第三届（2021）集成电路 EDA 设计精英挑战赛
赛题指南

一、**赛题九**：Schematic 布局布线

二、**命题企业**：上海安路信息科技股份有限公司

三、**赛题 Chair**：邸志雄　　西南交通大学信息学院电子系副系主任

四、**赛题描述**：

随着 FPGA 电路设计的复杂度的提升，设计者们越来越关注高层次的设计方法。为了提高在设计过程中的交互性，schematic（逻辑电路图）的使用是至关重要的。传统上，设计者们根据 EDA 工具的反馈结果手动绘制逻辑图，这将导致大量时间和精力的浪费，甚至容易导致错误。Schematic 的自动绘制工具不但可以根据电路网表快速、准确地绘制出相应的逻辑图，从而帮助设计人员理解电路，缩短电路设计开发周期，还可以作为电路设计的开发文档留存。

Schematic 具体是对 instance 和 net 进行绘制，其中 net 之间的交叉线数量是个重要的指标。为了让 schematic 看起来简洁规整，需要通过合理的布局布线来减少交叉线数量。

图 Figure 1 为安路的 TD 工具中的 schematic。

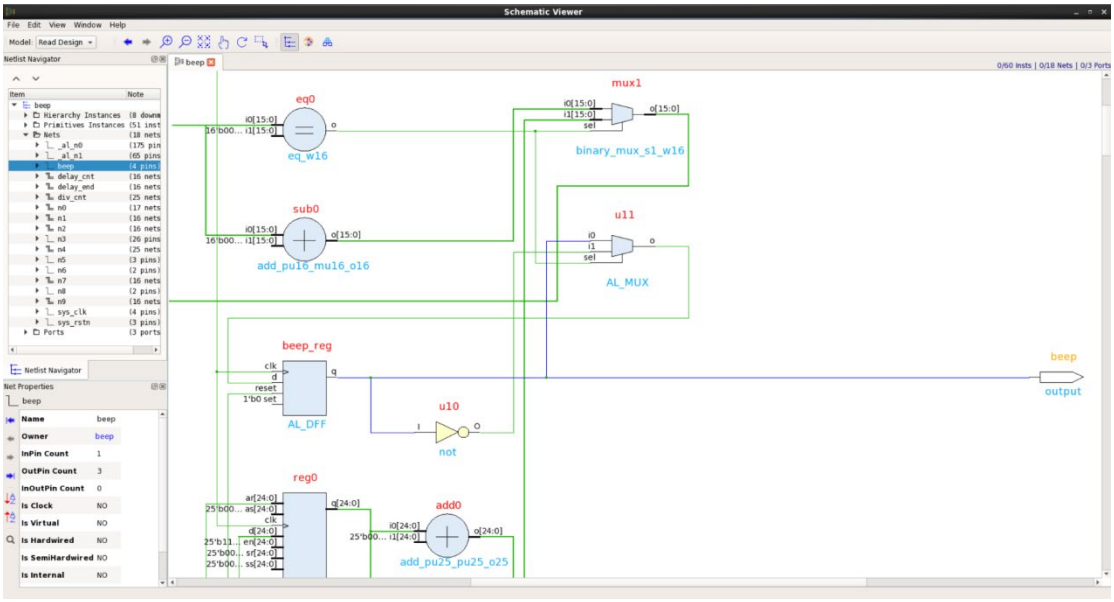

Figure 1

算法推荐

1. 确定 instance 的列

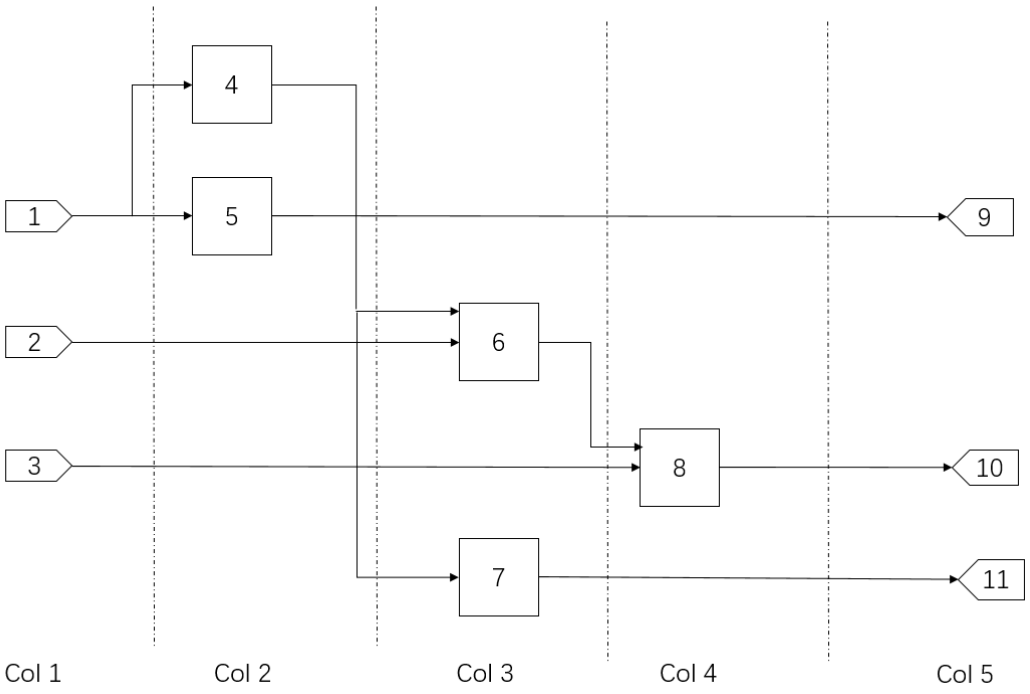

Figure 2

对所有 instance 的入度和出度（输入和输出的数量）进行统计，入度为 0 的是输入端口，作为第一列。出度为 0 的是输出端口，作为最后一列。如图 Figure 2 所示，1，2，3 为输入，9，10，11 为输出。

从一个输入端开始，把其输出的 instance 放在第二列，再从这些 instance 出发，把下一级输出放在第三列，依次直到输出端口。对其他输出同样处理，如果遇到之前已经处理过的 instance，则比较列数，取两者较大的作为列数。如 Figure 2 所示，从第一列的 1 开始，4 和 5 都是第二列，6 和 7 都是第三列，9 作为输出是最后一列，但具体列数目前还不能最后确定。再从 2 开始，6 是下一级输出，算是第二列，但在之前已经作为第三列，所以取较大的，作为第三列。8 是再下一级，是第四列。10 是最后一列。最后从 3 开始，8 是下一级，但已经是第四列了，10 是最后一列。最后统计出总列数为 5 列，所有输出都是第 5 列。

注意实际电路中还有可能出现几个 instance 输入输出形成环状结构，这时候需要从一处截断处理，连线会出现从列数较高的 instance 指向列数较低的 instance。

2. 确定 instance 所在行

在确定了 instance 所在列以后，还需要确定每一列上的 instance 所在的行。对于一列上的 instance 来说，它们的排列顺序直接影响到了连接 net 的交叉点数量。

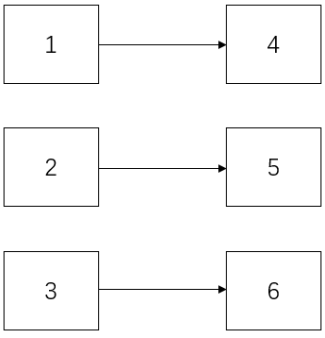

Figure 3

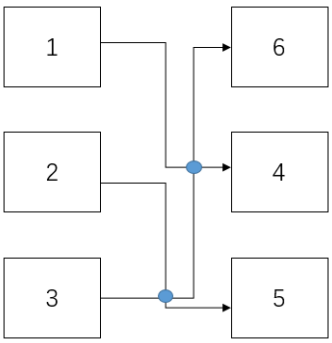

Figure 4

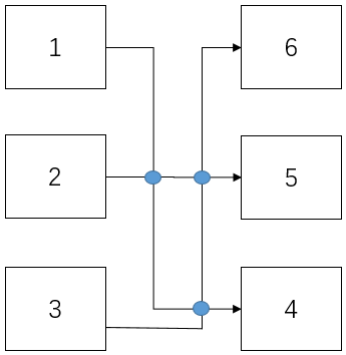

Figure 5

为了说明顺序影响交叉点数，这里举个例子，1 与 4，2 与 5，3 与 6 之间有连接关系。Figure 3 为最佳情况，没有交叉点。Figure 4 次之，有 2 个交叉点。而 Figure 5 为最差情况，两两之间都有交叉，交叉点数为 3。

给定每一列中的 instance 排序，则交叉点数就能计算出来，但需要对所有 instance 的连接进行计算，计算量比较大。并且如果通过穷举每一列的排序来找到最优解，计算量会进一步增加。这种 NP 问题，找到最优解的计算复杂度很高，一般会采用一些方法牺牲一定的质量，用更少的计算量找到次优解。

由于这部分是整个试题的难点，这里仅仅给出可供参考的思路。由于计算具体的交叉点数量计算量大，可以找一个其他的参数来进行优化，可以间接达到减少交叉点的结

果。比如，上面的例子中，对应的连接都在同一行的时候交叉点最少。如果有多个连接，并且与多个列都有连接的时候，考虑这些行的平均数，或者以一定方式加权平均，使其更接近对应的 instance 所在的行数，这样可能可以减少交叉。另外，可以加入一些操作，比如交换两个 instance 的位置，来进行多次迭代，使得每一次迭代的结果更优化，也是常用的方法。

输入：

1. inst.json

   Key: instId

   Value: [numIn, numOut, numInOut]

   numIn，numOut，numInOut 分别是这个 instance 的输入口，输出口，双向口的数量。numIn 为 0 的即为输入，numOut 为 0 的即为输出。

   如 Figure 1 所示，名为”beep_reg”的 instance，输入格式为：

   “2”：[4, 1, 0]

   “2”是 id 号，数组表示有 4 个输入口，1 个输出口，0 个双向口。

   名为“beep”的输出端口，输入格式为：

   “61”：[1, 0, 0]

   “61”是 id 号，数组表示只有 1 个输入口。

2. net.json

   [[driIndex driPort sinkIndex sinkPort]，...]

driIndex 是 net 的源端 instId，driPort 是源端输出口的 id。sinkIndex 是漏段 instId，sinkPort 是漏端输入口的 id。Port 的 id 表示着从上到下的排列顺序。

对于存在多个 driIndex driPort 相同的情况，说明是总线，即该端口会输出多位宽信号。

同样以 Figure 1 中的”beep_reg”为例，其输出端口只有一个，连接到了 3 处。输入格式如下：

[2, 1, 9, 1]

[2, 1, 10, 1]

[2, 1, 61, 1]

其中“2”是”beep_reg”的 id 号，“1”是输出端口的 id 号。“9”，“10”，“61”分别是“u10”，“u11”和“beep”的 id 号，“1”是它们的输入端口的 id 号。

输出：

1. inst_out.json

Key: instId

Value: [x, y]

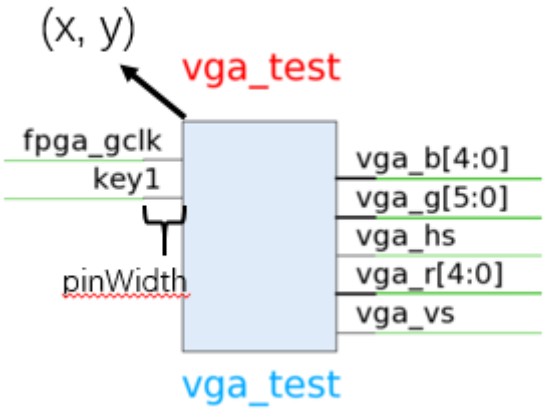

Figure 6

如图 Figure 6 所示，x, y 是 instance 矩形框左上角的坐标，instance 的输入和输出需要保留 pinWidth 宽度的距离，用来画输入输出管脚。对于只有输入或者输出的 instance，即为 schematic 的输入输出，就不需要保留。这里 pinWidth 的值固定为 2。Instance 的矩形框等宽，宽度为 8。对于有输入输出的 instance，高度为 2*(N+1), N 为输入数量和输出加双向口数量中较大值，输入口在左侧，双向口和输出都在右侧。对于只有输入或者只有输出的，高度固定为 2。对于 pin 的位置，只要不超过矩形高度即可。

考虑到规整性，instance 按列对齐，即同一列的 instance 的 x 坐标一致。并且所有输入在最左边一列，所有输出在最后边一列。每列 instance 矩形框之间高度的间隔不能小于 4，避免造成拥挤的现象。

Figure 1 中的”beep_reg”，输出格式为：

“2”：[58, 55]，“58”和“55”分别为矩形框左上角的 x 坐标和 y 坐标。

实际上占据的矩形框坐标为[58–2, 55, 58+8+2, 55+2*(4+1)]，坐标依次为左上角

x 坐标，左上角 y 坐标，右下角 x 坐标，右下角 y 坐标。这个矩形框的意义是不让 net 连线跨过 instance，否则会显得比较杂乱。

2. net_out.json

Key: “driIndex driPort sinkIndex sinkPort”

Value: [“llx lly urx ury”,...]

Port 的 id 从 1 开始计数。每条 net 由一组从输出到输入顺序的线段组成。即从点(llx, lly)到(urx, ury)。线段宽度为 1，只能是竖线或者横线，不能有斜线。并且同一 net 的相邻线段要能首尾相连。整条 net 的起点是对应源端 instance 的输出 pin 的位置，终点是对应漏段 instance 的输入 pin 的位置，这里需要注意算上 pinWidth。如果 instance 的坐标为（lx，ly），则其源端的 net 线段的起点 x 坐标为(lx+8+2)，8 为 instance 宽度，2 为 pinWidth，而其漏端的 net 线段的终点 x 坐标为(lx-2)。

Figure 1 中”beep_reg”与“u10”的连接输出如下：

“2 1 9 1”:[“68 58 74 58”，“74 58 74 68”，“74 68 82 68”]

其中“2”是”beep_reg”的 id，“1”是输出端口的 id，“9”是”u10”的 id，”1”是输入端口的 id，后面数组中为 3 条线段，4 个数字表示起始点的 x, y 坐标和终点的 x, y 坐标。这三条线段都是首尾相连的。

3. schematic.jpg

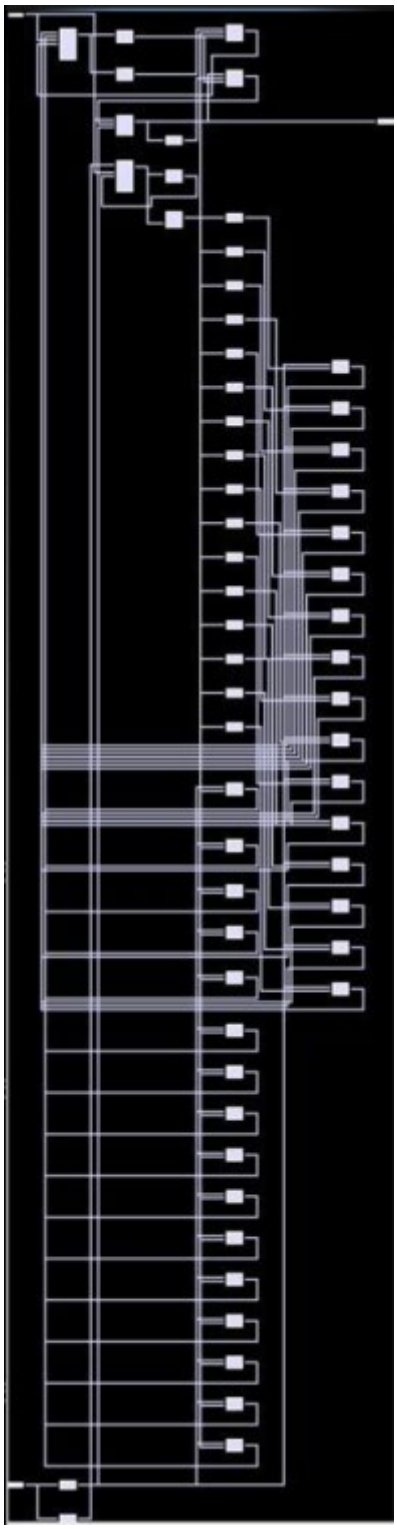

Figure 7

如图 Figure 7 所示，绘制出 instance 和 net 的 schematic，可以用来检查算法是

否合理。

正确性：

1. Instance 互相不会交叠，这里 instance 不止包括画出的矩形，还包括 pin 的引脚。
2. Net 的连接性，同一条 net 是连续的线段，并且首尾要在 instance 的 pin 的合法位置上。

四、性能评估：

1. 交差点数量

结果的交叉点计算是所有 net 的线段相交点的数量。对于同一条 net，折线段的拐点也包括在交叉点内。越少越优。

2. 总面积

包含 Instance 和 net 的最小矩形范围的面积。越小越优。

3. 行列之和

Instance 所在位置的行列之和。相同 x 坐标的 instance 认为是同一列，相同 y 坐标的 instance 认为是同一行。越小越优。

4. 运行时间

产生输出所需的时间。越短越优。但考虑到编程语言的运行时间不好比较，可以根据多个不同规模的电路估算出复杂度。每个电路都有运行时间上限，需要保证在限定时间内完成。

5. 美观程度

设立额外的奖项，对美观性进行评价。

五、评分方法：

分数分为两部分，主体是对交叉点数量的评分。对于每个评测电路，都有标准数值，评分为 100X（标准数值/实际数值），100 为上限。对总面积，行列之和采取排名制度，每一项对于前 10 名参赛作品给与名次 n 为（11-n）分的附加分，10 名以后没有分数。由于评测会给出多个电路，所以按总分作为最终分数。运行时间复杂度和美观程度不作为具体评分，但会在答辩环节体现。

为了参赛者能够更好地测试，会提供若干个评测电路。最后评分会在此基础上另外加上几个电路一起打分。

六、注意事项：

1. 原则上算法里允许使用第三方库，但需要在答辩时进行说明，以便评估工作量。
2. 美观程度主观性比较强，不影响具体分数评估，属于是额外设立的奖项。