第二届（2020年）集成电路EDA设计精英挑战赛
赛题指南

一、赛题一：基于虚拟原型平台探索软硬件协同设计

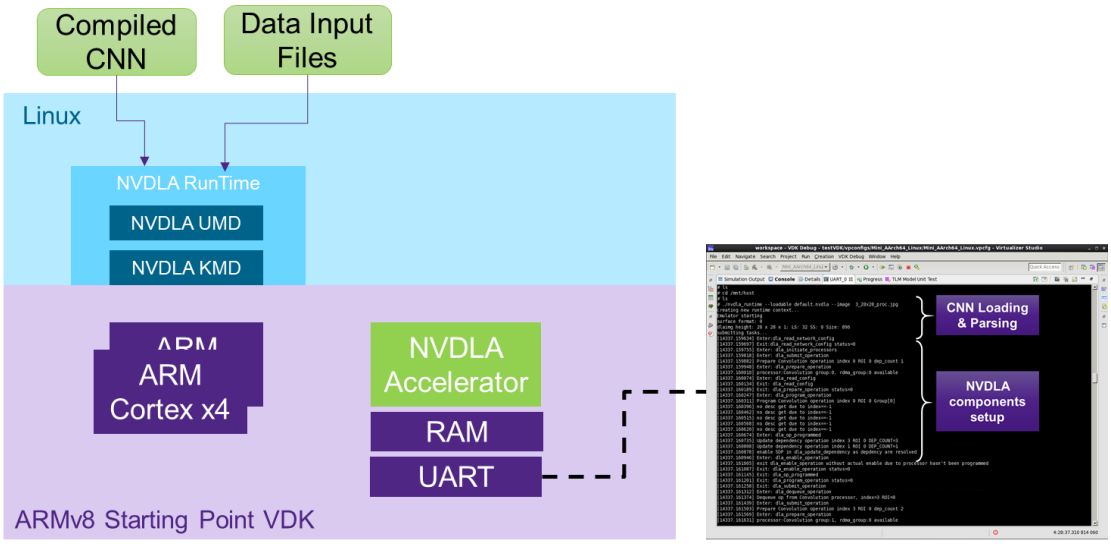
二、命题单位：新思科技（Synopsys）

三、赛题背景：

现代soc集成了越来越多的功能，不仅在硬件，而且在软件方面，都变得越来越复杂。在硬件就绪后开发软件在业界已不再高效和具有竞争力。特别是在5G、人工智能、汽车和应用处理器等高速发展的垂直市场上，迫切需要采用虚拟原型技术并行设计软硬件。

四、赛题描述：

Synopsys的虚拟原型解决方案是虚拟开发工具包(VDK)。它是一种基于IEEE SystemC语言的transaction级别仿真技术，能够运行实际的二进制软件image。我们会提供一个参考的AI-VDK，它由几个Arm Cortex A55s、GIC500、DDR&SRAM存储器、UART和Nvidia的开源AI加速器模型NVDLA-AI组成(www.nvdla.org). 这个VDK可以在不到一分钟的时间内启动Linaro Linux,并提供已经移植好的CNN网络示例,如AlexNet和ResNet。竞赛范围将以AI VDK为基础，利用现代仿真技术在该平台上运行复杂的人工智能网络，并实现仿真加速。对于每一项加速方法，速度提升快慢为衡量标准。



NVDLA VDK components

以下是学生在竞赛中需要完成的目标任务：

1. 按照给定的指导步骤设置工具环境，创建并编译 AI-VDK 平台，然后运行 AI-VDK 启动 Linux，执行 AlexNet/ResNet 并得到正确的结果。通常，AlexNet 需要 5~10 分钟才能完成，ResNet 需要 20~30 分钟，具体取决于不同的主机配置。

2. 在运行 AlexNet/ResNet 的同时，对 AI-VDK 仿真插入性能检测点并进行统计分析，找出 NVDLA 加速器模型中仿真速度的瓶颈；

3. 提高 AI-VDK 的仿真速度。根据任务 2 的结果，探索以下不同的仿真技术，以加速 NVDLA 加速器模型的执行。根据应用的不同技术收集性能提升数据：

a. 子任务 A：使用专门的 x86（SIMD）指令来加速关键算

法的计算功能。

b. 子任务 B：使用 OpenMP（Open Multi-Processing）加速关键算法的执行。

c. 子任务 C：在主机中有独立的 GPU 卡，利用主机 GPU 加速计算密集型的算法部分，即主要的 VDK 仿真运行在 CPU 上，而模型中的一些关键算法部分会调用 GPU 运行加速。

d. 子任务 D：根据给出的编码准则和简单示例，将 VDK 工具提供的多核仿真技术应用到 NVDLA 模型中。

五、评分标准：

技术评分 70%(按百分比折算 70%)：

任务1：正确理解任务，搭建平台和仿真环境	40分
任务2：正确并高效设计分析仿真速度瓶颈的手段，找到仿真瓶颈点	20分
任务3：实现SIMD 仿真加速，并提供加速比信息	10分
实现OpenMP仿真加速，并提供加速比信息	10分
实现GPU仿真加速，并提供加速比信息	10分
实现VDK多核加速，并提供加速比信息	10分

六、参考文献

- nvdla.org
- https://www.synopsys.com/verification/virtual-prototyping/vp-book.html

第二届（2020年）集成电路EDA设计精英挑战赛
赛题指南

一、赛题二：版图局部详细布线

二、命题单位：北京华大九天软件有限公司

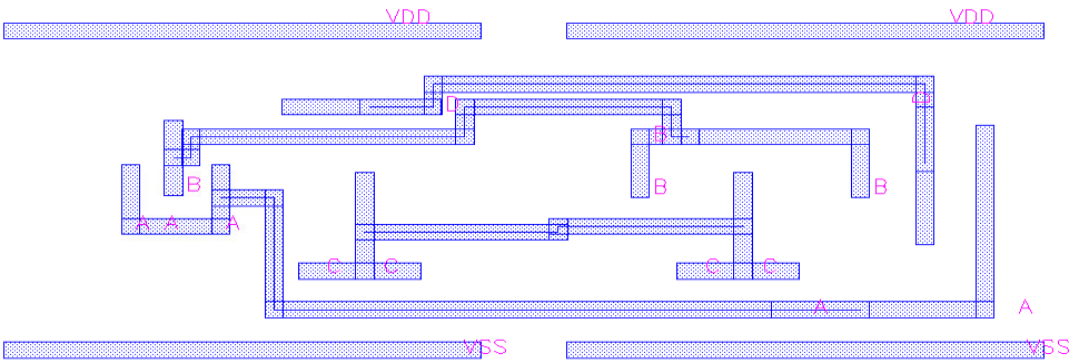
三、赛题背景：

布线在物理设计流程中是非常重要的一个步骤，它将分布在芯片核内的模块、标准单元和输入输出单元按照逻辑关系进行互连，并且满足设计规矩，布线结果的好坏直接影响整个芯片的性能。近年来，随着集成电路制造工艺的迅速发展，芯片集成度和复杂性不断增加，一块芯片所集成的线网数目不断增大且密度持续增高，使得 VLSI 布线难度越来越大。不仅仅是因为线网增长的数量和规模，还有随之增长的可制造工艺约束规则，为了提高良率，这些 DFM 规则已经增长到成百上千，并且随着制造工艺节点持续增加。因此，设计能够处理海量级网表、并且满足先进工艺设计规则的布线器一直是 EDA 领域的核心问题之一。

四、赛题描述：

1）总体描述：将版图内的模块、标准单元和输入输出单元（I/O Pad）按照逻辑关系进行互连，并且满足 ERC、DRC 和其他布线指标，如总线长、通孔数量

等，如下图所示：



2）输入：若干个版图的原始 LEF，DEF

3）输出：布线之后的 DEF

4）LEF 和 DEF 的说明：

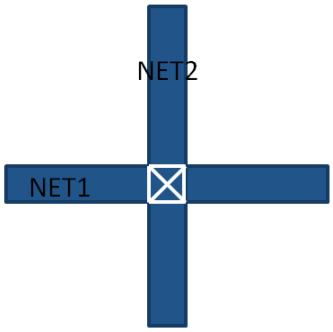
a） LEF： LEF(Library exchange format)是自动布局布线(APR)所必须的库文件，它主要分两个部分，Technology LEF 和 Cell LEF。Technology LEF 里边包含工艺信息、设计规则信息、通孔信息等。Cell LEF 中包含单元库中各单元的信息。

b） DEF：DEF(Design exchange format)设计交换格式。这个文件中包含了电路的连接关系网表，并且描述了电路布局布线后单元及互连线的具体物理信息。输入 DEF 中将带有布局数据；输出 DEF 要增加布线结果数据。

5) ERC 标准：

a） OPEN：如果在 net 上的任意一个 pin 没有进行连接，则此 net 视为 open。

b） SHORT：如下图所示，两条不同 NET 短路

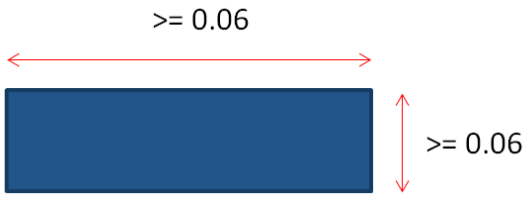


SHORT

6）DRC 标准：考察以下三种 DRC

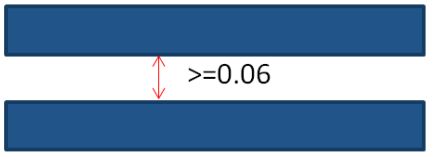
a) Min Width，语法如：

```
LAYER M1
  TYPE ROUTING ;
  MIN WIDTH 0.06;
END M1
```



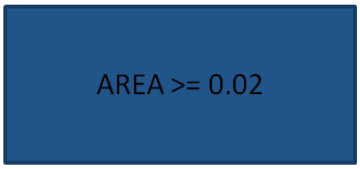
b) Min Spacing，语法如：

```
LAYER M1
   TYPE ROUTING ;
   DIRECTION HORIZONTAL ;
   SPACING 0.06;
END M1
```

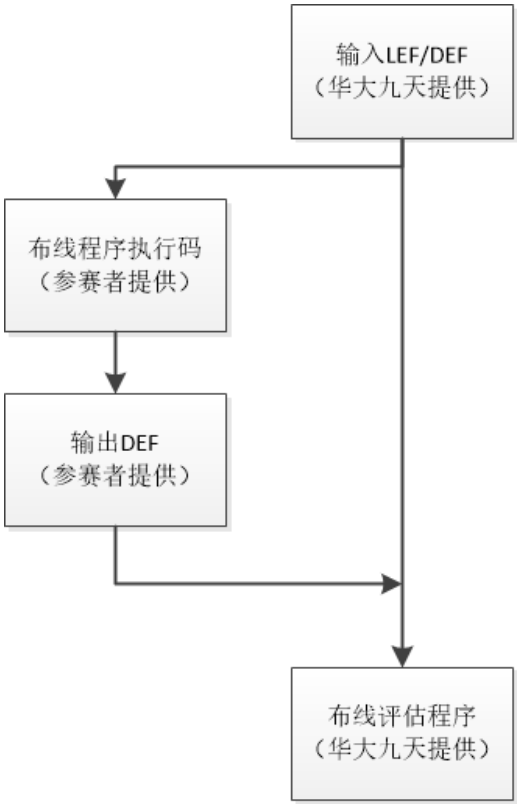


c) Min Area，语法如：

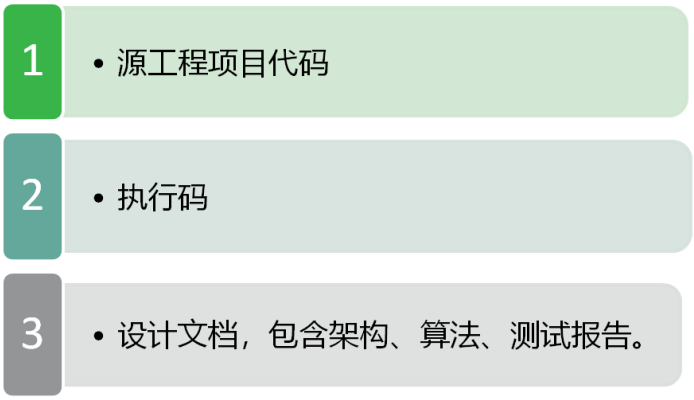
```
LAYER M1
   TYPE ROUTING ;
   WIDTH 0.06;
    AREA 0.02;
END M1
```



7）基本流程如下：



8）提交要求：



其中执行码的命令行和参数要求为：

router.exe –lef input.lef –def input.def –result result.def

–lef ：输入 lef

–def：输入 def

–result：输出 def，包含布线结果

五、 评分标准

a） 技术评分总分 100 分，每道题目均依据以下计分公式：

Score = basic_score * (1 + runtime_factor) 来进行计算，Score 越小得分越高，其中：

- basic_score 为以下布线指标的加权和，布线指标将由我们提供的评测程序计算。括号中的数值为每项的权值。线长单位是 M1 的 Pitch。
  - Open Net Number（10000）
  - Short Number（1000）

- Number of min-spacing violations (1000)
- Number of min-width violations (1000)
- Number of min-area violations (1000)
- Total number of vias (4)
- Total length of wires (1)

■ runtime_factor 的最大值为 0.1，最小值为-0.1。公式为：

runtime_factor = min (0.1 , max (-0.1 , 0.1 * log10 (runtime / reference_runtime)))

其中 reference_runtime 为参考运行时间，每道题都有自己的参考运行时间；runtime 为程序运行时间，多次结果挑选中间值。按照计算公式所述，如果布线器比参考标准快 1 倍或者慢 1 倍，则会获得大概 3%的分数奖励/惩罚。

■ 对于每道题目，所有队伍都要排名。分数越低，排名越高。最终计算每个队伍的平均排名值，排名值越低，技术评分越高。如下表所示：

	Team1	Team2	Team3
benchmark1	70	60	50
benchmark2	10000	INT_MAX	500
benchmark3	400	300	200
benchmark4	3000	5000	4000

	Team1	Team2	Team3
benchmark1	3	2	1
benchmark2	2	3	1
benchmark3	3	2	1
benchmark4	1	3	2
AVG	2.25	2.5	**1.25**

b)九天将根据评分规则的不足逐步改进规则，以利于提升比赛水平

六、参考资料

1） 《Handbook of Algorithms for Physical Design Automation》

2） 《超大规模集成电路布图理论与算法》

3） Dr-cu

https://github.com/cuhk-eda/dr-cu

4） Rsyn - x

https://github.com/RsynTeam/rsyn-x

5） CUGR

https://github.com/cuhk-eda/cu-gr

6） FastRoute

https://github.com/The-OpenROAD-Project/FastRoute

第二届（2020年）集成电路EDA设计精英挑战赛
赛题指南

一、 赛题三：智能 MPW 拼接

二、 命题单位：北京华大九天软件有限公司

三、赛题背景：

在 IC 的设计、制造等过程中，常常遇到模块拼接的情形，即对于给定数目的多个不同形状和尺寸的芯片版图（模块），根据给定规则，对其进行合理摆放，使得组合而成的矩形区域面积最小。

这个背景来源于多芯片的 MPW 拼接成一个曝光面积矩形（称为一次 shot，即光刻机进行一次曝光的面积大小），需要求出最小的曝光面积，对于晶圆的利用率则最高。

在带有连接关系的版图自动布局布线中，其结果好坏的指标计算时间很长，对问题的求解形成障碍。求解方案需要考虑时间代价。而人工版图拼接的工作量负担沉重、且无论对于以上那种情形，利用率均难达最优。急需自动生成版图拼接技术。

该需求可以扩展到其他有关 AI Placement 的应用，前景十分广泛

四、 赛题描述：

1）输入：给定 N 个不同形状和尺寸的多边形(相当于版图边界外形)，多边形为矩形或边均为正交方向的多边形。

2）求：最小的拼接面积

3）拼接规则：

a)各多边形之间，允许间距为 0，但是不允许两个多边形重叠（overlap）

b)各多边形允许做基本的几何旋转（0°/90°/180°/270°/MX/MY）

c)最终拼接的总区域外接矩形， 其 50um ≤ 宽 ≤ 300um，50um ≤长 ≤ 400um

4）难度升级：

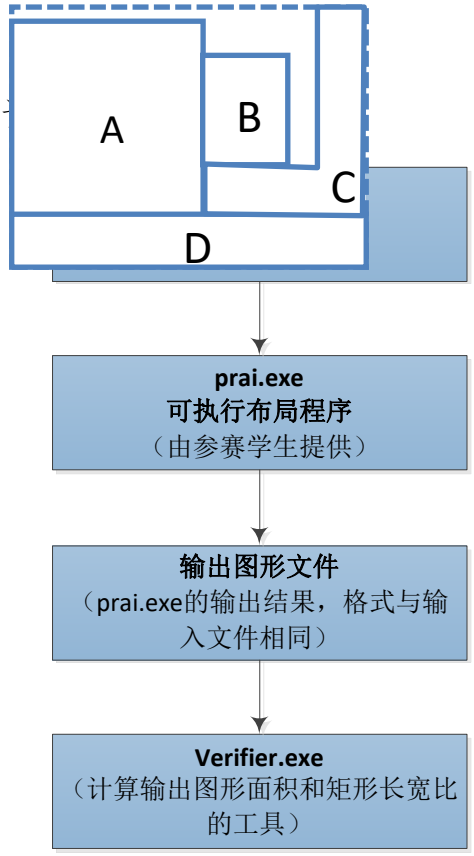
a) 增加时间代价函数，每次计算面积，均自动等待指定时间（比如 15 分钟）才给出结果

b) 增加对拼接矩形形状的限制，即在相同的矩阵面积情况下，长宽比越接近 1：1（也即越接近正方形）的矩形更优。这是因为 IC 制造过程中，如果一个 shot 越细长，意味着一次曝光中有半导体器件的几何距离越遥远，器件之间工艺误差可能会越大。

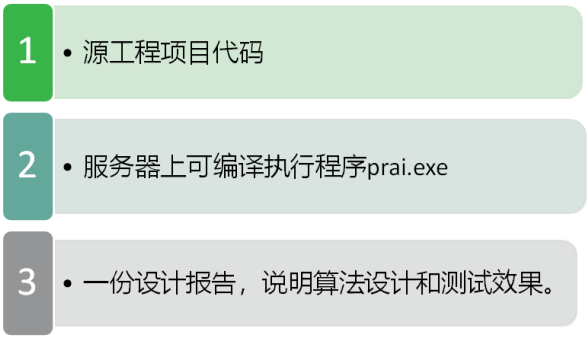
5) 举例：对以下 4 个多边形，只有如下的拼接方式（虽然必

不可少也会造成一些面积浪费），能使得虚线框的面积最小。

6）基本流程：



7）提交要求：



其中，对可执行程序 prai.exe 的要求是：

a) Linux 服务器环境上可以正确执行

b) 执行示例：prai.exe /xxx/xxx/ inputfig.txt

c) 输出：/xxx/xxx/result.txt

d) 说明：

■ 输出文本和输入文件在同一目录下

■ 输出文本命名有要求，必须是 result.txt

8) 求解思路猜想：

a) 爬山、局部搜索、蚁群算法、遗传算法、模拟退火等

b) 动态规划

c) 贝叶斯优化（Bayesian Optimization）

d) 强化学习

e) 参考文献，见本文最后章节

五、 评分标准

a) 每个程序的技术评分为 100 分，其中：

■ 总面积占 40 分：收集程序运行所有 case 的总面积进行排序。依据排序结果取前十个值（面积值相同时作为并列处理），分为满分 40 分、35 分、30 分、28 分、26 分、24 分、23 分、22 分、21 分和 20 分给分。其余情况计为 0 分。

■ 运行时间占 20 分：收集程序运行所有 case 的总时间（精确到毫秒）进行排序。依据排序结果取前十个值（时间相同时作为并列处理），分为满分 20

分、17 分、14 分、12 分、10 分、9 分、8 分、7 分、6 分和 5 分给分。其余情况计为 0 分。

- 运行内存占 20 分：收集程序运行每个 case 的内存峰值的和（精确到兆字节）进行排序。依据排序结果取前十个值（值相同时作为并列处理），分为满分 20 分、17 分、14 分、12 分、10 分、9 分、8 分、7 分、6 分和 5 分给分。其余情况计为 0 分。
- 拼接矩形形状占 20 分：计算程序运行每个 case 的输出结果的外接矩形的长宽比与目标矩形长宽比的偏离值（保留 2 位小数，取绝对值），将程序所有 case 的偏离值的和进行排序。依据排序结果取前十个值（值相同时作为并列处理），分为满分 20 分、17 分、14 分、12 分、10 分、9 分、8 分、7 分、6 分和 5 分给分。其余情况计为 0 分。如果矩形形状超出了 50um ≤ 宽 ≤ 300um， 50um ≤长 ≤ 400um 的限制，此项直接给 0 分。

c) 九天将根据评分规则的不足，逐步改进规则，以利于提升比赛水平

六、 参考资料： https://arxiv.org/abs/2004.10746v1

1） Azalia Mirhoseini, Anna Goldie, Google: Chip Placement with Deep Reinforcement Learning. *22 Apr 2020.* https://arxiv.org/abs/2004.10746v1

7) Lienig, J.: Layoutsynthese elektronischer Schaltungen – Grundlegende Algorithmen für die Entwurfsautomatisierung. 2. Aufl. Berlin, Springer Verlag 2006, ISBN 978-36-624-9814-9.

8) Determinismus eines Algorithmus. (PDF) In: Informatik Duden. Bibliographisches Institut, Berlin, 2001, abgerufen am 31. Januar 2018 [Online]. https://www.fmi.unijena.de/fmimedia/Fakultaet/Institute+und+Abteilungen/Abteilung+f%C3%BCr+Didaktik/GDI/Determinis-mus+eines+Algorithmus-p-16913.pdf

9) Selig, B.;Kern, V.; Walther, T : Eigenschaften von Algorithmen. Tilman Walther, März 2004, abgerufen am 31. Januar 2018. [Online]. http://www.tilman.de/uni/ws03/alp/eigenschaf-tenVonAlgorithmen.php

10) Otten, R.; Ginneken, L. P. P. P.: Floorplan Design Using Simulated Annealing. Prov. Of the ICCAD 1984, 96-98, 1984

11) Kohonen, T.: Self-organized Formation of

Topologically Correct Feature Maps. Biologi—cal Cybernetics, 01.43, pp.59— 69, 1982

12) Ritter, H.; Schulten, K.: Topology Conserving Mappings for Learning Motor Tasks. in Neural Networks for Computing, AIP Conference Proceedings 151, Ed. J.S.Denker, pp.376—380, Snowbird, Utah, 1986

13) Zhang, C.; Mlynski, D. A.: VLSI—placement with a neural network model. Proc. Int. Symp. on Circ. and Syst. , pp. 475—478, 1990.

14) Aykanat, C.; Bultan, T.; Haritaoglu, I.: A fast neural—network algorithm for VLSI cell placement', NeuralNetworks, Vol. 11, No. 9, pp.1671—1684. 1998

15) Hopfield, J.; Tank, D.W.: Neural computation of decisions in optimization problems

16) N e t t o , R .; F a b r e , S .; F o n t a n a , T .: How Deep Learning Can Drive Physical Synthe—sisTowards More Predictable Legalization [Online] https://hal.archivesouvertes.fr/hal—02057042/document

17) H .; F u ji y o s h i, K .; K a jit a n i, Y .: VLSI module placement based on rectanglepacking by the sequence—pair," IEEE Trans. on Computer—Aided Design of

Integrated Circuits and Sys-tems, Vo1.15, No.12, pp.1518-1524 (1996).

18) MCNC Benchmark Netlists for Floorplanning and Placement [Online] https://s2.smu.edu/~manikas/Benchmarks/MCNC_Benchmark_Netlists.html

19) Rosenblatt, F.: The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory. 1957

20) Maegan Tucker, Ellen Novoseller, Preference-Based Learning for Exoskeleton Gait Optimization, 26 Sep 2019, https://arxiv.org/abs/1909.12316

[2004.10746v1] Chip Placement with Deep Reinforcement Learning

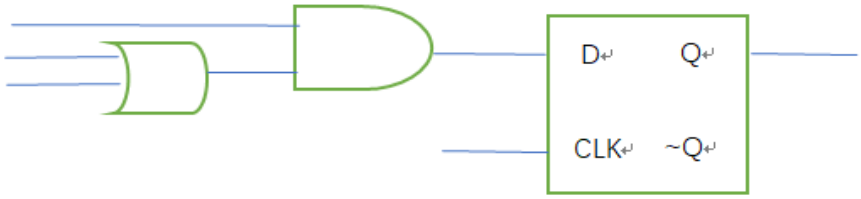
第二届（2020年）集成电路EDA设计精英挑战赛
赛题指南

一、赛题四：时序模块驱动冲突的检查

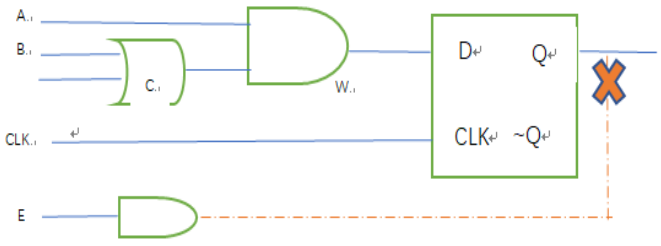
二、命题单位：芯华章科技股份有限公司

三、赛题描述：

在 RTL 的设计中，有一部分是组合逻辑 (combinational logic)，一部分是时序逻辑(sequential logic)。一般而言，时序逻辑的每个输入只能有一个驱动（driver），该驱动可以是组合/时序逻辑的输出。 如：



在实际 RTL 开发中，有一类比较常见的错误，就是一个时序电路的输出（如 Q）会有多个驱动，如：



其对应的 RTL Verilog 为：

```
always @(posedge clk)
    Q <= A & (B | C);
always @ (E)
    Q = E;
```

通常，这种错误往往在仿真结果出现问题的时候才能发现，用户调试起来周期非常长，尤其是 RTL 设计非常大的时候。理想的情况是验证工具进行静态检查(static check)， 让这类问题在编译阶段(compile time)报出，从而节省用户的调试时间。本赛题就是要求参赛者开发这样一种静态检查的功能。

四、驱动冲突的规则

4.1 驱动冲突的检查对象是变量（variable）， 而非 wire（net type），因为 IEEE 标准已经确保了 wire 是不能用被时序电路驱动的。变量的类型包括 reg, integer 等。

4.2. 驱动冲突的检查模块是 always block. 在其他 scope 下，多个驱动是允许的。如两个 initial block 里驱动同一个变量是合法的. 但是同一个变量同时被一个 initial block 和 always block 驱动就是有冲突的。

4.3 在同一个 always block 中的多次写操作不视为驱动冲突

```
always @(posedge clk)  begin
  if (sel)
    Q <= D1;
  else
    Q <= D2;
end
```

五、测例描述

RTL 设计以 Verilog 形式，如：

```
1: module dut (input a, b, c, e, output q);
2:  wire a, b, c;
3: reg r;
4: reg clk;
5: reg q;
6:  reg d;
7:  always @(a, b, c)
8:     d = a & (b | c);
9:  always @(e)
10:    q = e;
11: always @(posedge clk)
12:    q <= d;
13: endmodule
```

输出：
屏幕上需要打出如下报错信息（至少两个）

```
The following drivers conflict:
 Line 10
 Line 12
```

六、开源代码

可以下载开源的 Verilog simulator
http://iverilog.icarus.com/home

七、 评分标准

评分标准包括功能的覆盖程度、代码质量、测试，及性能等几部分

7.1 功能覆盖及质量 （80 分）

1） 基本功能(20 分)
支持 scalar type (reg 等)，以各种 driver 类型，

2) 支持 vector/memory，以及 select （30 分）

E.g.

```
reg [3:0] aa

always @(bb)
    aa[0] = bb;
always @(posedge clk)
    aa[1] = cc;
```

这个例子应该是没有冲突的，因为他们的 select 不同

而下面这个会在 a[1]有冲突：

```
always @(bb)
    aa[1:0] = bb;
always @(posedge clk)
    aa[3:1] = cc;
```

3）支持 for loop 分析 （30 分）：

a)支持单层的 for loop （15 分）

```
reg [0:7] aa;
reg [0:7] bb;
reg [0:7] cc;
always @(posedge clk)
    for (i = 0; i < 4; i++)
        aa[i] <= cc[i];

always @(posedge clk) begin
        for (i = 3; i < 8; i++)
            aa[i] <= bb[i];
end
```

这里，aa[3] 是有冲突的，其他位没有冲突

b) 支持嵌套的 for loop (15 分)

```
reg [0:7] aa [0:15];

always @(posedge clk)
   for (i = 0; i > 7; i++)
      for (j = 0; j < i + 1; j++)
```

```
            aa[i][j] = cc;


always @(posedge clk)
    for (i = 0; i > 7; i++)
        for (j = i+1; j < 15; j ++)
            aa[i][j] = bb;
```

以上两个是没有冲突的。

7.2. 代码及测试 （10 分）

主要评估标准：

1）选手代码的整洁程度，可读性，可调试性

2）对原开源源代码的改动在满足性能/功能的前提下，要尽可能的小（避免重复开发而导致质量问题）

3）是否有测试方案以确保软件的正确性

7.3 性能 （10 分）

根据一组提供的 benchmark， 以 compile time（CT）为衡量依据。性能超越 80%参赛选者得 10 分，性能超越 50%参赛选手者得 5 分。

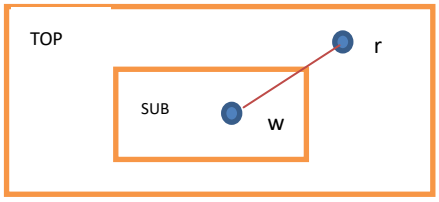
7.4 附加题 （20 分）

Verilog 有一种概念是 hierarchical reference，即从一个 module 可以访问定义在另一个 module 的 variable. 例如：

```
module top;
    sub s0();
    wire r;
endmodule

module sub;
    wire w;
    assign top.r = w;  //hierarchical reference
endmodule
```

其对应的概念是：



如果 ‘sub’ module 在’top’ 下面有多个 instance，会造成’r’ 有多个驱动。

完整的例子如下：

```
1: module top;
2:     reg q;
3:     submod s1();
4:     submod s2();
5: endmodule
6: module submod();
6:      reg d, clk;
7:      always @(posedge clk)
8:          top.q <= d;
9: endmodule
```

由于这个驱动在源代码中只有一处，为了给用户更多信息，需要把这个驱动所在的 hierarchal path 打出来：

```
The following drivers conflict:
 Line 8,  in instance top.s1,
 Line 8,  in instance top.s2
```

八、测试集

比赛开始时，将提供 50 个左右单元测试供参赛选手使用。

验收时，将在另外一个测试集 （100-200 个 case）中测试选手的提交的软件，以通过率作为功能覆盖率（5.1）的打分标准。另外，会有一个性能测试集（20 个左右 benchmark），作为性能（5.3）的打分标准。

九、技术指导

对于不熟悉 Verilog 以及 iverilog 开源软件的选手，可以提供一定的技术指导，使选手可以尽快进入核心功能开发阶段。

十、 参考资料：

1) IEEE1364 Verilog Standard
2) Iverilog : http://iverilog.icarus.com/home

第二届（2020 年）集成电路 EDA 设计精英挑战赛
赛题指南

一、赛题五：基于分布式计算框架的自动测试向量生成算法

二、命题单位：深圳市海思半导体有限公司

三、赛题背景：

a)、概述

在 VLSI 电路的设计过程中，DFT 是保证回片测试质量的重要手段，TPG（Test Pattern Generation）是其中一个重要的环节。随着电路规模的增大、设计复杂度的提升和先进工艺演进，高效率、高质量的产生测试向量变得更加困难。

ATPG（Automatic Test Pattern Generation）是整个 TPG Flow 中的核心组件，一个好的工业级 ATPG 算法应能够在更短的时间内，产生更少的测试向量数，并获得更高的测试覆盖率，使得测试成本更低。多年以来，业界针对 ATPG 有着持续研究，从布尔差分法、GF 二值算法[1]将电路模型转换成数学模型来生成测试向量，到 D 算法[2]及其改进算法 PODEM[3]、FAN[4]等基于故障激活的测试向量生成算法，以及 BDD、SAT-Based ATPG 算法[5]，持续努力在寻求新的算法方案以达到高效率、高质量的测试向量。

然而，随着电路规模的日益增大，TPG Flow 的性能逐渐成为瓶颈，主要表现在算法的效率和内存两方面。在算法上的优化探索不能十分有效地解决问题。传统的数据计算和数据存储方式已经无法满足要求。为解决内存和效率问题业界提出了分布式的解决方案。MapReduce[6]是当前较为成熟的开源分布式计算框架，Hadoop 和 Spark 是 MapReduce 框架的两个比较有影响力的实现。Hadoop 的核心组件包括 Hadoop Distributed File System（HDFS）、Hadoop MapReduce 等模块，其中 HDFS 完成数据的分布式存储，Hadoop MapReduce 完成数据的分布式计算。而 Spark 不同于 Hadoop 的是其 job 的中间输出和迭代结果可以保存在内存中，从而可以获得更高的数据访问速度。同时，结合 Redis 高性能数据库，在提升 TPG 中的静态数据的读取速度方面有较大可能性。综上所述，分布式计算框架与 Redis 的结合来实现 TPG Flow，是当前解决 TPG Flow 性能瓶颈问题的值得探索方向之一。

a) 传统的 TPG 流程

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 fault model。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。

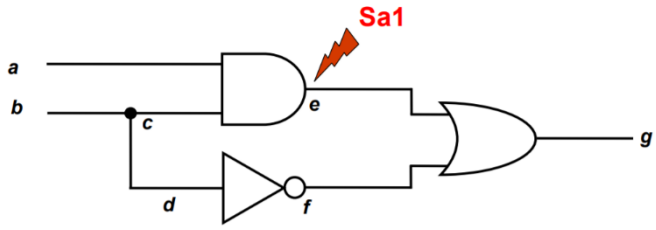


图 1

在集成电路中，对指定的 fault 产生 pattern 的过程，称为测试向量产生（Test Pattern Generation，缩写为 TPG）。本赛题仅针对组合电路的 stuck-at fault 的向量产生。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。由于 ATE 的测试成本非常高，因此如何针对指定电路的 fault list 生成一组尽可能少的 pattern 集合是 TPG 需要解决的问题。

一个传统的 TPG 流程[7]如图 2（组织方提供该 flow 的源码和说明文档，C++实现，参赛者基于对源码的理解实现自己的并行 flow）所示。首先，解析网表并得到 netlist；根据 netlist 生成完整的 faultlist 集合；选择部分 fault 做测试向量生成；然后，经过 fault simulation 验证 pattern 正确性。如果这些 pattern 能够检测到除了被选择的之外的 fault，则将这些 fault 一起标记为 tested。循环此过程，直至所有的 fault 被选择。如果最终得到的 pattern 能够标记 tested 的 fault 个数除以 fault 的总个数，就是这组 pattern 针对该网表的覆盖率（coverage）。覆盖率是评价一组 pattern 的好坏的重要指标之一。

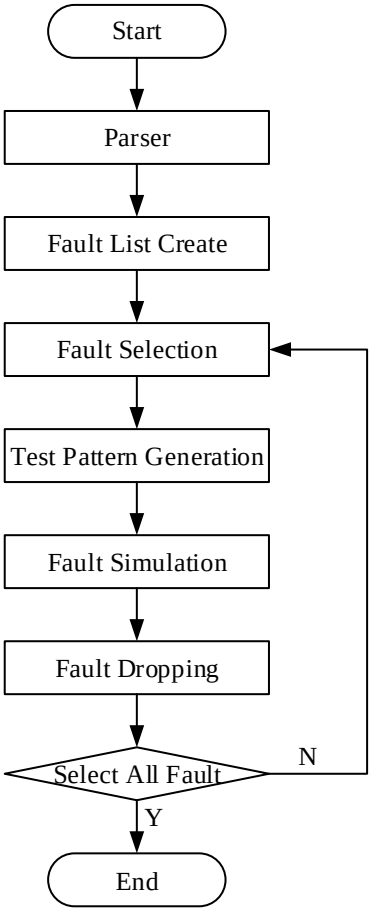


图 2

b) 传统的 TPG 算法的问题与分布式计算的优势

TPG 是一个 NP-Complete 问题，且传统的 TPG Flow 往往采用前述的串行迭代的计算方式。随着芯片技术的不断进步，芯片集成电路规模在快速增加。对超大规模集成电路的执行测试向量产生的计算，不仅计算量十分巨大，还需要巨大的内存资源。这不仅导致测试向量的生成过程十分耗时，还导致在单台计算机资源难以支撑巨量的内存消耗，最终会导致测试向量无法生成。

当前 IT 领域已经步入云计算的时代，以云计算为基础设施的分布式计算架构，可以同时利用多台计算机的计算资源来完成大规模的计算任务。与传统的单机处理模式相比，分布式计算架构在单位时间内可以拥有更高的信息处理速度、算力扩展性和更多的内存资源。理论上，分布式计算架构可以很好的解决传统的 TPG Flow 的高算力和高内存的诉求。

c) TPG 与 MapReduce 结合的新思维

MapReduce 是一种编程模型，用于大规模数据集的并行运算。它借助于函数式程序设计语言 Lisp 的设计思想，提供了一种简便的并行程序设计方法，用 Map 和 Reduce 两个函数编程实现基本的并行计算任务，提供了抽象的操作和并行编程接口，以简单方便地完成大规模数据的编程和计算处理。

MapReduce 的工作原理如图 3 所示。MapReduce 主要分为两个计算阶段：

- Map 阶段

Map 阶段由一个或者多个 MapTask 组成。每个 MapTask 处理输入数据集合中的一片数据(InputSplit)，如下图中的 split 0~ split 4。并将计算结果（同样是多个数据片段）写到本地磁盘上。

- Reduce 阶段

Reduce 由一个或者多个 ReduceTask 组成。ReduceTask 则从每个 MapTask 上远程拷贝相应的数据片段，经分组聚集和归约后，将输出数据回写到 HDFS 上作为最终结果。

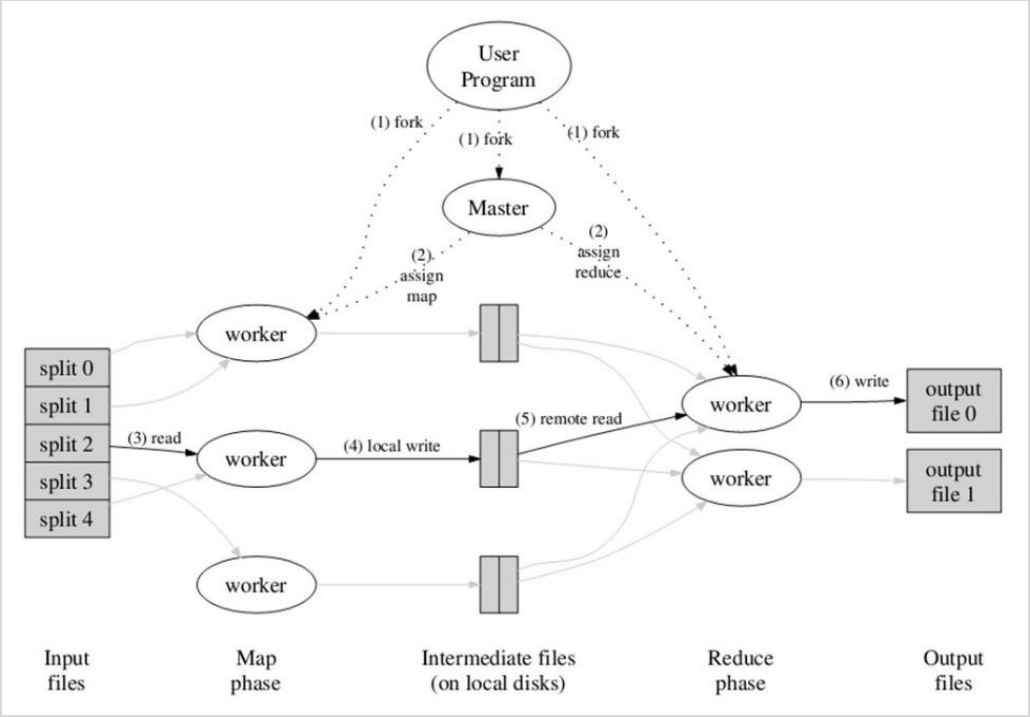


图 3

我们可以利用 Hadoop 的 MapReduce 组件来完成 ATPG 的计算。为此，我们可以把 ATPG 模块放到 Map 任务中，并将 fault list 放到 HDFS 上。由框架自动实现 fault list 切分和任务调度。由于每个 ATPG 任务需要在生成测试向量的过程中使用网表的连接关系，我们可以考虑将网表信息存到 Redis 数据库中，以支持多节点的快速访问。在 Reduce 阶段，获取生成的 pattern 以便执行后续处理，比如精简 Pattern 等。由于 Hadoop 的 MapReduce 组件会涉及大量的 HDFS 文件读写，在处理迭代式计算时显得力不从心，因此如果想实现一个多轮迭代式的 TPG flow，还可以使用类似于 Spark 的内存计算框架。

四、赛题描述

1）描述

本课题需要参赛队伍基于给定的 ATPG 算法完成一个分布式的 TPG 系统，并期望达成下列几个基础的技术目标：

1. 基于 MapReduce 的分布式的 TPG 系统。
2. 从软件的算法、流程、系统架构等方面优化系统以获得更优的技术指标。

本赛题的总体方案如图 4 所示：

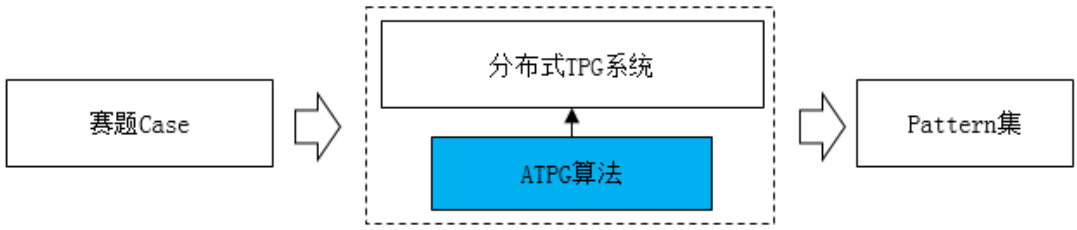


图 4

- **赛题 Case**

赛题 Case 为系统输入之一，该部分由出题方提供。参赛队伍也可自行构造一些 Case，便于验证和优化自己的算法和系统。

- **分布式 TPG 系统**

构建分布式的 TPG 系统为本赛题的核心任务。出题方会提供简单的搭建指导，参赛队伍需要实现环境自动搭建脚本，用于将自己实现的环境从无到有自动搭建出来。

- **ATPG 算法**

ATPG 算法也是整个系统的核心，该部分出题方会开源可以构建和运行的基础代码。但该算法并非最优化，需要参赛队伍大胆创新和优化。

- **Pattern 集**

Pattern 集为系统输出，是一组 Pattern 集合，出题方仅会给出基准 Pattern 总数和 TPG 的基准运行时间。参赛队伍利用自己实现的算法和系统生成 Pattern 集。

算法基础源代码及竞赛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://gitee.com/openeda/OpenTPG

2）竞赛规则

竞赛流程

下面为该课题竞赛流程：

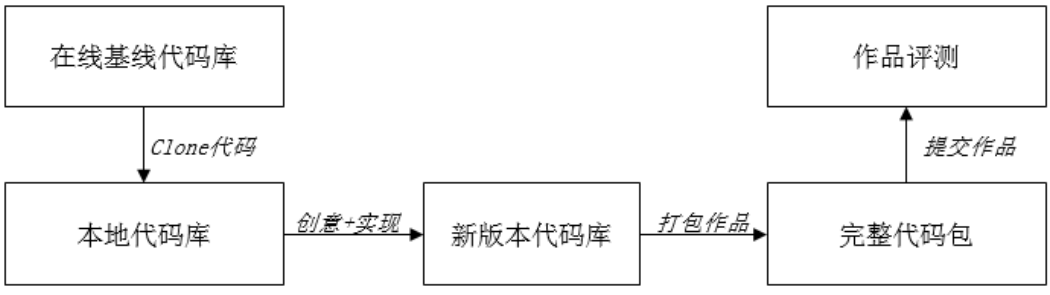


图 5

各参赛队伍需要以赛题提供的在线代码库为基础，通过自己的知识、技能、实践和创意来实现一个新的分布式 TPG 系统。

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**

考虑到从零开始完整实现一个 TPG Flow 软件，是有一定难度和工作量的，我们建议参赛队伍都以已经开源的源代码为基础进行修改和增强。除非您的队伍有信心，否则您应该避免完全重新开发全部源代码。

- **竞赛Case**

出题组会提供 4 套竞赛 Case，用作日常调测和竞赛比分。其中，3 套为常规 Case，1 套为挑战 Case。各 Case 有难易程度之分，相应地会有不同的评分权重。各 Case 的评分权重和计分方式，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。

五、评分标准

b) 技术评分，共 80 分

i. 技术点评分标准

技术评分的主要的评价标准是 TPG 系统的性能的几个核心技术指标。本竞赛项目我们只关注 TPG 系统的输出的 Pattern 的覆盖率、Pattern 数、运行时间这三项技术指标。这三项技术指标有一定的约束关系：

- 覆盖率，决定了系统输出的Pattern的质量，是所有TPG系统中的各项技术指标中最关键的要素。如果输出的Pattern的覆盖率不达标，即使Pattern数再少，运行时间再快，也将失去价值。
- Pattern数，决定了系统输出的Pattern的测试成本。如果测试成本太高，那么这些Pattern将无法交付使用。
- 运行时间，是影响芯片交付周期的关键因素之一。更短的ATPG运行时间，能够加快芯片的研发速度，芯片可以更快上市。

以下为我们关注的主要技术指标以及评分标准：

（1） 覆盖率，40分

所有 Pattern 的总覆盖率是 ATPG 基本要求，我们认为覆盖率不达标的 Pattern 是没有价值的。不同覆盖率的评分方式如下：

- 覆盖率达到或者高于95%：40分

- 覆盖率低于95%：0分

（2） Pattern数，20分

TPG 最终所产生的 Pattern 数越少越好。考虑到不同的 Case，其网表 Pattern 的数量都是有差距的，所以每个网表都会有一个基线 Pattern 数。具体评价标准如下：

- Pattern数小于基线值的0.5倍：20分
- Pattern数为基线值的：[0.5,0.6)倍：15分
- Pattern数为基线值的：[0.6,0.7)倍：10分
- Pattern数为基线值的：[0.7,0.8)倍：5分
- Pattern数大于或等于基线值0.8倍：0分

（3） 运行时间，20分

TPG 的运行时间越短越好。考虑到不同的 Case，其 TPG 的计算时间是有差距的，所以每个网表都会有一个基线运行时间。具体评价标准如下：

- 运行时间小于基线值的0.4倍：20分
- 运行时间数为基线值的：[0.4,0.5)倍：15分
- 运行时间数为基线值的：[0.5,0.6)倍：10分
- 运行时间数为基线值的：[0.6,0.7)倍：5分
- 运行时间大于或等于基线值的0.7倍：0分

ii. 技术评分总分计算方式

技术评分环节由多个 Case 组成，每种 Case 会有不同的电路特点，用以验证参赛队伍搭建的 TPG 系统的对不电路特性的适应能力。所以，各参赛队伍应该避免设计方案过于偏向特定电路。每个 Case 的各技术分评分会单独评分，技术评分总成绩有各 Case 的技术评分的加权和来决定。下面为每个 Case 的评分权重为：

- Case 0：10%（常规）
- Case 1：30%（常规）
- Case 2：40%（常规）
- Case 3：20%（挑战）

我们设定 case“i”的评分权重为 p(i)，每个 Case 的覆盖率得分为 c(i)，Pattern 数得分 n(i)，运行时间得分 t(i)，那么技术评分总分计算公式为：

$$\sum_{i=0}^{3}[c(i)+n(i)+t(i)]*p(i)$$

c) **设计报告，共 20 分**

行文要求条理清楚，详略得当，清楚易读，内容应该包括以下几个方面：

（1） 架构描述、问题分析、算法优化原理描述，共10分

（2） 方案创新性，共10分

六、参考文献

[1] Rolf Drechsler,Stephan Eggersglüß,Görschwin Fey,Daniel Tille.Test Pattern Generation using Boolean Proof Engines.

[2] J.P.Roth,W.G,Bouricius,and P.R.Schneider.Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits,IEEE Transactions on Electronic Computers.

[3] Prabhakar Goel.PODEM-X:AN AUTOMATIC TEST GENERATION SYSTEM FOR VLSI LOGIC STRUCTURES

[4] H.Fujiwara.FAN:A Fanout-Oriented Test Pattern Generation Algorithm.In Proceedings of the IEEE International Symposium On Circuits and Systems,pages 671-674,July 1985.

[5] Armin Biere,Wolfgang Kunz.SAT and ATPG:Boolean engines for formal hardware verification.

[6] MapReduce:Simplified Data Processing on Large Clusters.

[7] ELEC 516 VLSI System Design and Design Automation Spring 2010 Lecture 10 – Design for Testability.
https://www.slideserve.com/danton/elec-516-vlsi-system-design-and-design-automation-spring-2010-lecture-10-design-for-testability

七、Q&A

Q：ATPG 算法对于覆盖率和 pattern 数的影响？

ATPG 的算法本质上属于 NP-Complete 问题，该算法的优化总的来讲就是尽量缩小搜索空间和减少 backtrack 的次数。实际使用时通常会限制最大 backtrack 的次数，好的算法应在该限制之内尽可能提高产生 test cube 的成功率，该成功率最终变现为生成 pattern 集合的故障覆盖率。由于一个 pattern 通常能 detect 多个 fault，因此好的算法应该能产生更高质量的 pattern（more specified bits），用更少的 pattern 数量去获取目标覆盖率。

Cube 的定义：ATPG 产生的向量会对一部分 primary input 指定值，另外一部分为 X 态（do not care），此时的向量成为 cube，cube 经过 random fill 或者其它算法对剩余的 primary input 赋值后的向量成为 pattern。

d) Q：并行对于 pattern 数量的影响？

使用并行意味着可以获取更多的算力，在同样的时间内可以 target 更多的 fault 来生成 cube，对这些 cube 做合并和剔重等操作，更易于获取高质量的 pattern，从而减少最终的 pattern 数量。

e) Q：关于减少 Pattern 数和运行时间有哪些典型的方法？

ATPG 中可以优化的点是非常多的，也是该行业的研究热点，典型的优化思路有：

1. 将多个不同的Cube按照一定算法进行合并，可以直接减少Pattern的数量；
2. 在Fault Simulate之后，执行清除冗余Pattern的操作，可以进一步减少Pattern的数量；

第二届（2020年）集成电路EDA设计精英挑战赛
赛题指南

一、 赛题六：快速电路仿真器（FastSPICE）中的高性能矩阵向量运算实现

二、 命题单位：概伦电子科技有限公司

三、 赛题背景：

在晶体管级电路瞬态仿真过程中，仿真器需要根据电路连接关系并结合 KCL、KVL 定理建立微分方程，然后求解离散化的方程，中间每个步长输出电压和电流的仿真结果。在瞬态仿真中的每个步长都会涉及到大量的矩阵向量运算，例如电流输出的计算，需要计算很多条支路电流并求和。矩阵向量运算在整个仿真中占据了相当比例的仿真时间，尤其是针对存储器（memory）电路的快速电路仿真器（FastSPICE）。因此，加速矩阵向量运算可以直接提高电路的仿真速度。

电路方程的矩阵大小与电路的节点个数成正比，比如，大型的存储器电路（DRAM, Flash, SRAM 等等），其维度可达到 1000 万以上，如此高维度的矩阵以普通二维数组的方式存储会消耗巨大的内存。由于电路的每个节点通常只与少数节点有连接关系，因此电路方程矩阵通常是稀疏度很高的稀疏矩阵。稀疏矩阵特有的存储方式仅包含矩阵中的非零元素，可大大减少内存的消耗量，为存储高维度的矩阵提供可能。稀疏矩阵的矩阵向量运算与普通的矩阵向量运算略有不同，对整体运算速度有较明显的影响。

希望参赛者能够开发一种针对电路方程的高效矩阵向量运算方案和实现，减少运算时间，从而提高电路仿真速度。

四、 赛题描述：

4.1 概伦电子提供电路方程矩阵的数据接口和计算方法，参赛队开发高效矩阵向量运算的实现，并以动态库的形式提供。电路仿真器在执行指定电路网表仿真时会加载该动态库，并将相应的电路仿真矩阵向量运算通过预定接口传递给动态库，由参赛队开发的高效矩阵向量运算架构完成。电路仿真器在仿真中会反复地产生不完全相同的运算任务，而且在同一时刻会向动态库提供多个矩阵的向量乘运算任务。动态库在完成运算任务后需要将结果写入到接口指定的数据块中，然后电路仿真器继续其他任务的执行。参赛队可以采用并行、指令优化等多种高性能计算相关的优化方法对运算过程进行加速，以实现用更短的时间得到正确结果的目标。

4.2 本赛题要求参赛队完成快速电路仿真器 (FastSPICE)中的相关支路电流计算任务，其中电路元件的电导将构成 N×N 阶稀疏矩阵，节点电压组成 N×1 阶矩阵，其计算过程包含以下矩阵运算：

$$R_{n\times 1} = A_{n\times n} S_{n\times 1} \quad (1)$$

$$H_{n\times 1} = L_{n\times 1} - G_{n\times n} S_{n\times 1} \quad (2)$$

$$E_{n\times n} = G_{n\times n} + C_{n\times n} \quad (3)$$

其中稀疏矩阵的存储方式采用 Compressed Sparse Row(CSR)格式，由 3 个数组row_offset、$column_indices$和$values$组成（如图 1 所示）。CSR 格式仅对稀疏矩阵中非零元进行存储，其中$column_indices$和$values$两个数组分别按行依次存储矩阵中非零元的列值和数值。row_offset表示某一行的第一个非零元在$values$数组里的偏移位置。row_offset数组的长度为矩阵行数+1，$column_indices$和$values$两个数组的长度相同，为矩阵中非零元的个数。设稀疏矩阵第i行（$i = 0, 1, 2, \ldots, n$）的第一个零元的偏移位置为$row_offset[i] = P_i$，第$i+1$行的第一个非零元的偏移位置为$row_offset[i+1] = P_{i+1}$，则第$i$行的非零元的列值和数值分别为

$column\_indices[j]$, $values[j]$, $j \in [P_i, P_{i+1})$，且第i行的非零元的个数为$P_{i+1} - P_i$。

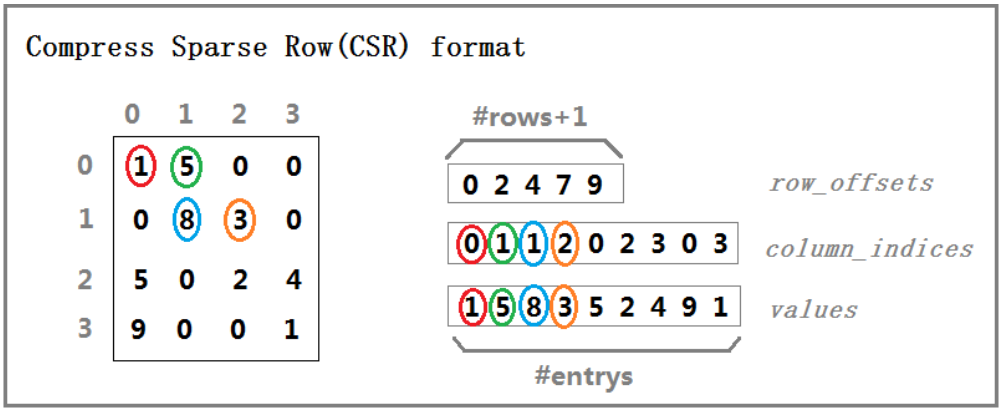
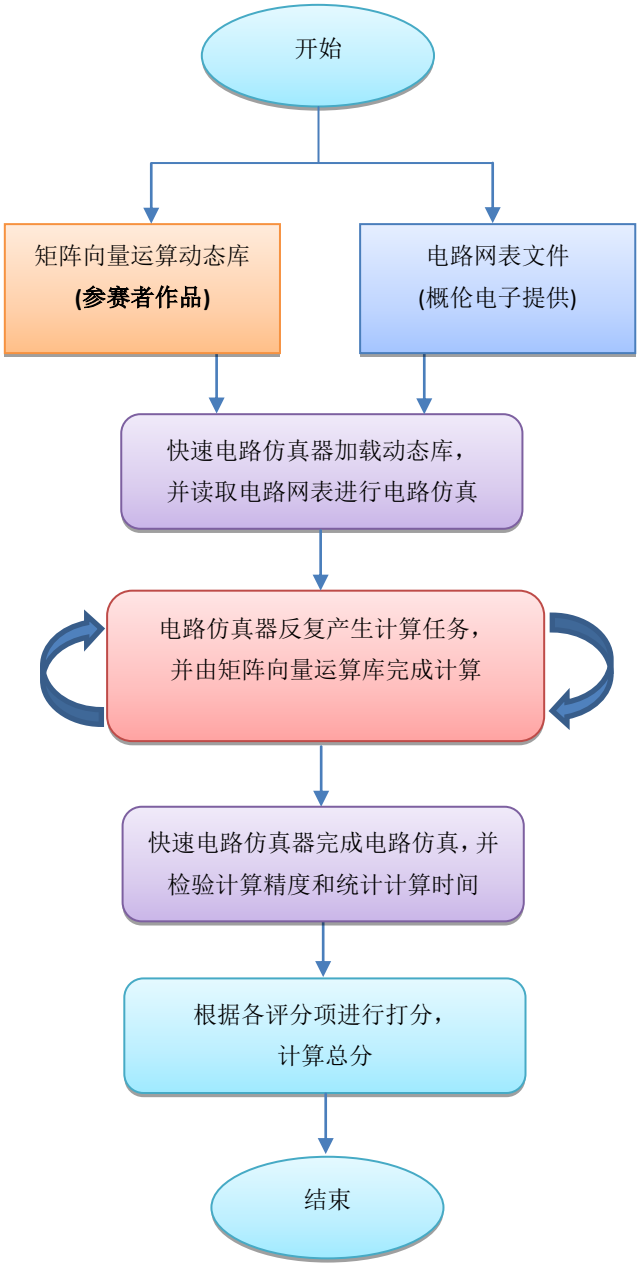


图 1

4.3 在电路仿真中，电路仿真器会在每个仿真点产生一次计算任务，每次计算任务包含多块 N×N 阶稀疏矩阵与 N×1 阶矩阵的运算的子任务，不同块之间矩阵向量运算是独立的。每次计算任务包含的矩阵块数是相同的，而且矩阵的维度和非零元的位置是不变的，但非零元的数值会发生变化。电路仿真器会根据电路的工作情况，只对矩阵或向量的某些行列进行运算，该信息可通过给定函数接口获取。电路仿真器会在每个仿真点通过调用指定的动态库函数接口将每次计算任务的所有矩阵按分块的形式给出，参赛队需要在动态库内完成的所有矩阵的运算任务，并将结果写入指定内存区域，然后返回。电路仿真器在动态库函数返回后再进行后续的电路仿真。

4.4 竞赛基本流程如下图：



五、 评分标准（最终解释权归概伦电子）

5.1 技术评分（共 100 分）

1） 精度

确保双精度数据类型计算正确，如果错误，总分直接计 0 分。

2） 速度（60 分）

根据参赛队伍的数量排名，每个排名之间相差 5 分。不同测试用例的权重不同，综合速度第一名为 60 分，以第一名的速度为基准，其它参赛队作品的速度比这个基准慢 10%以内的得 55 分，以此类推，20%以内的得 50 分，30%以内的得 45 分，40%以内的得 40 分，50%以内的得 35 分，60%以内的得 30 分，70%以内的得 25 分，80%以内的得 20 分，90%及以上的得 10 分。

3） 内存消耗（30 分）

对同等测试用例下不同参赛作品的内存消耗量做归一化，按照与最优值的差距比例进行评分。综合使用内存量最小的作品得满分 30 分，以此为基准，其他参赛队作品的内存消耗量每比基准多 10%，则减 1 分，直到扣完 30 分为止。假如基准为 100MB，则内存消耗量为 110MB 的得 29 分，内存消耗量为 150MB 的得 25 分，内存消耗量为 200MB 的得 20 分，内存消耗量为 300MB 的得 10 分，内存消耗量为 400MB 及以上的得 0 分。

4） 创新项（10 分）

依据报告中加速算法创新点的数量，以及对未来改进的展望进行评分。

第二届（2020年）集成电路EDA设计精英挑战赛
赛题指南

一、赛题七：图分割算法（数字集成电路设计方向）

二、命题单位：思尔芯（上海）信息科技有限公司

三、赛题背景

随着计算技术的发展，大数据时代的到来，超大规模图的分割问题越来越引起人们的关注，并被广泛应用于大数据处理的各个领域，如分布式计算问题划分、推荐策略、布局等等。

典型应用有超大规模数字集成电路仿真验证中的多 FPGA 系统分割，通过不同的分组权重将图分割成若干分组，进而实现快速高效的系统验证。

四、赛题描述

对给定的数字集成电路门级电路的网表文件，按照分组权重约束条件 - PIO / LUT / FLIPFLOP / CONNECTION / FIX-ASSIGN，对网表进行不同模式的 Partition 算法分割，分割成若干个分组。在保证整个图拓扑结构不变和满足约束条件的情况下，确保每个分组之间互联线数目最少。

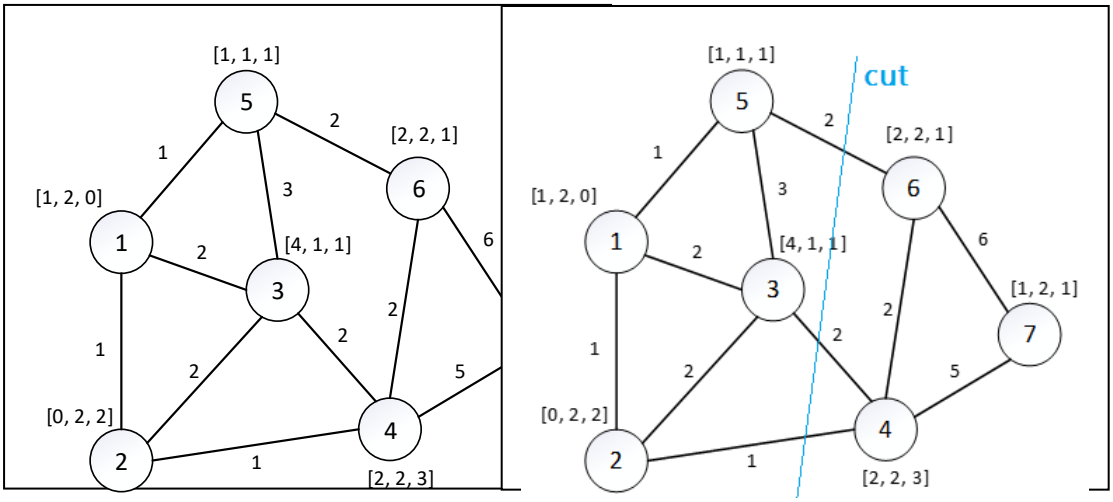
涉及知识点：

- 门级电路的图论建模
- 最小割相关 min-cut 算法

数学原理：

Partition 算法的目的是对一个带顶点权重与连接权重的图，根据设定的分组权重约束条件，进行不同的切割尝试，最后获取到一个最小的连接权重值的切割过程。

例如：对左下图进行切割，约束条件要求分为 2 个分组，每个分组资源权重不限，一个分组包含节点 5，另一个分组包含节点 4， 则最优的切割结果为右下图， 互联线权重为 5 < 2+2+1 >。



技术评分点：

- 不同权重约束条件和不同工作模式下分割结果的正确性（主办方提供检查程序）
  - 满足资源约束
  - 网表等价（节点互连结构等价）
- 分割结果的最优化
  - Cut size（互联线权重）最小
- 满足分割结果正确的前提下，在同一平台（主办方将给出操作系统版本和编译器版本）上的运行时间，即所需的 CPU 算力
- 内存使用量
  - 使用 top 命令或工具 memtime
- 附加 Benchmark 测试（隐藏测试）的完成度

注：参赛队伍提供技术报告（含时间空间复杂度分析）、可执行程序与源代码

五、评分标准：

技术评分，共 100 分。

1. 基础任务 – 60 分 。（Benchmark 测试的完成度占比 70%，结果正确性、最优化、运行速度、内存使用量指标占比 30%）

测试项	测试描述
testdata-0	简单的 2-FPGA 切割，较少的节点(20）和连线（40）
testdata-1	小型的 2-FPGA 切割，少量的节点(1000）和连线（1000）
testdata-1-e	针对 testdata-1 的错误权重约束的检测
testdata-2	大型的 2-FPGA 切割，大量的节点(12000）和连线（14000）
testdata-3	大型的 2-FPGA 切割，大量的节点(12000）和连线（33000）
testdata-4	大型的 4-FPGA 切割，大量的节点(12000）和连线（14000）
testdata-5	大型的 8-FPGA 切割，大量的节点(12000）和连线（14000）

2. 高性能任务 – 40 分。（Benchmark 测试的完成度占比 70%，结果正确性、最优化、运行速度、内存使用量指标占比 30%）

测试项	测试描述
testdata-4-cnn	大型的 4-FPGA 切割，基于可变的 FPGA 之间的互联关系，大量的节点(12000）和连线（14000）
testdata-4-mean	大型的 4-FPGA 平均值切割，基于可变的 FPGA 之间的互联关系
testdata-5-ffd	大型的 8-FPGA 切割，基于节点之间的 FlipFlop 驱动规则，大量的节点(12000）和连线（14000）
testdata-5-clk	大型的 8-FPGA 切割，基于节点之间的 CLK 关系，大量的节点(12000）和连线（14000）
testdata-6	200,000 节点范例的切割

说明：相关 FAQ 解答指南请联系 eda_ec@s2cinc.com。

附件：赛题接口文件说明

5.1 输入文件

输入文件包含当前图(Graph)里面所有节点(Node)，连线(Net)，权重(Weight)和分组(Group)的资源信息，要求参赛者的分割工具根据这些信息对图中的节点进行切割，根据实现算法的最优结果，生成输出若干个小图.

- **Node definition file <节点定义文件 design.are>**

每个节点名称以字母 g 和一个不重复的数字组成，
每行表示一个节点的资源信息，包含 10 种资源，每种资源权重以 10 进制数值表示。
<node> PIO INT FF LUT BUFG TBUF DCM BRAM DSP PPC [timing-property-list]

各个资源的定义如下表。

Resource	*Description*	*Comment*
资源 1	PIO	PIO
资源 2	INT	INT
资源 3	FF	Flipflop, Latch
资源 4	LUT	LUT2, LUT4, LUT6, LUT8
资源 5	BUFG	BUFG
资源 6	TBUF	BUFR, BUFT
资源 7	DCM	PLL, DLL
资源 8	BRAM	RAMB36, RAMB128
资源 9	DSP	DSP, ARM
资源 10	PPC	PPC

例如，
常规节点文件，

g0 0 1 200 0 0 0 0 0 0 0
g1 0 1 200 0 0 0 0 0 0 0
g2 0 1 200 0 0 0 0 0 0 0
g3 0 1 200 0 0 0 0 0 0 0
g4 0 1 200 0 0 0 0 0 0 0
g5 0 1 200 0 0 0 0 0 0 0

带 timing 属性的节点文件，

g0 0 1 200 0 0 0 0 0 0 0 {ff}
g1 0 1 200 0 0 0 0 0 0 0 {c0}
g2 0 1 200 0 0 0 0 0 0 0 {c2}
g3 0 1 200 0 0 0 0 0 0 0
g4 0 1 200 0 0 0 0 0 0 0 {ff c2}
g5 0 1 200 0 0 0 0 0 0 0 {ff c2c3}

其中， *[timing-property-list]* 为可选性，里面属性定义如下，
ff *节点为 ffd 类型*

c0　　节点含有名为 c0 的 clk 属性
c2c3　　节点含有名为 c2 和 c3 2 个 clk 属性

- **Net definition file <连线定义文件 design.net>**

每个连线信息由 2 个或更多节点组成，一个为驱动节点(driver)，其他为负载节点(load),
每行表示一个连线的部分信息，格式如下，

<node> <s/l> [weight]

s　　含有驱动(driver)节点的连线部分
l　　含有负载(load)节点的连线部分，一个连线可能含有一个或多个负载部分
weight　　连线的权重值，可选

例如,
g1 s 1
g0 l
g0 s 1
g2 l
g1 l

- **The FPGA group resource list file <FPGA 资源文件定义 design.info>**

Fpga 分组的资源定义与节点资源一致，每一行表示一个分组的资源最大权重值信息和分组的序号。
格式如下，
<FPGA> PIO INT FF LUT BUFG TBUF DCM BRAM DSP PPC [int-list]

其中，[int-list] 为可选性，
采用 list 列表表示，每个分组的序号(从 1 开始)，对应列表中相应的位置元素(从第 1 个元素开始)。列表里面的值表示当前分组与其他分组的互连线约束权重值。
每个分组的内部互联权重值为 0，不用考虑。
第 2 列 INT 的值是最后 list 的和，如下 800 = 0+200+200+400

分割算法必须根据分组资源信息对图中节点进行分割，输出的结果不能够超出每个分组的资源权重值。

例如 2 个分组，
FPGA 576 890 427200 427200 24 2048 24 55562240 0 0

FPGA 576 890 427200 427200 24 2048 24 55562240 0 0

4 个分组，

FPGA 576 800 427200 427200 24 2048 24 55562240 0 0 { 0 200 200 400 }
FPGA 576 800 427200 427200 24 2048 24 55562240 0 0 { 200 0 100 500 }
FPGA 576 900 427200 427200 24 2048 24 55562240 0 0 { 200 100 0 600 }
FPGA 576 1500 427200 427200 24 2048 24 55562240 0 0 { 400 500 600 0 }

- **The fix or preassigned node list file <预先分配的节点定义文件 design.fix>**

预先定义的节点分配文件，一行一个或多个节点在指定分组里面的分配信息。
分割算法必须根据节点预分配信息对图中节点进行分割，输出的结果满足所有预分配的节点分组信息。

注： 这个是优先级最高的规则，分割分组结果必须满足此文件预定义的分组

格式如下，
<FPGA TYPE mm>: <node> <node> ...

mm 分组编号是从 1 开始的正整数，不一定是连续的数字定义.

例如，
FPGA TYPE 1: g0
FPGA TYPE 2: g1

例如，编号不连续的预分配定义，

FPGA TYPE 1: g0
FPGA TYPE 2: g1
FPGA TYPE 3: g3013
FPGA TYPE 4: g6026
FPGA TYPE 7: g30
FPGA TYPE 8: g40

- **The cut mode <分割模式>**

目前定义的分割算法的工作模式，

--fix-mincut 默认分割模式，所有的资源权重都是固定不变的，要求分割算法能够在不超出分组资源权重的前提下算出最优的结果，且所有分组

的对外互联数目总和最小。

--int-mincut 同 *--fix-mincut,* 此外，额外要求分割算法满足每个分组之间的互联权重约束，得出最优解，且所有分组的对外互联数目总和最小。

--ffd-mincut 同 *--fix-mincut,* 此外，额外要求分割算法满足每个分组 flipflop 驱动规则的情况下，得出最优解，且所有分组的对外互联数目总和最小。

Flipflop 驱动规则如下:

1. *每个可分割的节点必须为带有 ffd 属性的节点，或者其他节点*

   仅当它所有的驱动连线都为带 ffd 属性的节点。
2. *每个分组的节点必须由其他分组的 ffd 节点驱动.*

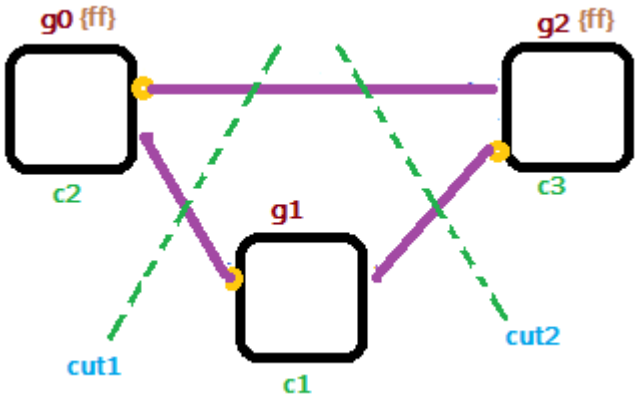


Figure 1 正确的 FFD 切割示意图

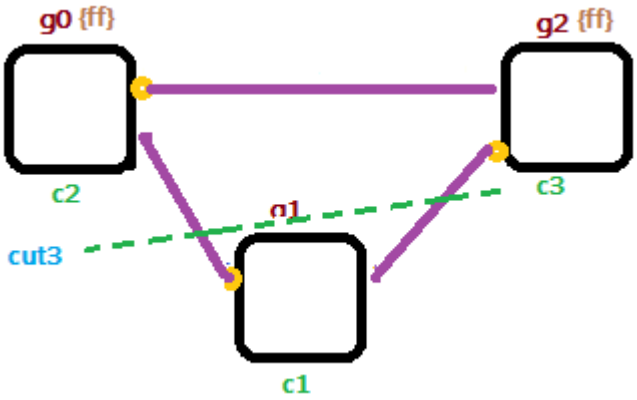


Figure 2 错误的 FFD 切割示意图

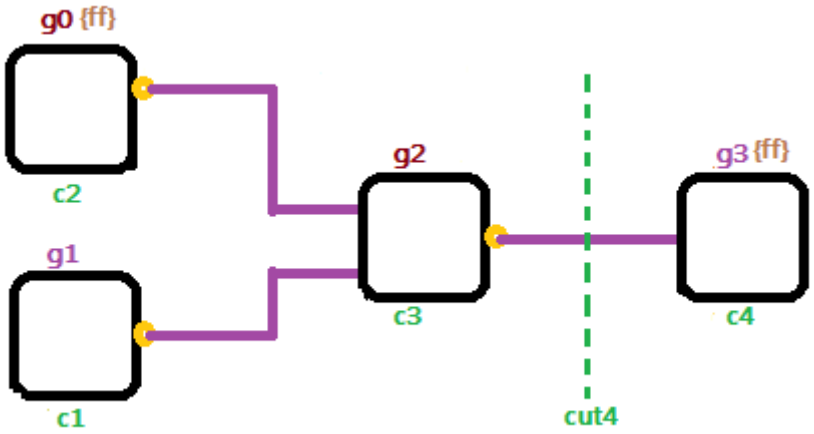


Figure 3 错误的 FFD 切割示意图

--clk-mincut
--num <n>
--clks <candidate_clk_list>

同 *--fix-mincut*，此外，额外要求分割算法满足每个分组间出现的节点关联的 clk 最大数目或（和）合适名称的条件下，得出最优解，且所有分组的对外互联数目总和最小。*--num/--clks* 为可选项。没有指定时要求每个分组间出现的节点关联的 clk 最大数目最小。

clk 规则如下:

每个不带 clk 属性的节点可以继承所有驱动它的最近的带 clk 属性节点的 clk 属性值。

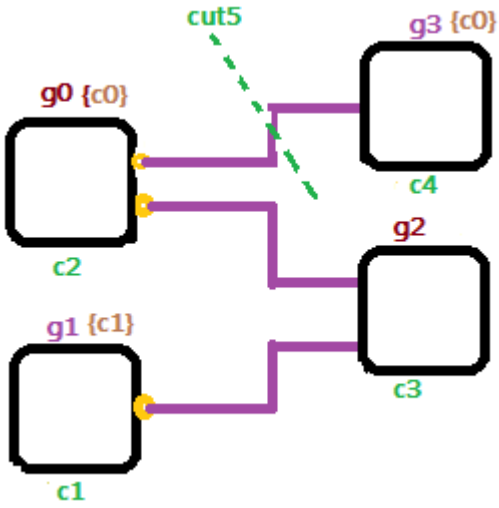


Figure 4 正确的 CLK 分割示意图

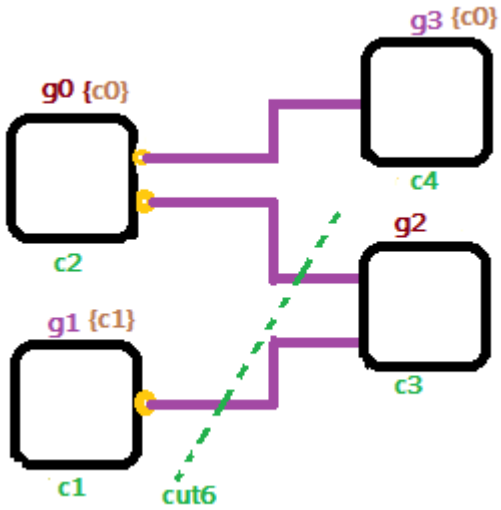


Figure 5 错误的 CLK 分割示意图

--mean-mincut
--percentage

同 *--fix-mincut*, 此外，额外要求分割算法满足每个分组间互联线与资源权重值的比例在给定的均值误差的范围内，得出最优解，且所有分组的对外互联数目总和最小。
--percentage 为可选项，没有指定时默认值为 20.

5.2 输出文件

输出文件包含对图中所有节点进行分割后的最优分割结果和报告信息。

- **The output partition result file <分割结果文件 design.output>**

  分割结果文件，由分割算法根据当前的分割策略和模式运算得出的最优结果。
  每一行包含一个分组信息里面包含的节点列表，以 'FPGA+序号 分组+mm' 字串开始，每个节点之间以空格隔开，每行建议最大包含 20 个节点，可以多行表示。

  格式如下,
  <FPGAnn TYPE nn>: <node-list>

  例如,
  FPGA1 TYPE 1 :g0 g1 ..... g19
  g20 g21 gp0
  FPGA2 TYPE 2 :g100 gp1 gp2

- **The output partition report file <分割结果报告文件 design.rpt>**

分割结果报告文件，包含分组的实际资源权重使用值，以及每 2 个分组之间的互联信息。
只有 2 分组的情况下，{int-list} 不需要。
格式如下,
<FPGAnn TYPE nn>: <resource-list> [int-list]
{int-list} 采用 list 列表表示，每个分组的序号(从 1 开始)，对应列表中相应的位置元素(从第 1 个元素开始)。列表里面的值表示当前分组与其他分组的互连线约束权重值。

每个分组的内部互联权重值为 0，不用考虑。

例如,
2 个分组的报告文件,
FPGA1 TYPE 1: 1 401 109 1814 0 0 0 0 0 0
FPGA2 TYPE 3: 2 401 0 4000 0 0 0 0 0 0

4 个分组的报告文件件，
FPGA1 TYPE 1: 1 400 109 1814 0 0 0 0 0 0 { 0 200 50 150 }
FPGA2 TYPE 2: 2 350 0 4000 0 0 0 0 0 0 { 200 0 50 100 }
FPGA3 TYPE 3: 1 200 109 1814 0 0 0 0 0 0 { 50 50 0 100 }
FPGA4 TYPE 4: 2 350 0 4000 0 0 0 0 0 0 { 150 100 100 0 }

- **关于 FPGA 编号和 TYPE 编号**

FPGA 后面的编号 n 仅仅表示一个处理序号，不是分组号。一般情况下代码处理时 fpga 后编号和 Type 后分组号一致。比如 design.output， design.rpt 文件格式。如果不一致，以 TYPE 后编号为准（分组号）
如果 fpga 后无编号，以 TYPE 后编号为准，作为分组号。比如 design.fix 格式。

FPGA TYPE 1: g0
FPGA TYPE 2: g1
FPGA TYPE 3: g3013
FPGA TYPE 4: g6026
FPGA TYPE 7: g30
FPGA TYPE 8: g40

第二届（2020年）集成电路EDA设计精英挑战赛
赛题指南

一、 赛题八：FPGA 芯片的布局合法化

二、 命题单位：上海安路信息科技有限公司

三、 赛题背景：

可编程逻辑模块（PLB）是 FPGA 芯片的基本单元，FPGA 芯片的主要部分是由 PLB 和相对较少的其他模块（如 IP）组成的阵列。PLB 除了可以实现基本的组合逻辑查找表和时序逻辑功能外，还提供了专用的快速进位链，以执行快速算术加法和减法。例如，上海安路的 PH1 系列 FPGA 芯片的每个 PLB 包含 8 个 SLICE 单元，可以实现 8 个独立的 LUT6 查找表逻辑，也可以实现一个 8 比特位的进位链，并且可以级联纵向相邻的 SLICE 实现更宽比特位算术逻辑。上述宽比特位算术逻辑，组成了一个逻辑进位链 Macro，在芯片自动化设计流程的后端布局时，需要连续占用纵向相邻的若干个 SLICE。

解析式方法是目前工业界常用的集成电路布局方法之一。在采用解析式方法进行全局布局后，器件布局相互之间有可能存在着重叠，布局结果是不合法的。因此需要设计一个布局合法化算法，在合理的运行时间里，根据全局布局返回的初始布局位置将包括快速进位链 Macro 在内的所有器件的布局合法化（这里将问题简化为，所有器件

布局相互之间没有重叠），同时尽可能保持全局布局的结果质量（即所有器件的合法化布局位置要尽量保持初始位置）。

四、 赛题描述：

给定 FPGA 芯片包含 M 行 N 列的 PLB 阵列，每个 PLB 为宽、高都为 8 的正方形。每个 PLB 中包括 8 个纵向放置的 SLICE,假设每个 SLICE 的宽、高分别为 8 和 1。给定 n 个普通逻辑器件和进位链 Macro（所有器件的宽度相同为 8，普通逻辑器件的高度为 1，进位链逻辑 Macro 的高度大于 1），以及每个器件的初始布局位置（Xi Yi）（器件之间存在适度重叠）。求解器件的合法化布局位置（XXi YYi），满足下列条件（1）器件布局之间不存在重叠，并且（2）最小化器件布局移动的距离平方和：

$$S = \sum_{i=0}^{n} Wi\,((XXi - Xi)^2 + (YYi - Yi)^2)$$

式中的 Wi 是器件 i 的高度。

上海安路赛前将提供若干典型测例用于程序调试，以及网页来实时评测程序的结果质量（不公开）。测试用例的输入输出文件格式描述如下：

4.1 输入文件 case.in

文件第一行是 PLB 阵列的行列数 M 和 N，和器件总个数 n（1 <= n <= M*N）。之后的每一行是每个器件的高 Wi（对进位链 Macro，1 < Wi <= 1000；对于其他器

件，Wi = 1)，以及左下角的初始坐标 Xi Yi（0 <= Xi < N，0 <= Yi < M)。

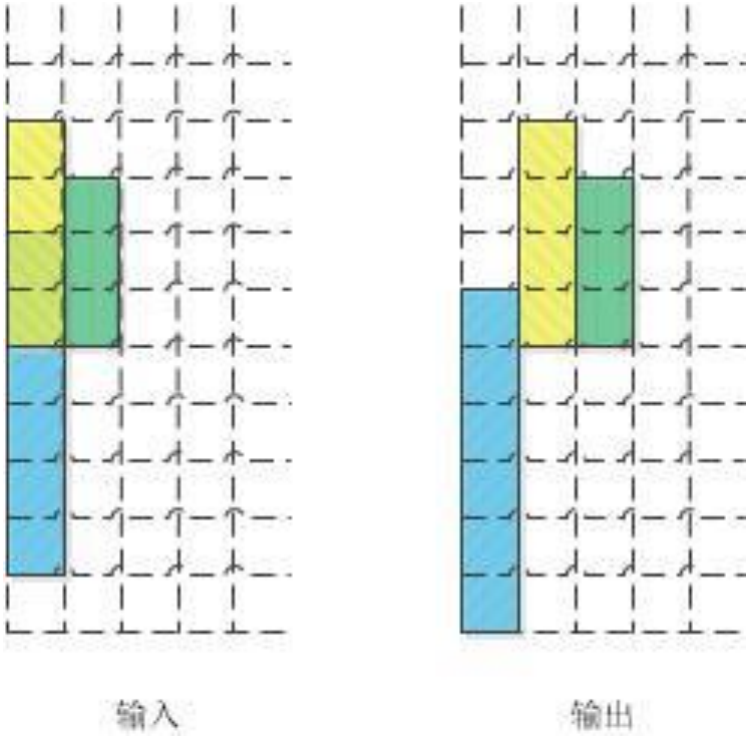
4.2 输出文件 case.out

文件的每一行是按照输入文件中的给定顺序，每个器件的合法化布局坐标 XXi YYi（0 <= XXi < N，0 <= YYi < M)。

4.3 样例

样例输入 case1.in:
160 80 3
48 0 8
32 0 40
24 8 40

样例输出 case1.out:
0 0
8 40
16 40



五、评分标准

技术评分标准主要包括程序质量评分,源代码和设计报告评分两部分。

1. 程序质量评分（80 分）

（1） 基本要求

程序在合理运行时间(单线程)和内存，正常运行结束，输出格式正确合法，所有逻辑器件和进位链 Macro 的布局合法。

a) 运行时间上限:程序运行时间超出 10m，认为程序死循环，程序质量得分为 0 分。程序运行时间上限会根据服务器环

境和测例，有所调整。

b) 运行内存上限：运行内存超过 10G，程序质量得分为 0 分。程序运行内存上限会根据服务器环境和测例，有所调整。

c) 优化目标上限：程序的优化目标值 S 超出最优结果（best_S）20%以上，程序质量得分为 0。

（2） 程序质量

要求竞赛方综合考虑优化目标和运行时间来选择算法。根据竞赛程序的优化目标值 S 和运行时间 T，程序质量度量值 Q 计算如下：Q = S * (1 + T_factor)，T_factor = 0.05 * log_2(T/median_T),并且 T_factor 的范围限定在[−0.1，0.1]，即根据运行时间有最大 10%的结果质量调整。

所有竞赛程序中程序质量度量最小 best_Q，得到满分（80 分）。其他竞赛程序根据其程序质量度量值 Q，计算得分如下：80 − 160*(Q/best_Q − 1)。

2. 源代码和设计报告评分（20 分）

（1） 对于源代码，将对如下方面进行考察：（10 分）

a) 代码风格是否优美，组织架构是否清晰，可读性是否良好

b) 有无良好的模块化设计

c) 命名风格是否良好

（2） 设计报告行文要求条理清楚，详略得当，清楚易读，内容应该包括以下几个方面：（10 分）

a) 算法原理，测试结果

b) 时间/空间复杂度分析

c) 代码设计上其他亮点（如果有），比如架构设计，模块复用