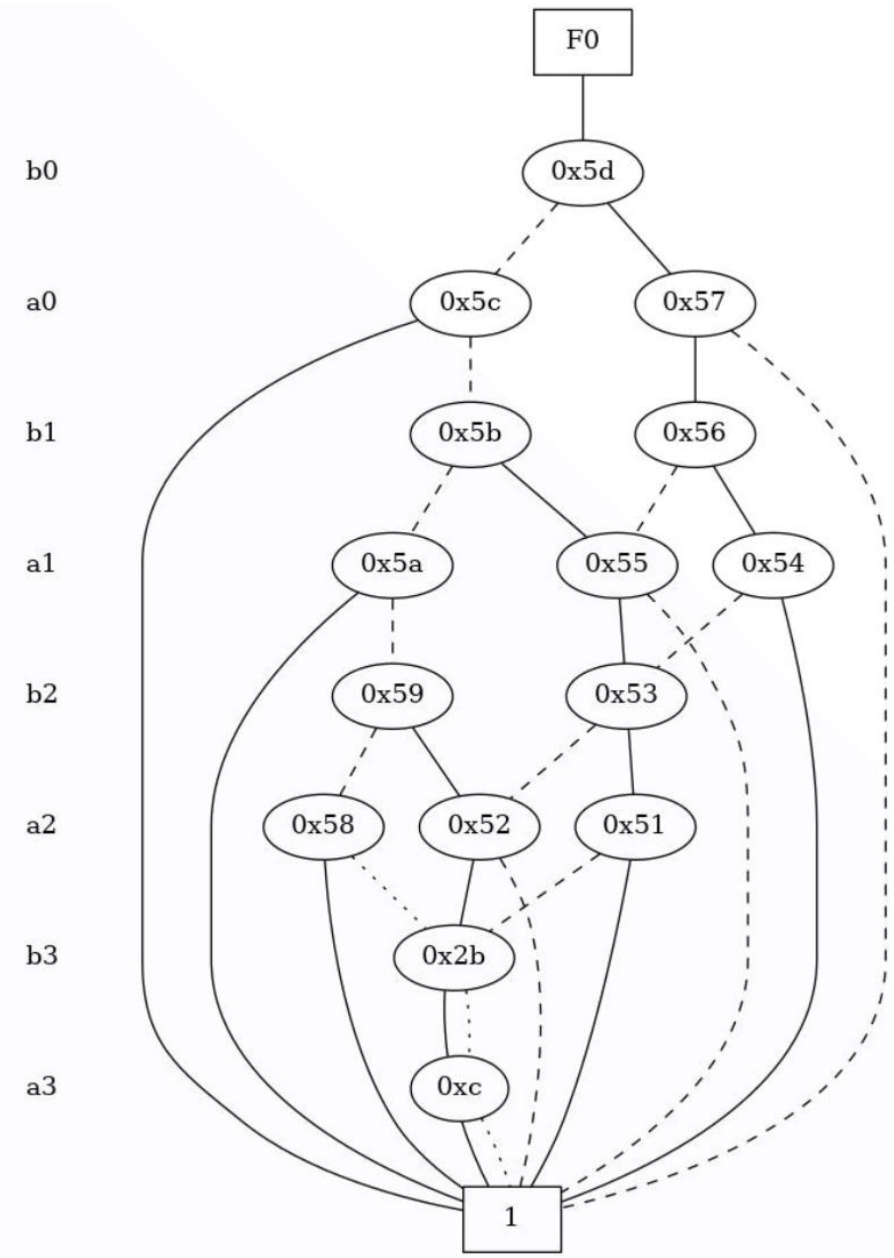
赛题详解 – BDD & CUDD

bit[3:0] A,B;

A + B > 4’b0000



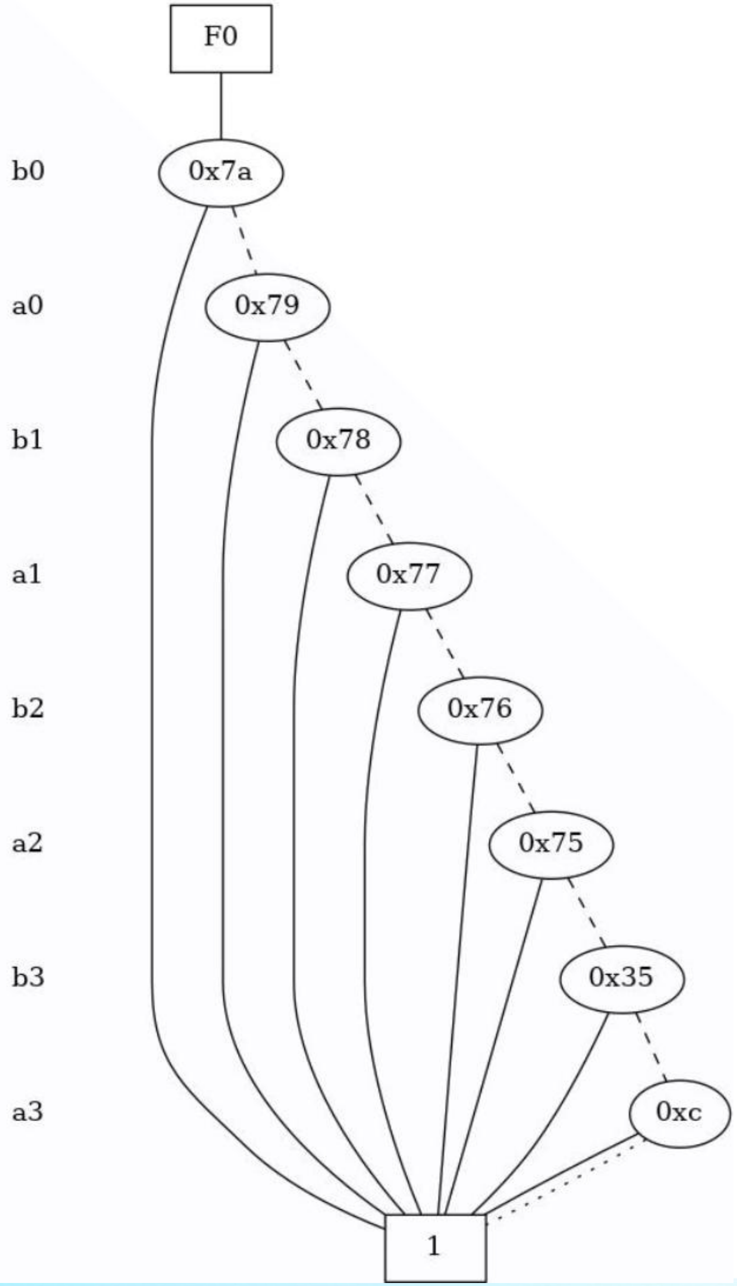
```
#include "cuddObj.hh"
BDD a0, a1, a2, a3, b0, b1, b2, b3; // bddVar
BDD s0, s1, s2, s3;
BDD result;

a0 = bddVar();
...

s0 = a0 ^ b0;
c0 = (a0 & b0);
s1 = a1 ^ b1 ^ c0;
c1 = (a1 & b1) | (c0 & (a1 | b1))
s2 = a2 ^ b2 ^ c1;
c2 = (a2 & b2) | (c1 & (a2 | b2))
s3 = a3 ^ b3 ^ c2;
c3 = (a3 & b3) | (c2 & (a3 | b3))

result = s0.Xor(s1.Xor(s2.Xor(s3)))
```



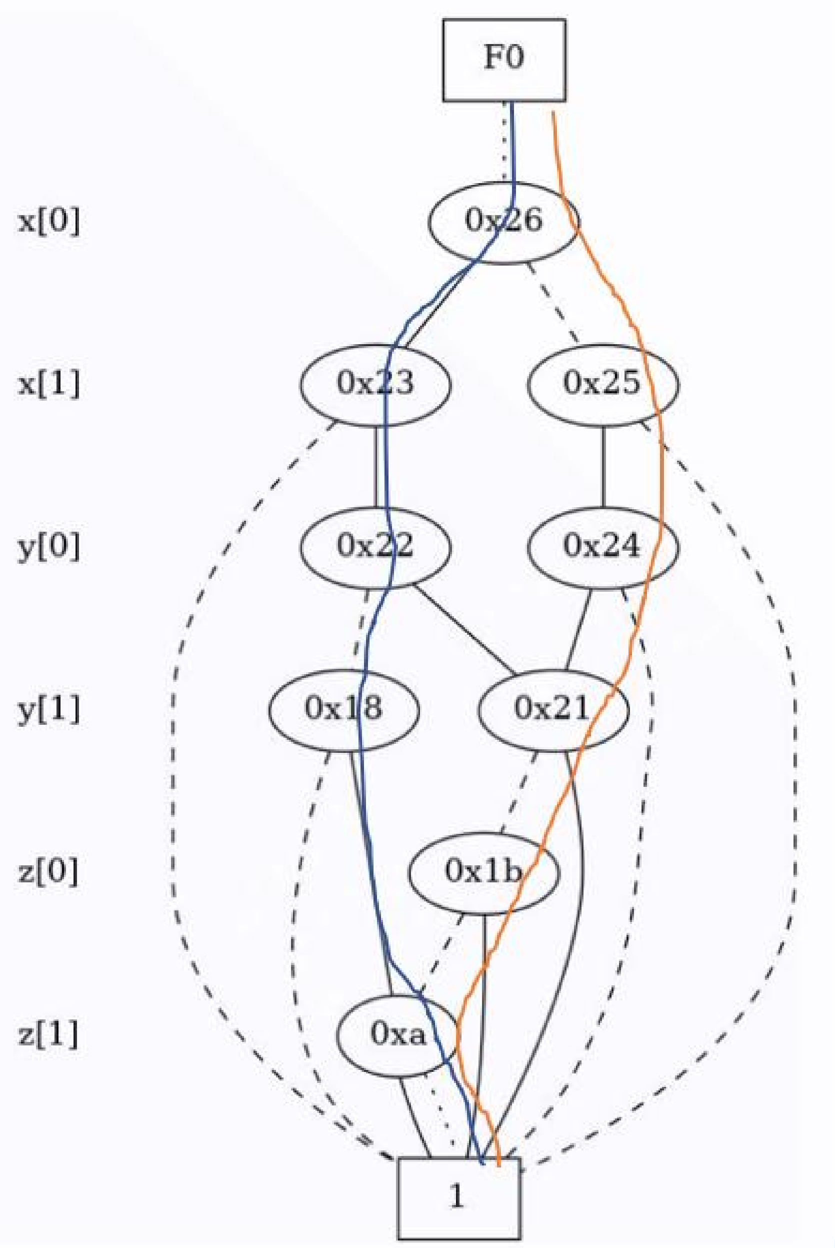
bit[3:0] A,B;

A + B > 0

32’(A) + 32’(B) > unsigned’(0)



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

x > y;

y > z;
```



The CUDD package, BDD, ADD Tutorial and examples | David Kebo

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

bdd.pdf (utexas.edu)

```
"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
				}
	}
]
```

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

Constraint Problem:

```
bit [31 : 0] a;
bit [31 : 0] b;
bit [31 : 0] c;
bit [31 : 0] abj_a;
bit [31 : 0] abj_b;
bit [31 : 0] abj_c;
constraint cb {
(abj_a < 32'h0000064);
(abj_b < 32'h00000064);
(abj_c < 32'h0000064);
(abj_a == a);
(abj_b == b);
(abj_c == c);
((a * b) * c) == 32'h0000000533);
}
```

Constraint Problem in Json:


```
{
"variable_list": [
    { "id": 0, "name": "a", "signed": false, "bit_width": 32 },
    { "id": 1, "name": "b", "signed": false, "bit_width": 32 },
    { "id": 2, "name": "c", "signed": false, "bit_width": 32 },
    { "id": 3, "name": "abj_a", "signed": false, "bit_width": 32 },
    { "id": 4, "name": "abj_b", "signed": false, "bit width": 32 },
    { "id": 5, "name": "abj_c", "signed": false, "bit_width": 32 }
],
"constraint list": [
    {
            "op": "LT",
            "lhs_expression": { "op": "VAR", "id": 3 },
            "rhs_expression": { "op": "CONST", "value": "0000000001234556789012345678901234567890" }
    },
    ...
    {
            "op": "EQ",
            "lhs_expression": {
                    "op": "MUL",
                    "lhs_expression": {
                            "op": "MUL'
                            "lhs_expression": { "op": "VAR", "id": 0 },
                            "rhs_expression": { "op": "VAR", "id": 1 }
                    },
                    "rhs_expression": { "op": "VAR", "id": 2 }
            },
            "rhs_expression": { "op": "CONST", "value": "000001533" }
    }
]
}
```

results in Json:


```
{
"assignment_list":
[
	[
		{"value": "0"},
		{"value": "1"},
		{"value": "2"},
		{"value": "3"},
		{"value": "4"},
		{"value": "5"}
	],
	[
		{"value": "5"},
		{"value": "4"},
		{"value": "3"},
		{"value": "2"},
		{"value": "1"},
		{"value": "0"}
	],
	[
		{"value": "5"},
		{"value": "4"},
		{"value": "3"},
		{"value": "2"},
		{"value": "1"},
		{"value": "0"}
	]
}
```

赛题详解 – 任务分解

1.读取问题文件，存储到内部数据结构

2. 精度转换

3.遍历表达式，使用CUDD API构造BDD

4.基于BDD，挑选合法路径

5.输出随机解到文件

评分标准

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。

References

[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.

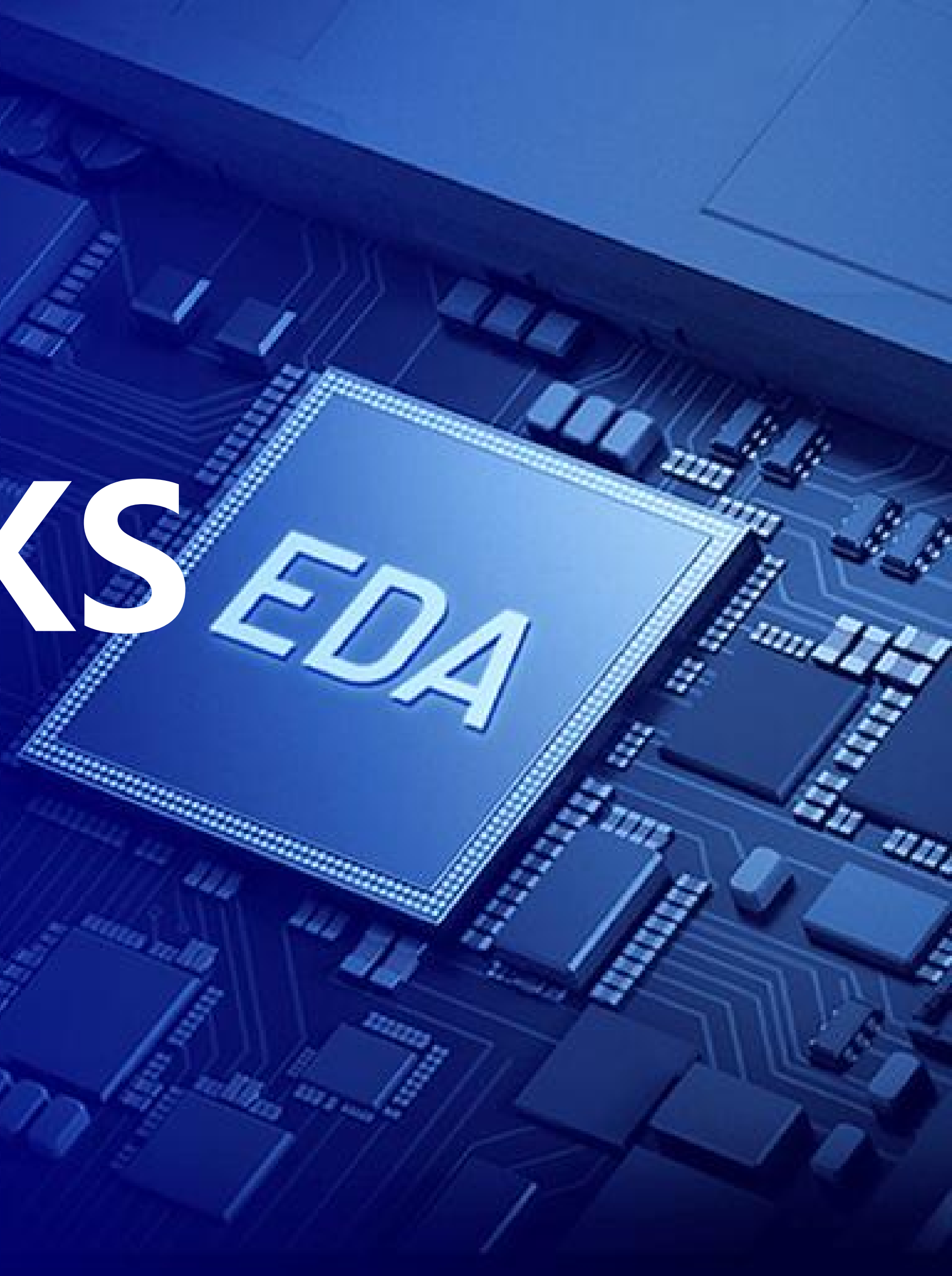
[8] Jun Yuan, Carl Pixley, Adnan Aziz, Constraint-Based Verification, 2006

[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

合 见 工 软

THANKS



上海合见工业软件集团有限公司
Shanghai UniVista Industrial Software Group Co.,Ltd.