2023（第五届）集成电路EDA设计精英挑战赛

赛题指南

版本	时间	修订内容
V1.0.0	2023-08-18	初版
V1.0.1	2023-09-06	打榜赛题服务器配置更新
V1.0.2	2024-01-31	打榜赛题服务器配置更新

一、 赛题题目

基于ARM多核CPU架构的故障仿真并行加速

二、 命题企业

海思半导体有限公司

三、 赛题Chair

黄宇 华为半导体领域科学家 海思EDA首席架构师

四、 赛题背景

故障仿真（Fault Simulation）是电路测试和诊断的重要步骤，是DFT（Design for Test）的重要组成部分，主要用于评估电路运行过程中可能出现的故障。故障仿真在测试向量生产、故障诊断、生产故障分析等环节中都发挥了至关重要的作用，确保

芯片设计稳定可靠的同时，为芯片制造过程提供了必要的支持。

国外EDA厂商（如Synopsys、Siemens EDA等）在故障仿真领域积累了丰富的经验和专业技术，设计了多种故障模型和测试方法，支持的芯片类型、规模和功能也非常广泛，如Stuck-At故障仿真、扫描链故障仿真、门延迟故障仿真、路径延迟故障仿真、模拟故障仿真等。

(一) 故障仿真流程介绍

芯片在制造过程中，总会受到各种不确定因素的影响。比如，环境干扰、硅片质量不佳、机台设置偏差、工程师操作失误等等，我们实际制造出来的芯片总会存在各种缺陷。

如图1所示，如果电路的e点对电源短路，我们可以将这一电路缺陷抽象为名为Stuck-At-1的故障模型。电路表现为不论a，b的输入如何变化，e点电平始终为1（即高电平）。在芯片量产测试的ATE机台上，为了检测到这个芯片内部的缺陷，我们只能通过对输入a，b施加特定的激励，捕获并检测输出g的电平与预期的电平是否一致，以推断电路是否存在缺陷（可能是由于e点的Stuck-At-1故障导致）。举个例子，当电路的e点出现对电源短路故障时，如果对a，b施加激励（a = 0，b = 1），此时，g的预期输出为0。但是，由于e点存在类型为stuck-at-1的fault，因此实际输出为1。因此，我们称e点的Stuck-At-1 fault能被pattern

（a = 0，b = 1）检测到。

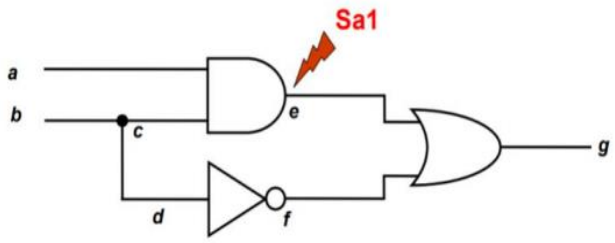

图1

本赛题仅针对组合电路的Stuck-At故障模型进行故障仿真。由于电路中的每一个点都有可能存在fault，我们可以通过解析电路网表得到完整的fault集合，称为fault list。本赛题中故障仿真的输入netlist、fault list、test patterns均为用例外部文件输入。图2是故障仿真的整体流程。

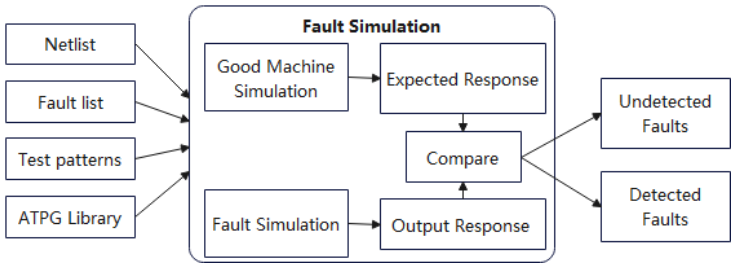

图2

赛题提供的Case已打包了故障仿真所需的输入和运行脚本，可直接运行用例。需要说明的是：

- ATPG Library会由打包Case中提供；
- Fault list为FAN_ATPG自动插入；
- Pattern的格式为FAN_ATPG私有自定义.pat文件格式，如下所示：

```
G0 G1 G2 G3  |
U_G5 U_G6 U_G7  |
G17
BASIC_SCAN
_num_of_pattern_5
_pattern_1 0000 |  | 011 |  | 0 |  | 011
_pattern_2 0111 |  | 000 |  | 1 |  | 000
_pattern_3 1010 |  | 010 |  | 1 |  | 100
_pattern_4 1011 |  | 000 |  | 0 |  | 010
_pattern_5 0001 |  | 110 |  | 1 |  | 000
```

- 导出Undetected Faults和Detected Faults格式如下所示：

```
#    type    code    pin (cell)
#    ----    ----    ------------------
#    SA0      UD     U_G6/QN (SDFF_X1)
#    SA1      UD     U_G6/QN (SDFF_X1)
#    SA0      UD     U_G7/QN (SDFF_X1)

#    type    code    pin (cell)
#    ----    ----    ------------------
#    SA0      DT     G0 (primary input)
#    SA1      DT     G0 (primary input)
#    SA0      DT     G1 (primary input)
```

(二) 故障仿真面临的挑战

内存挑战：芯片工艺在微米级、纳米级集成之后，设计中的逻辑门数量、Scan Chain数量、Scan Cell数量、测试点连接数量等都呈现出大幅增长、庞杂化的趋势。这就使得传统的故障仿真方法所需内存随之增加，需要采用新的仿真平台和算法来满足内存需求。

Run Time挑战：在硬件实现中，复杂电路的大小远远超过计算机的容量，软件实现的仿真性能比硬件还要低。当今电路设计中需要进行的Test Patterns数量较多，因此仿真过程需要大量的计算资源，这也是故障仿真面对的一个重要挑战。

(三) 常见的故障仿真硬件并行加速方法

并行计算是一种有效的方法，可以在短时间内完成大规模、高复杂度电路的测试和诊断过程。硬件并行加速故障仿真主要采用以下几种方式：

1. GPU并行加速：GPU具有高并行计算能力，可以同时处理大量数据，可有效加快故障仿真的计算速度。GPU的价格昂贵且功耗较高，需要使用特殊的编程模型（如CUDA），存在编程复杂性高和移植性差的问题；
2. 分布式并行：相对于传统的集中式仿真，分布式故障仿真允许仿真任务在多个计算机或服务器上分配和并行运行，以提高仿真效率和性能。但需要考虑到多个节点之间的数据通信和同步开销等问题。
3. 多核CPU并行：利用现代多核处理器的强大计算能力，将故障仿真分割为多个独立的任务并分配给不同的CPU内核。多核并行计算具有效率高、成本低等优点，且易于实现，是目前大规模设计测试中应用最广泛的手段。

（四） 赛题采用多核CPU并行加速方式

本赛题将采用多核CPU并行的方式，并在鲲鹏ARM多核CPU架构服务器进行打榜。打榜服务器CPU为Huawei Kunpeng920,主频2.6GHz,32vCPUs，内存50GiB。参赛队伍需要设计合理的并行算法方案，以减少仿真端到端的时间和提高多核并行的

Scalability。

五、 赛题描述:

(一) 描述

本赛题需要参赛队伍在开源项目(FAN_ATPG)提供的一系列的命令接口基础上开发多核并行故障仿真软件,本赛题打榜关注端到端的运行时间,参赛队伍应关注整个阶段的效率。提交的作品需保证命令接口与开源项目保持一致,以确保用例能正常运行。

参赛队伍可以直接在 FAN_ATPG 项目上进行并行版本的开发,也可以接入自有的其他故障仿真模型,但接口需要与 FAN_ATPG 项目保持一致。参赛队伍需要分别提供一个单线程版本和一个多线程版本的故障仿真软件,所以需要关注并行化算法以获得更高的多核并行加速比,可参见评分章节说明。

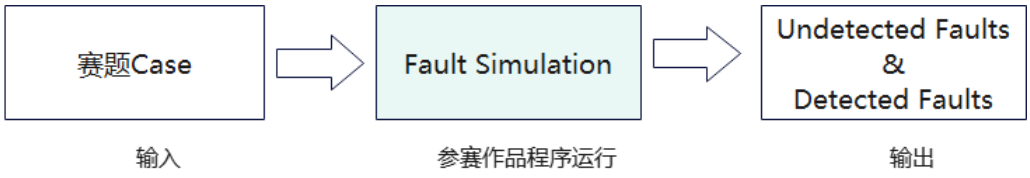

图 3

- 赛题 Case

赛题 Case 为系统输入之一,该部分由出题方提供。开源项目中亦提供了一些用例,参赛队伍也可自行构造 Case, 便于验证和优化自己的算法和系统。

典型 Case 脚本如下:

```
read_lib techlib/mod_nangate45.mdt
read_netlist mod_netlist/s5378.v
report_netlist
build_circuit --frame 1
report_circuit
read_pattern pat/FAN_s5378.pat
report_pattern
set_fault_type saf
add_fault -a
run_fault_sim
report_fault -s UD > rpt/s5378_fsim_ud.rpt
report_fault -s DT > rpt/s5378_fsim_dt.rpt
exit
```

用例初始化

故障仿真运行

运行结果导出

- Fault Simulation

Fault Simulation 的并行加速是本赛题的核心任务。故障仿真多核并行加速比是打榜排名的最重要指标，参赛队伍应该设计合理的并行化算法，充分发挥 Kunpeng920 多核的性能。

- Undetected Faults & Detected Faults

参赛作品必须保证 Fault Simulation 的正确性。用例会通过命令接口导出 Undetected Faults 和 Detected Faults，并检查其正确性。

Fault Simulation 基础源代码及用例等相关资源可参见下列代码仓库。关于算法和系统的优化思路，可参考 Q&A。

https://github.com/NTU-LaDS-II/FAN_ATPG

(二) 竞赛规则

竞赛流程

下面为该课题竞赛流程：

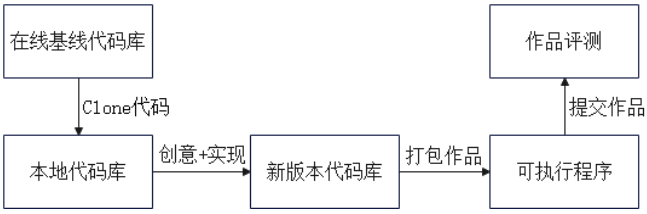

图 4

各参赛队伍需要以赛题提供的在线代码库为接口，通过自己的知识、技能、实践和创意来实现一个新的并行化故障仿真软件。

为尽量排除环境干扰，在作品评测阶段，会连续运行三次，取平均成绩。

- 源代码

推荐参赛队伍以开源的源代码为基础进行并行化开发；参赛队伍也可以在不改变运行接口的前提下使用其他故障仿真模型。

- 竞赛 Case

本赛题会提供两组竞赛Case，一组是公开的用于日常调测，另一组是非公开的用于竞赛打榜。可参见后文的"评分标准"章节。

- 接口

请仔细阅读开源代码的README，请勿随意修改系统的接口。整个基础代码里面接口有两类：一类是源代码算法接口；另一类是系统环境搭建脚本；README 里面有详述。

- 环境

构建环境和运行环境采用的系统版本一致。建议各参赛队伍也在该环境下开发和测试。具体的构建和运行环境信息，以及各构建工具链、依赖组件版本等信息均可参见源码仓库的README。

六、 评分标准:

本赛题关注多核并行的加速比，按加速比进行打榜比赛。参赛队伍提交的作品包含一个单线程版本和一个多线程版本；这两个版本将运行同一组打榜用例，系统将记录每个用例的故障仿真阶段 Run Time 时间和内存占用峰值。具体打榜分数的计算和排名规则如下：

(一) 单个用例的加速比 = （单线程版本 Run Time / 多线程版本 Run Time） * 因子，因子大小与用例 Fault 数量级正相关；

(二) 累加所有用例的加速比作为打榜分数，分数从大到小排序，分数相同的情况下，所有多线程版本用例的内存占用峰值累加值小的排前面；

(三) FAN_ATPG 原始代码的 Run Time 作为基线，如果提交作品 Run Time 大于基线的则按 FAN_ATPG 原始代码的 Run Time 作为计算。反之，则按提交作品的实际 Run Time 计算。

(四) 对于某个用例的 Fault 输出不正确的情况,按 FAN_ATPG 原始代码的基线 Run Time 计算。

评分举例：

队伍1						
	Run time 单位(s)			因子	加速比	多线程 内存峰值 单位(M)
	基线	单线程	多线程			
用例1（Fault数量10万）	150	120	30	1	4	400
用例2（Fault数量20万）	320	300	100	2	6	800
用例3（Fault数量50万）	1000	800	320	5	12.5	2000
总计	-	-	-	-	22.5	3200

说明：队伍 1 单线程版本 Run Time 小于基线的 Run Time，则按单线程版本 Run Time 计算加速比，加速比总计 22.5，内存总计 3200M；

队伍2						
	Run time 单位(s)			因子	加速比	多线程 内存峰值 单位(M)
	基线	单线程	多线程			
用例1（Fault数量10万）	150	140	40	1	3.5	500
用例2（Fault数量20万）	320	315	84	2	7.5	1000
用例3（Fault数量50万）	1000	759	330	5	11.5	2600
总计	-	-	-	-	22.5	4100

说明：队伍 2 单线程版本 Run Time 小于基线的 Run Time，则按单线程版本 Run Time 计算加速比，加速比总计 22.5，内存总计 4100M；

队伍3						
	Run time 单位(s)			因子	加速比	多线程 内存峰值 单位(M)
	基线	单线程	多线程			
用例1（Fault数量10万）	150	180	50	1	3	380
用例2（Fault数量20万）	320	290	145	2	4	720
用例3（Fault数量50万）	1000	1200	666	5	7.5	1900
总计	-	-	-	-	14.5	3000

说明：队伍 3 部分单线程版本 Run Time 大于基线的 Run Time，则按基线的 Run Time 计算加速比，加速比总计 15.5，内存总计 3000M；

队伍 1 与队伍 2 加速比虽然相同，但队伍 1 峰值内存总计小于队伍 2，因此队伍 1 排在队伍 2 前面；队伍 3 加速比小于队伍 1 和队伍 2，因此排在最后。最终排名如下：

第一名：队伍 1

第二名：队伍 2

第三名：队伍 3

七、 声明

获胜者所提交的二进制文件（包括相关的说明文档等）将在比赛结束后删除，本次比赛不做任何形式的保留。所有版权归获胜者所有。所有参赛者的二进制文件需要符合 MIT license。本次比赛所涉及的所有比赛题目也将在比赛后删除，不做任何形式的保留。所有参赛者提供的二进制文件、文档以及本次比赛的题目仅供本次比赛使用，不做任何形式的商用。如果比赛结束后有任何知识产权争议，将交由对应的委员会进行裁决。

八、参考资料

[1] Roth J P, Bouricius W G, Schneider P R. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits[J]. IEEE Transactions on Electronic Computers, 1967 (5): 567-580.

[2] Hu J, Dai G, Wang L, et al. Adaptive Multi-Dimensional Parallel Fault Simulation Framework on Heterogeneous System[J]. IEEE Transactions on Computer-Aided

Design of Integrated Circuits and Systems, 2022.

[3] Li, Min , and M. S. Hsiao . "FSimGP^2: An Efficient Fault Simulator with GPGPU." IEEE IEEE, 2010:15-20.

[4] Markas T, Royals M, Kanopoulos N. On distributed fault simulation[J]. Computer, 1990, 23(1): 40-52.

[5] Waicukauski, J. A. , et al. "Fault simulation for structured VLSI." vlsi system design (1985).