集成电路设计中的时钟网络优化

摘要

随着集成电路技术的迅速发展，时钟信号作为数据传输的基准，在同步数字系统中的性能和稳定性上扮演着关键角色。本研究综述了数字集成电路中时钟网络设计的当前方法及其局限性，特别是传统的基于树的时钟分配方法。随后，介绍了如广义 H 树（GH-tree）等新型时钟树设计方法，这些方法利用改进的拓扑结构和动态规划算法，优化了时钟网络的偏差、延迟与功率平衡。此外，本文还探讨了采用最小费用流和局部聚类技术在非均匀布局下优化时钟网络设计的可行性和有效性。通过理论分析与实验验证，展示了这些方法在提高设计鲁棒性和减少功耗方面的潜力。研究指出，虽然新方法优化了网络性能，但高复杂度和成本以及实际应用的局限性是未来研究需要重点解决的问题。

关键词：时钟树综合；广义 H 树；动态规划；最小费用流；

1 引言

在数字集成电路中，时钟信号是数据传输的基准，它对于同步数字系统的功能、性能和稳定性起决定性作用，所以时钟信号的特性及其分配网络尤被人们关注。时钟信号通常是整个芯片中有最大扇出、通过最长距离、以最高速度运行的信号。时钟信号必须保证在最差的条件下，关键的时序要求能得到满足，否则对时钟信号任何不当的控制都可能导致紊乱情况，将错误的数据信号锁存到寄存器，从而导致系统功能的错误。

在现有基于标准单元库的 ASIC 流程中，通常将扇出较大的时钟网络在综合阶段设置为理想时钟，在物理设计阶段进行时钟树综合（clock tree synthesis, CTS），因此它是后端物理设计的关键步骤之一。

时钟信号在物理设计中的实现结果被形象地称之为时钟树，时钟信号的起点叫做根节点(root pin)，时钟信号经过一系列分布节点最终达到寄存器时钟输入端或其他时钟终点（例如，存储器时钟输入端）被称为叶节点。根节点、分布节点和叶节点都依附于的逻辑单元则分别称作根单元、分布单元和叶单元。时钟网络从根节点逐级插入驱动器（buffer），从而到达其叶节点，按照芯时钟网络的约束要求产生时钟树的过程叫做时钟树综合。即生成从时钟源（root）到所有寄存器（FF）的时钟端口（sink pin）的时钟树。该过程如图 1所示。

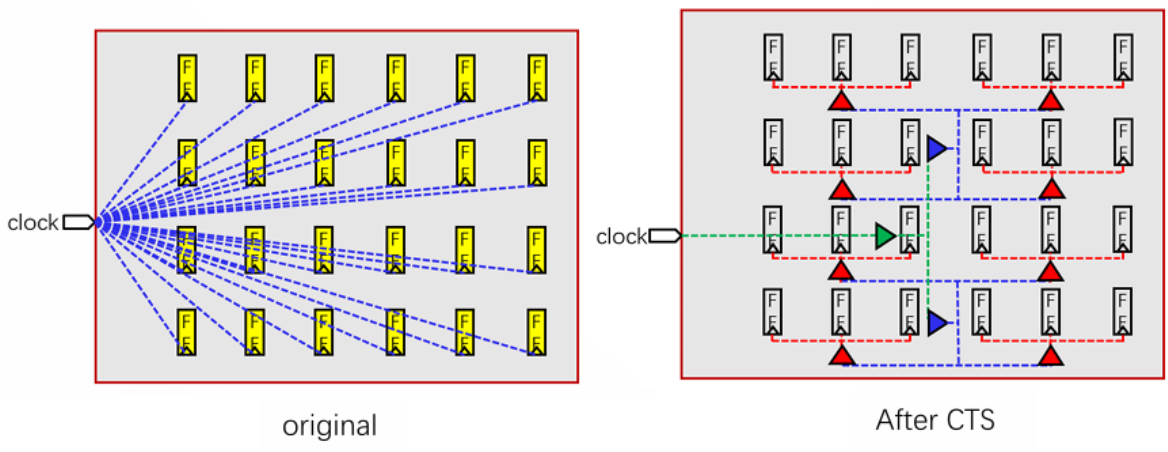


图 1. 时钟树综合示意图

时钟树根据其在芯片内的分布特征，可分为多种结构，主要有 H 树（H-tree）、X 树（X-tree）、平衡树（balanced tree），以及梳状或脊椎状时钟网（clock tree mesh：comb or spine）。它们的拓扑结构图如图 2所示。

H-tree 从中心点到达各个叶节点的距离是相等的，因此信号到达叶节点的时间理论上相等，时钟偏差理论值为 0，该结构的不足是布线比较困难，扇出难以协调一致，适用于较小的设计。

X-tree 也是一种实现等长互连线的方法，与 H-tree 相比，X-tree 采用了很多非 90° 的互连线，与 H-tree 的扇出为 2 相比，X-tree 的扇出是 4。

为了减小叶节点反射电流的影响，将分支后的时钟网线的电阻提高到分支前的 2 倍，也即分支后的线宽为分支前的 1/2，得到“裁剪状的”（tapered）H-tree。

平衡树采用的是两个层次的驱动，比较适用于分层次的时钟树设计以及较大的时钟树综合。在顶层上，它采用一个层次的驱动插入以平衡时钟偏差，到模块级再采用模块级的驱动插入平衡时钟偏差。从图中可以看出从时钟的根节点通过一级驱动到达模块的时间是不同的，存在时钟偏差，但是比较小。

对于很大的时钟树，用时钟网络或时钟网格（clock mesh, clock grid）的方法是获取较小时钟偏差较好实用方案，如图所示。尽管当代的 EDA 工具能自动生成时钟网络或网格等拓扑结构，但它还是需要很丰富的经验和细致的手动方法去设计并作调整。

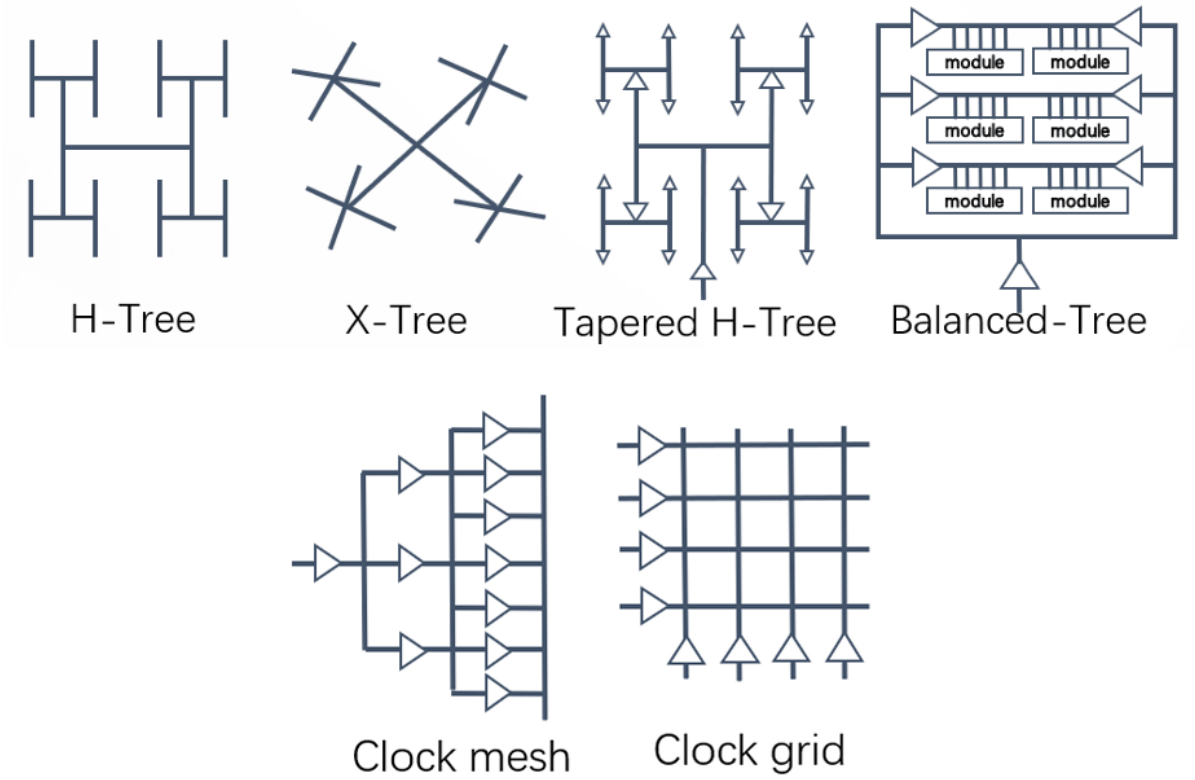


图 2. 不同时钟树拓扑结构

传统的基于树的时钟分配结构虽然在商业时钟树合成工具中仍占据主导地位，但为了减少偏差和提升系统鲁棒性，研究人员已经开始探索包括网格和其他非树状拓扑（例如加入交叉链接的树结构）在内的其他方法。这些方法虽然增强了系统的鲁棒性并减少了偏差，却以增加功耗、占地面积及布线长度为代价，同时也复杂化了信号分析。

为了解决这些问题，作者提出了一种新型的时钟树结构——广义 H 树（GH-tree）。如图 3所示，GH-tree 是一种具有不同分支因子的平衡树形拓扑，每个级别的分支因子都可以根据设计需求进行调整。例如，在一个八级 GH-tree 中，其分支模式为（4, 2, 2, 2, 4, 2, 2, 2），显示了从根到各分支的层次分布。这种结构不仅在理论上创新，通过动态规划算法有效生成了在特定延迟和偏差目标下的最优 GH-tree，极大地优化了时钟功率。GH-tree 结构旨在寻找偏差、延迟与功率之间的最佳平衡点，为高效的时钟网络设计提供了一种新的解决方案，具有重要的实际应用价值。

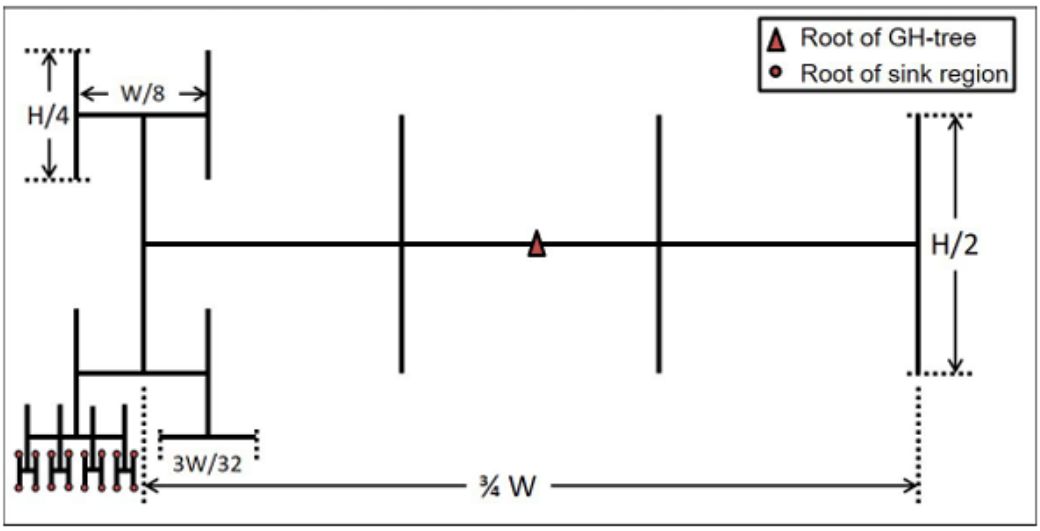


图 3. 分支模式为 (4,2,2,2,4,2,2,2) 的 8 级 GH-tree

2 相关工作

2.1 非树型方法

网格拓扑通常被理解为能提供鲁棒性和小的偏差。许多研究如 [9], [10], [21] 和 [22]，为需要鲁棒时钟网络的高性能电路提出了时钟网格设计。为了降低成本（例如，线长和功率），已提出将网格和树拓扑结合的混合时钟分配方法 [20], [24]。一些工作 [7], [14], [17], [23] 提出在初始时钟树方案的基础上插入交叉链接，以最小化时钟偏差，并且功率开销小。Rajaram 和 Pan [17] 提出了一种递归合并子树并进行逆向延迟传播的算法。Ewetz 和 Koh [7] 提出了系统的交叉链接插入方法，以增强时钟树的鲁棒性同时最小化其开销。他们提出了一个顶点减少方法来减少其非树结构中的冗余。虽然非树方法减少了时钟偏差并增强了时钟网络的鲁棒性，但其固有的冗余造成了额外的成本（例如，线长、功率和验证工作）与基于树的方法相比。此外，如网格这样的非树时钟分配拓扑缺乏灵活性和可调整性；这可能阻碍了例如有用的偏差优化等。

Dolev 等人 [6] 提出了一种基于六边形网格的时钟拓扑，包括一个控制网格中时钟信号并向附近功能单元供应时钟信号的六边形网格和中间节点。Abdelhadi 等人 [1] 提出了一种构建基于非均匀网格和无缓冲树的变异容忍混合时钟网络的算法。他们的方法有选择地减少了关键时序路径上的时钟偏差变化。Zhou 等人 [30] 提出了一种算法，用于确定在树驱动的网格时钟网络中驱动非均匀负载分布的时钟网格的本地缓冲的接点。他们的算法首先计算每个节点的负载，然后对节点进行聚类。接点是基于每个聚类的最小和最大延迟确定的。

最近，Kim 和 Kim [13] 提出了一个时钟脊网络的综合算法，有效地优化了时钟资源和变异容忍度之间的权衡。他们算法的关键思想是将时钟脊的分配和放置问题视为一个切片平面图优化问题。这些时钟树解决方案根据 CTS 算法 [13] 生成，作者用他们的 GH 树解决方案与之进行了比较。

2.2 基于树的方法

由于成本效率,基于树的方法常用于低功耗设计中的时钟分配。早期的工作 [3], [5], [12], [28] 提出了基于线性或 Elmore 延迟模型的时钟树构建，以最小化给定偏差目标的线长。然而，这些工作忽略了缓冲器的延迟和功率影响。方法 [4], [8], [15], [16], [18], [19], [27] 和 [29] 理解了缓冲影响，并共同优化了时钟树构建（即，树拓扑）和缓冲。Vittal 和 Marek-Sadowska [29] 提出了一个早期算法，共同优化树拓扑和缓冲器插入。Mehta 等人 [16] 提出了一个聚类算法，以获得近似负载平衡的聚类并构建时钟树，从而最小化偏差。这些以前的方法通常以自下而上的方式用贪婪算法构建时钟树，并没有探索偏差与成本的权衡。此外，很少有工作调整他们的树构建方法，并用商业的 P&R 工具和实际的设计块验证他们的解决方案质量。其他工作 [2], [11] 是基于由商业 P&R 工具生成的初始时钟树解决方案的 ECO-based 增量优化。这些工作的目标函数与我们的不同。Chan 等人 [2] 在顶层最小化了偏差，而 Han 等人 [11] 在角落间最小化了偏差变化。

3 本文方法

作者描述了 GH 树的构建问题，通过 GH 树的研究，作者能够更深入地探讨偏差、延迟和时钟功率之间的权衡关系。构建过程中，考虑了缓冲器插入、接收器放置以及多个设计约束（如最大转换时间和最大负载电容）对延迟和功率的影响。

具体来说，作者面对的挑战是如何构建 GH 树。这涉及根据给定的接收器放置方案（即布局区域 $W \times H$）、接收器区域的数量 N（每个区域包含不超过 40 个接收器），以及相关的时钟缓冲库（.lib）、最大时钟偏差、最大延迟、最大转换时间和最大负载电容的约束来构建 GH 树。

GH 树的构建过程主要包括两个步骤：

(1) 根据总汇聚区容量和布局区域面积及纵横比，首先制定一个动态规划（DP）问题，以共同优化分支模式和缓冲。

(2) 然后进行平衡的 K-means 聚类，并制定一个整数线性规划（ILP）问题，以确定时钟缓冲器的放置，即将我们的广义 GH 树结构嵌入给定的汇聚区布局中。

图4详细展示了 GH 树构建的整个过程。构建实例包括布局后的布局、接收器区域的数量及其约束，并辅以预先定义的技术和库特定的查找表（LUTs），这些表中包含了候选缓冲解决方案的功率和延迟信息。关键步骤是第一步（即，基于 DP 的时钟拓扑和缓冲共同优化），系统地探索了从 H 树到脊柱之间的连续体，以在此范围内实现时钟功率、偏差和延迟之间的最优权衡。最后，作者在商业 P&R 工具中实现我们的 GH 树解决方案，并报告度量（例如，偏差、延迟、功率等）来评估解决方案的质量。

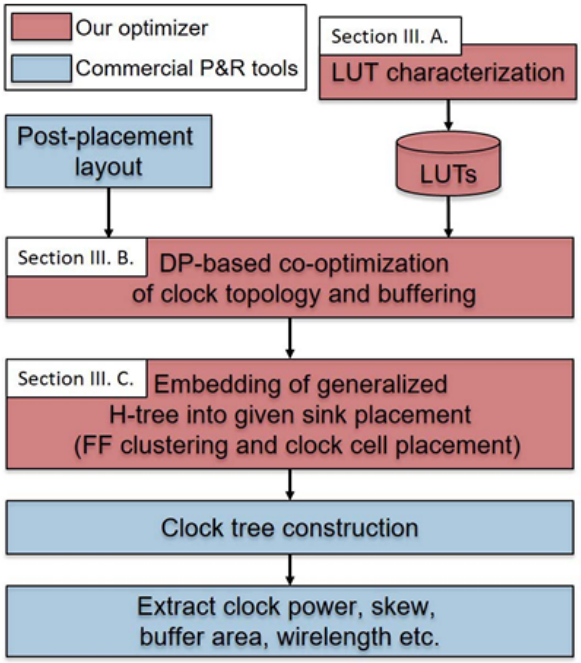


图 4. 构造 GH 树流程图

尽管作者的方法系统地优化了时钟功率、延迟和偏差之间的权衡，但存在两个局限性：首先，由于运算复杂度较高，没有将树的拓扑结构、缓冲和接收器放置一起优化；其次，方法中没有考虑在时钟树中使用时钟门控单元。因此，改进接收器区域的根部放置和顶层树拓扑优化之间的“先有鸡还是先有蛋”的问题，以及在优化过程中考虑时钟门控单元，仍然是值得进一步研究的方向。

3.1 查找表（LUT）表征

在这部分研究中，作者通过使用 Synopsys PrimeTime [25] 的仿真数据来定义查找表（LUTs），这些数据支撑了基于动态规划（DP）的优化方法。查找表中包含了具有缓冲和无缓冲导线段的相关信息，例如功率、输入电容、信号传播和延迟等。研究中使用了 28LP 库中的四种缓冲器：X50、X67、X100 和 X134。特别地，X134 缓冲器的门区域大约是最小尺寸 X2 缓冲器的七倍。此外，还运用了“并联缓冲器”配置，即通过将两个、四个或六个 X134 缓冲器的输入和输出端短接，以增强驱动力。

在制造导线段时，作者设定了 15 至 90 微米不等的多种长度，以此枚举所有可能的缓冲方案，每个方案的最小间隔设为 15 微米。这些 LUTs 的细节设置（如缓冲器的候选数量和导线段的最小长度）决定了优化过程中运行时间与解决方案质量之间的权衡。实际操作中，LUTs 的粒度是经验性选择的，目的是在保持运行时间可比的情况下，实现比两种商业工具更优的解决方案质量。例如，在 45 微米的段长下，最小的缓冲距离设为 15 微米，并且考虑七种不同的缓冲器尺寸，这样就形成了 83 种独特的缓冲方案。值得一提的是，这些 LUTs 还包括了无缓冲的方案，即纯导线方案。

在导线的具体表征中，研究考虑了单宽单距（1W1S）和双宽双距（2W2S——这是时钟分配中常见的非默认路由规则（NDR））两种配置。输入信号上升时间从 5 皮秒调整到 60 皮秒，以 5 皮秒为步长，输出负载从 1 飞法调整到 150 飞法，1 到 5 飞法之间步长为 1 飞法，5

到 150 飞法之间步长为 5 飞法。对于每一种（距离、输入上升时间、输出负载）组合，作者都能获取相应的缓冲方案，包括输入电容和输出上升时间。在许多可能的方案中，作者通过选择延迟范围内的 10%、50% 和 90% 三个点的最小功率解决方案来进行剪枝，以简化选择过程并提高效率。

在研究报告中，图5向我们展示了作者在查找表（LUTs）上执行剪枝操作的一个实例。在此实例中，作者选择了在不同输出负载条件下消耗最小功率的解决方案。图中红色和蓝色的点分别代表输出负载为 75 飞法和 35 飞法的缓冲解决方案。而被标记为十字（x）的点则显示了这些被选择的缓冲方案。

实际操作中，作者发现这种剪枝技术显著减少了缓冲方案的数量，具体减少了约 94%，而这种大幅度的减少只以大约 5% 的解决方案质量损失为代价。这里的质量损失主要体现在偏差或延迟方面，说明尽管剪枝极大地简化了解决方案的选择过程，但对性能的影响相对较小。这种权衡对于优化设计中的计算效率和性能至关重要，表明了在维持可接受性能损失的同时如何有效地减少计算资源的使用。

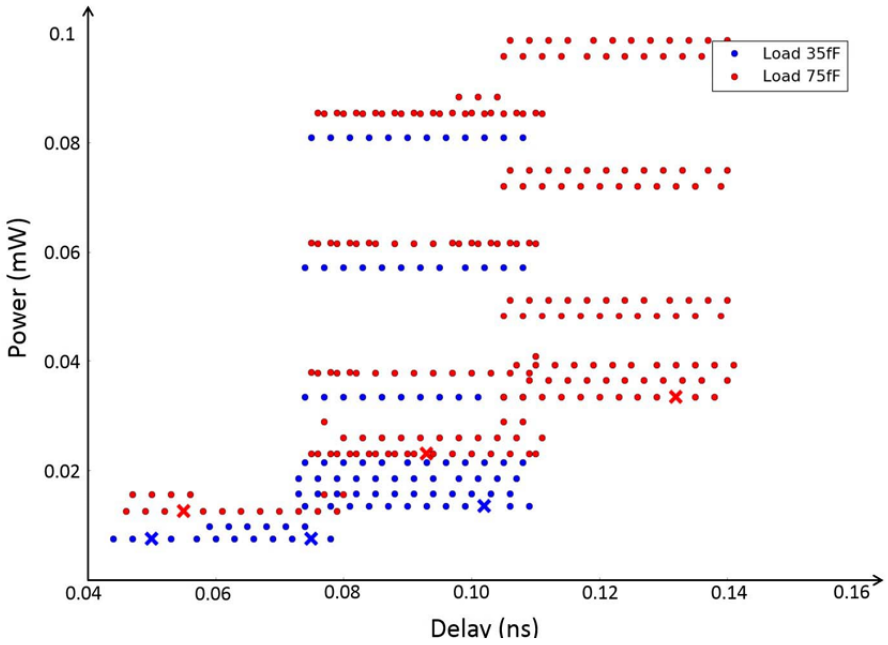


图 5. 缓冲解决方案剪枝的例子；距离 =45 微米，输出上升时间 =35 皮秒

3.2 基于 DP 的时钟拓扑和缓冲共同优化

基于已经特征化的查找表（LUTs），作者使用动态规划（DP）确定 GH 树的最佳分支模式以及相应的缓冲解决方案。作者优化的其他输入包括布局区域、接收器的放置、接收器区域的数量以及最大偏差和最大延迟的约束。在作者的优化中，一个接收器区域通常包含少于 40 个接收器。作者的目标是在满足给定的最大偏差和最大延迟约束的同时，最小化时钟功率。如所讨论的，作者假设接收器区域是由接收器的均匀放置诱导出来的，且分支点总是位于相应接收器区域的中心。作者明白，对于真实的放置解决方案，接收器区域通常不是均匀的。然而，由于高运行时间的复杂性，作者当前的方法实际上无法在 DP 优化过程中考虑接收器的放置。因此，作者在基于 DP 的 GH 树构建过程中假设接收器区域是均匀的。然后，作者将基于 DP 的解决方案（没有降低解决方案质量）嵌入到给定的（真实的）接收器放置中，这

基于接收器聚类和分支点的位移。作者在一个高维解决方案空间中构建 DP，该空间涉及七个维度（即，关于时钟树优化的七个关键参数），并以自下而上的方式构建 GH 树。这七个维度包括时钟树深度（P），区域面积（宽度（w）和高度（h）），接收器区域数量（n），最大和最小时钟延迟（t_{max} 和 t_{min}）以及输入电容（即从根部看到的负载电容）。

Algorithm 1 DP-Based GH-Tree Construction

$R \leftarrow build\_base\_trees(w, h, n, \emptyset)$, $\forall\, w, h, n$ $s.t.\ 0 < w \leq W,\ 0 < h \leq H,\ 2 \leq n \leq N,\ n$ is an even number
for $P := 2$ to P_{max} **do**
  for $w := 0$ to W **do**
    for $h := 0$ to H **do**
      for $n := 2$ to $(N \cdot w \cdot h)/(W \cdot H)$ **do**
        $R \leftarrow retrieve\_subtrees(P_l, w_l, h_l, n_l)$
        for all $(r_l) \in R$ **do**
          $r \leftarrow build\_tree(w, h, n, r_l)$
          $r' \leftarrow retrieve\_tree(R, P, w, h, n, r.t_{max}, r.t_{min})$
          if r' = **null then**
            $R \leftarrow R \cup \{r\}$
          else if $r.power < r'.power$ **then**
            remove r' from R
            $R \leftarrow R \cup \{r\}$
          end if
        end for
      end for
    end for
  end for
end for
$r_{opt}.power \leftarrow \infty$
for all $r' \in R$ s.t. $r'.w = W,\ r'.h = H,\ r'.n \geq N$ **do**
  if $r'.t_{max} - r'.t_{min} \leq \Omega$ && $r'.t_{max} \leq T$ && $r'.power < r_{opt}.power$ **then**
    $r_{opt} \leftarrow r'$
  end if
end for
return r_{opt}

算法 1 描述了作者的优化过程；另见图6中的插图。作者首先为基础情况构建 GH 树，即在所有不同区域面积（即，$w \times h$）和接收器区域数量（即，n）上的树深度等于 1（第 1 行）。例如，图6展示了不同区域面积下深度为 1 的子树（即，p 层）的解决方案，如 5×5（红色）、10×10（绿色）和 15×15（紫色）。构建基础树的过程 build_base_trees$(w, h, n, \emptyset)$ 在一个 $w \times h$ 区域内构建深度为一的 GH 树（即，具有不同缓冲解决方案的脊柱），并且具有分支因子 b_p。根据文献 [26]，作者使用术语 “脊柱” 来指代时钟树中的一个水平或垂直导线段。请注意，在最底层，$b_p = n$。如图6所示，作者基于特征化的 LUTs 沿脊柱优化缓冲解决方案。沿脊柱优化每个树段产生了一个在高维空间中的最小功率 Pareto 面，该空间由 LUT 输入参数（例如，最大和最小延迟，输入电容）索引。最终优化产生了多个子树解决方案。作者将这些子树解决方案存储在一个由树深度、区域面积、接收器区域数量、最大和最小时钟延迟以及输入电容索引的集合 R 中。换句话说，R 是一组子树解决方案，连同它们的深度、区域面积、接收器区域数量、时钟延迟、输入电容和与最小功率 Pareto 面对应的功率信息。

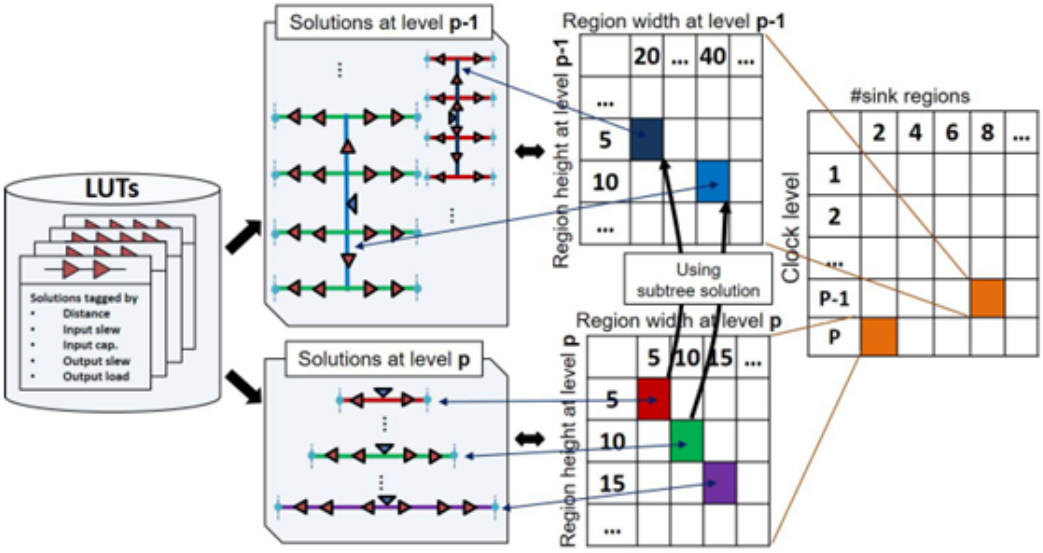


图 6. GH 树拓扑和缓冲的协同优化

接下来，作者递归地寻找具有深度 $P > 1$、区域面积 $w \times h$ 和接收器区域数量 n 的最优（即最小功率）GH 树解决方案。在优化过程中，作者每次迭代增加 P，直到 $P = P_{max}$（第 2-20 行）。对于给定的 N，最大深度 P_{max} 是基于传统的 H 树（其所有级别的分支因子为二）估算的，即 $P_{\max} = \lceil \log_2 N \rceil$。对于每个深度 P，作者根据存储在深度 $(P-1)$ 的子树解决方案，在最顶层的树段上执行缓冲优化。换句话说，作者使用现有的解决方案（即，来自 R 的子树解决方案）并在最顶层添加一个优化后的缓冲层，以构建一个深度增加一个的新树。图 5 展示了作者的优化器如何根据 p 层的解决方案（即，深度为 1 的子树）构建 $p-1$ 层的解决方案（即，深度为 2 的子树）。在此示例中，基于四个为区域面积 5×5（分别为 10×10）在 p 层实例化的解决方案，构建了区域面积 20×5（分别为 40×10）在 $p-1$ 层的解决方案。

对于每个 (P, w, h, n) 元组，作者基于所有已存储子树 (r_l) 的集合 R 构建优化解决方案，这些子树满足条件：$P_l = P - 1; w_l = h; h_l = w/b_t; n_l = n/b_t$。其中 P_l 是 r_l 的深度；$w \times h$ 是布局区域的尺寸；$w_l \times h_l$ 是接收器区域（即，r_l 的子树布局区域）的尺寸；b_t 是当前树最顶层的分支因子；n_l 是 r_l 的接收器区域数量。第 3、4 和第 5 行分别列举了可能的尺寸和接收器区域数量的子树解决方案。在算法 1 的第 6 行中，作者找到所有在先前迭代中优化过的子树解决方案，这些解决方案的分支因子 b_t 满足 $2 \le b_t \le n/2$。

然后，过程 build_tree(w, h, n, r_l) 使用收集到的子树 r_l R（其深度为 $(P-1)$）作为其下层子树来构建深度为 P 的树 r（第 8 行）。换句话说，作者在最顶层构建具有优化缓冲的树段，在最顶层的每个接收器处，作者使用（即，实例化）相同的子树 r_l 来构建树 r 的更低层次。为了降低运行时间的复杂性，作者的当前方法假设在任何层次上，子树都是相同的。如下文第四节所示，这并不妨碍最终解决方案的高质量。在所有构建的深度为 P 且具有相同最大和最小延迟的树中，作者选择最小功率的解决方案，并将其添加到解决方案集 R 中（第 7-16 行）。过程 retrieve_tree$(R, P, w, h, n, r.t_{max}, r.t_{min})$ 在第 9 行中检索满足条件深度 $= P$，宽度 $= w$，高度 $= h$，接收器区域数量 $= n$，且具有等于指定的 t_{max} 和 t_{min} 的最大和最小延迟的先前存储的解决方案 r。最后，作者选择满足最大偏差约束和最大延迟约束 T 的最小功率解决方案，从具有接收器数量 N 和区域面积 $W \times H$ 的集合 R 中选择（第 21-27 行）。

作者注意到，树内的偏差是从顶部向底部传播的。然而，作者的优化执行从底部向顶部

的 GH 树构建。作者从底部向上传播偏差，以准确捕捉偏差衰减并避免最大过渡违规。作者首先在每个接收器区域的根部假设几个偏差值（例如，25、30、35 和 40 皮秒）。对于每个假设的偏差值，作者根据 LUTs（注意作者的 LUTs 包含每个缓冲和/或接线解决方案的输出和输入偏差）从底部向上传播偏差。在基于 DP 的优化过程中，作者只选择确保在整个偏差传播过程中偏差值始终在 [5, 60 ps] 范围内的解决方案，其中 60 ps 是作者实验中的最大偏差约束，5 ps 是作者实验中实际可达到的最小偏差。作者还注意到，缓冲位置由选择的具有自下而上偏差传播的 LUT 解决方案确定。因此，缓冲并不一定在所有分支点上插入。

作者提出的算法的运行时间复杂度是 $O(P^{\max}.[(W \cdot H)/(w_{\text{int}} \cdot h_{\text{int}})] \cdot N^2 \cdot (\gamma_{\max}/\gamma_{\text{int}}))$，其中 w_{int}、h_{int} 和 γ_{int} 分别是离散化原始连续解决方案空间以形成 DP 的最小距离和时间间隔；$\gamma_{\max}$ 是指定的最大时钟延迟约束。作者在实验中将 w_{int}、h_{int} 设置为小于 5 微米，γ_{int} 设置为 1 皮秒。为了减少运行时间，作者采用以下剪枝技术：

(1) 根据叶区域数量进行剪枝：对于给定的 $w \times h$ 大小的子区域，作者剪除叶区域数量大于 $[(N \cdot w \cdot h)/(W \cdot H)]$ 的解决方案。

(2) 根据偏差/延迟约束进行剪枝：作者剪除偏差大于最大偏差约束或最大延迟大于延迟上限的解决方案。

(3) 根据最大扇出约束进行剪枝：作者剪除分支因子大于最大扇出约束的解决方案。

图7显示了基于 DP 的优化运行时间，其中接收器区域数量从 200 到 4000 不等，每个接收器区域包含约 25 个触发器。实验中使用的最大偏差约束为 30 皮秒。结果显示，通过剪枝，基于 DP 的优化可以在 6 小时内优化一个包含超过 4K 个接收器区域（或 100K 个触发器）的设计。假设触发器数量与总实例数量的比例通常为 10% 至 25%，作者的方法可以在 6 小时内优化一个包含 1M 个实例的设计。作者的研究表明，如果不采用所提出的剪枝技术，由于中间解决方案的大量增加（以至于由于过度的内存使用，无法优化超过 1K 个接收器区域的设计），运行时间和内存使用将显著增加。此外，作者观察到，使用剪枝和不使用剪枝的运行之间的解决方案质量相同。

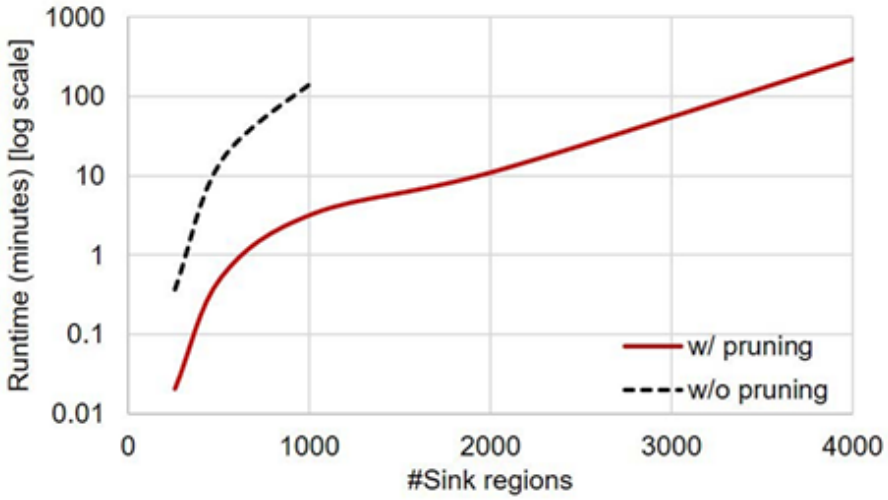


图 7. 基于 DP 的优化方法在使用和不使用剪枝技术的情况下，针对不同数量的接收器区域的运行时间

3.3 将 GH 树嵌入接收器放置中

作者基于 DP 的优化得到的时钟树拓扑和缓冲解决方案假设接收器（区域）分布是均匀的（其中分支点位于区域的中心）。然而，考虑到一个（现实的）非均匀的接收器（触发器）放置，作者必须对触发器进行聚类，以在不同的聚类之间实现负载平衡，从而避免偏差和延迟的增加。换句话说，应该根据实际的接收器触发器放置，将触发器的聚类分配给 GH 树的接收器。为了将优化的 GH 树适配给定的接收器（即，触发器）放置，作者执行一个平衡的 K-means 接收器聚类，并根据聚类解决方案调整缓冲器放置。作者注意到，他们的方法与传统的自上而下的时钟树构建方法（例如，Planar-DME [12]）不同，因为：1）作者将一个优化的时钟树拓扑和缓冲解决方案嵌入到一个给定接收器放置的布局区域中；2）作者在接收器区域之间平衡负载电容。通过维持连续分支点和每个时钟级别上的缓冲器之间的距离以及在接收器区域之间平衡负载电容，他们保持了通过基于 DP 的优化获得的 GH 树解决方案的质量（即，偏差和延迟）。此外，作者理解最佳的 GH 树拓扑和缓冲解决方案可能会因不同的接收器放置而变化。考虑到这一点，作者保留了基于 DP 的优化中最好的 M 个解决方案，并为给定的（实际的）接收器放置选择最小功率解决方案作为作者的最终解决方案。根据作者的初步实验，他们经验性地使用 $M = 5$ 来生成报告的结果，其中 M 增加到 5 以上将不会显著提高解决方案质量。

作者描述了两个数学规划（整数线性规划，ILPs），这些规划执行接收器聚类（即，将触发器分配给构建的 GH 树的接收器）并放置 GH 树的缓冲器（图8中展示了一个示例）。这两个 ILPs 分别在全局和局部聚类层面上进行操作者自上而下、逐层执行聚类和分支点放置，聚类优化在不同的聚类（每个聚类分配给 GH 树的一个接收器）之间平衡负载电容，以最小化 DP 解决方案（假设均匀的接收器放置）与最终解决方案（给定的实际接收器放置）之间的差异。这意味着必须在最底层考虑线路电容，即通过商业化的布局与布线（P&R）工具实现布线。然而，任何关于这种线路电容估计的建设性方法都是不准确的，并且可能在顶层优化期间显著增加运行时间的复杂性，因为每个分支点可能有许多接收器触发器。因此，作者根据下游接收器（即触发器）的总数将 GH 树划分为全局和局部聚类，并且只在局部聚类期间考虑线路电容。

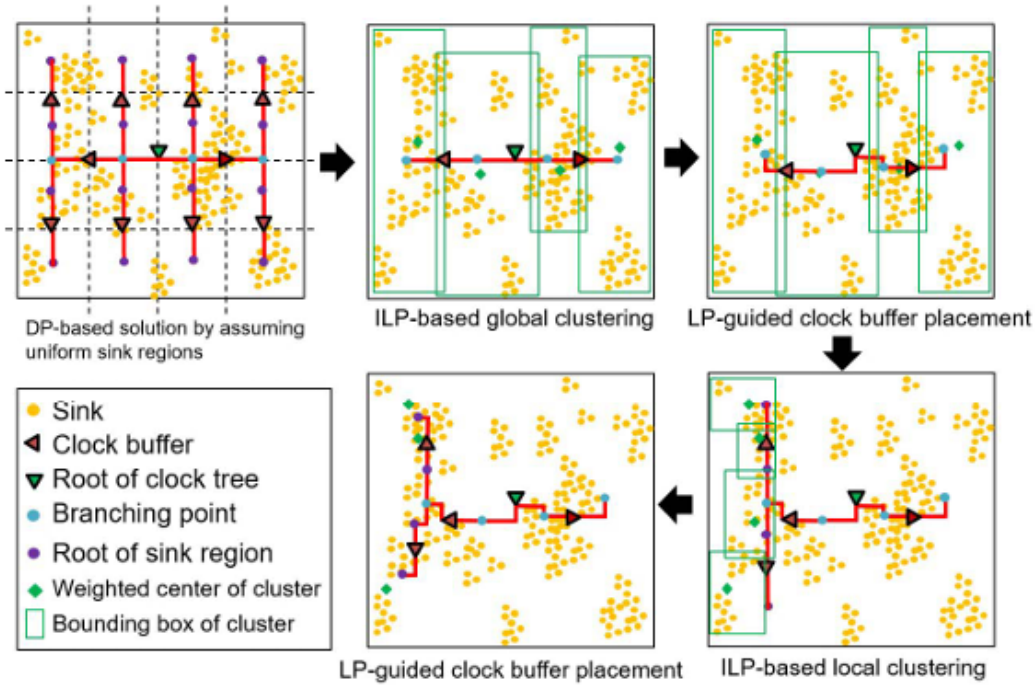


图 8. 接收器聚类和时钟缓冲器放置示例

算法 2 描述了作者的聚类程序。对于每个层级 p，作者迭代地应用全局或局部聚类，随后是基于线性规划的分支点和时钟缓冲器放置。从树的第一层（最顶层）开始，作者根据 DP 基础 GH 树解决方案的分支点的初始位置（即图8（左上角）中的浅蓝色点，位于均匀接收器区域的中心）对接收器进行聚类。全球聚类的数量与分支点的数量相同。例如在图8中，由于 $p = 1$ 且 $b_1 = 4$，作者的 ILP 将生成四个具有相似负载电容的全球聚类。然后，作者制定一个线性规划来确定每个全球聚类中的确切分支点位置以及缓冲器位置。当每个聚类中的接收器数量小于阈值（即，$|S|/|U| < Q_{\text{th}}$）时，作者应用基于 ILP 的局部聚类，并使用作者的线性规划在每个局部聚类中细化分支点的位置。

Algorithm 2 Embedding of GH-Tree Into a Sink Placement

```
for p := 1 to P do
    if |S|/|U| ≤ Q_th then
        global_clustering()
    else if r.power < r'.power then
        local_clustering()
    end if
    branching_point_and_buffer_placement()
end for
```

4 复现细节

4.1 与已有开源代码对比

本项目的小部分代码参考了国内外著名 EDA 开源社区的代码，其中 Clock.h 参考了 github 上 OpenROAD 时钟树综合点工具的数据结构的代码，MinCostFlow.h 参考了 gitee

上 iEDA 时钟树综合点工具的最小费用流（mcf）代码。本实验的其余大部分代码均为原创，在此明确申明。

为了构建一个拥有完整流程的时钟树综合点工具，本项目的实现分为了以下三个层次：文件 I/O 层、算法层、用户交互层，如9所示。这种实现方法，可以让使用者按照需求来调整他们所需要的时钟树拓扑。本项目实现的主要难点，是需要寻找合适的聚类算法和拓扑树结构构建时钟树，达到使用 buffer 尽可能少，最小化 skew 和 latency，同时满足 max fanout、max RC、单元不能重叠的约束。同时，在找到较优解的情况下，使用快速方法，提升运行速度。而且需要合理处理版图的不同部分，寻找后处理方法，进一步降低 skew 和 latency。针对这些难点，我提出了以下几个主要关键技术点：

(1) 使用 K-Means 聚类和 H 树拓扑，可以达到 skew 和 latency 平衡

(2) 进行版图划分，对 FF 分组，减小每次聚类的规模，可以有效提升运行速度

(3) 构建完时钟树后，对其进行后优化，进一步减少 skew 和 latency

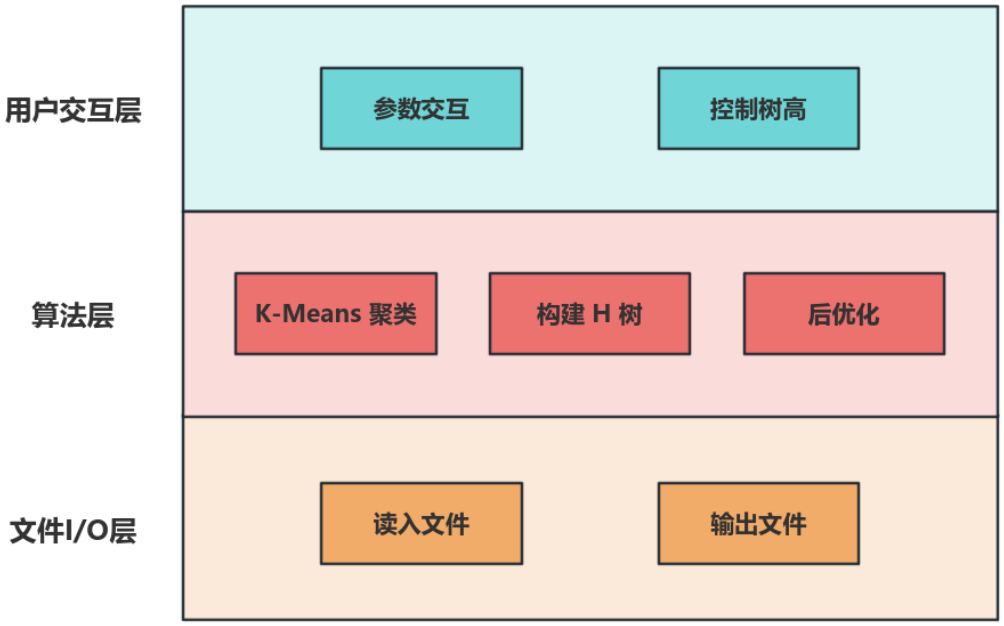


图 9. 项目层次图

4.2 主要算法流程

（1）读输入文件，保存数据

首先，我选择了适当的数据结构，对初始版图进行建模，这样能够存储后续构建出的时钟树。数据结构的类图如10所示。其中，class Instance 用于表示一个 instance，包含时钟的名称、类型和位置信息；class ClockSubNet 用于表示一个时钟子网，管理多个 instance，并提供子网相关操作；class Clock 用于表示一棵完整的时钟树，管理 instance 和子网，并提供计算和查询功能。

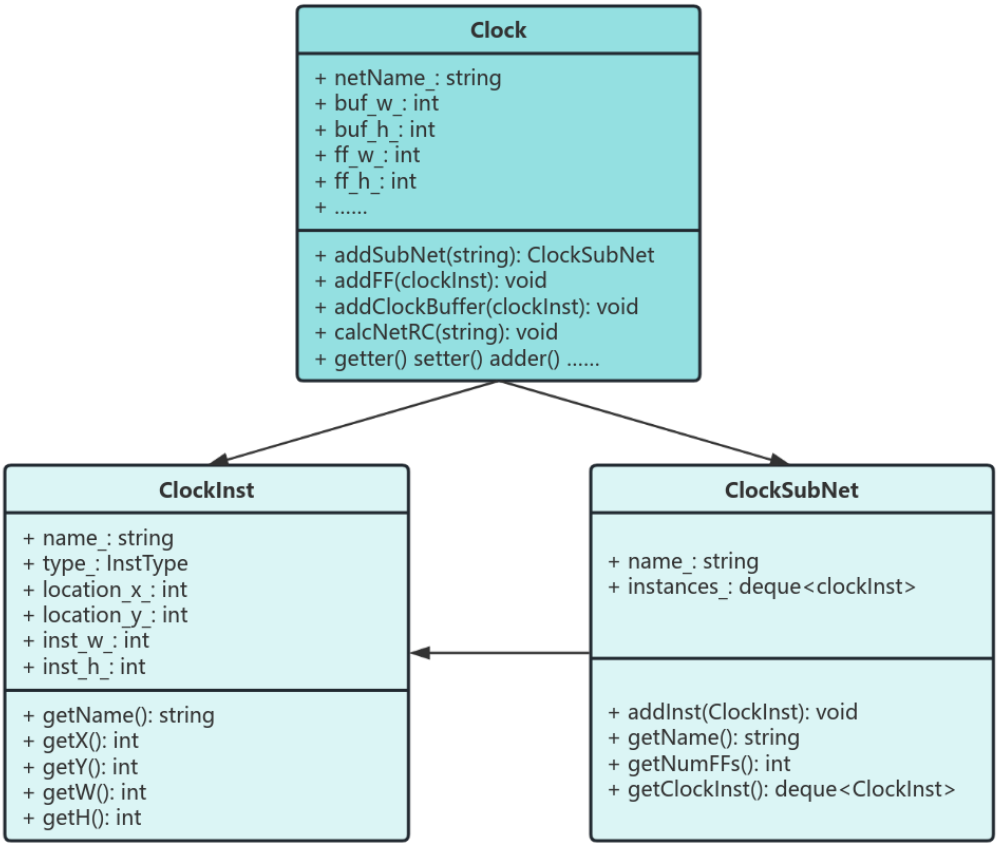


图 10. 数据结构的类图

（2）版图划分，多次聚类

对版图进行行和列的划分，以便对 FF 的坐标进行分组，反复进行划分，直到满足 max RC 的约束，再保存聚类结果，创建子网。版图划分可以有效控制每个簇的数据点与中心点之间的距离，减小 Net RC，同时避免离中心点较远的数据点聚类时被加到簇中，导致 Net RC 骤增。同时，对每组进行聚类时，簇的容量设为 max fanout，再动态调整行数和列数，能有效减少 buffer 数量

由于 buffer 之间距离较远，则通过降低 fanout（即减小每个簇的容量）来聚类，从而解决 net RC 的限制。以数据集中的 case2 为例，约 170k 个 FF 进行聚类，生成约 1900 个簇，这些簇的中心点即为 buffer 插入的位置。当前层的 buffer 作为下一次聚类的输入数据，再次聚类，直到 buffer 数量满足 GH 树的构建条件。以 case2 为例，经过版图划分，多次聚类的处理后，时钟树拓扑结构如图11所示。

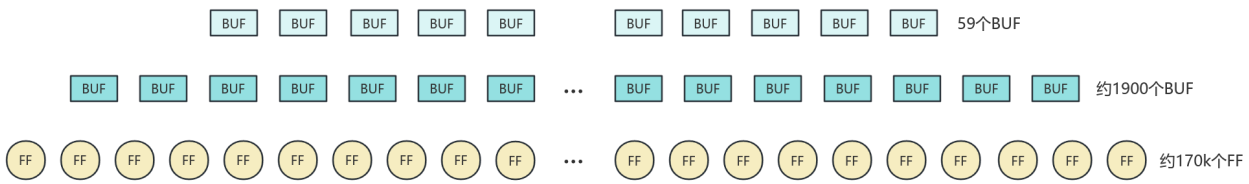


图 11. 聚类后的时钟树拓扑

（3）构建 GH 树

经过聚类后，以 case2 为例，剩余 59 个 buffer，记录下它们的坐标。然后计算出 bounding_box 的大小，这样便可以从 bounding_box 的中心开始，递归构建 GH 树。图12展示了这一过程。

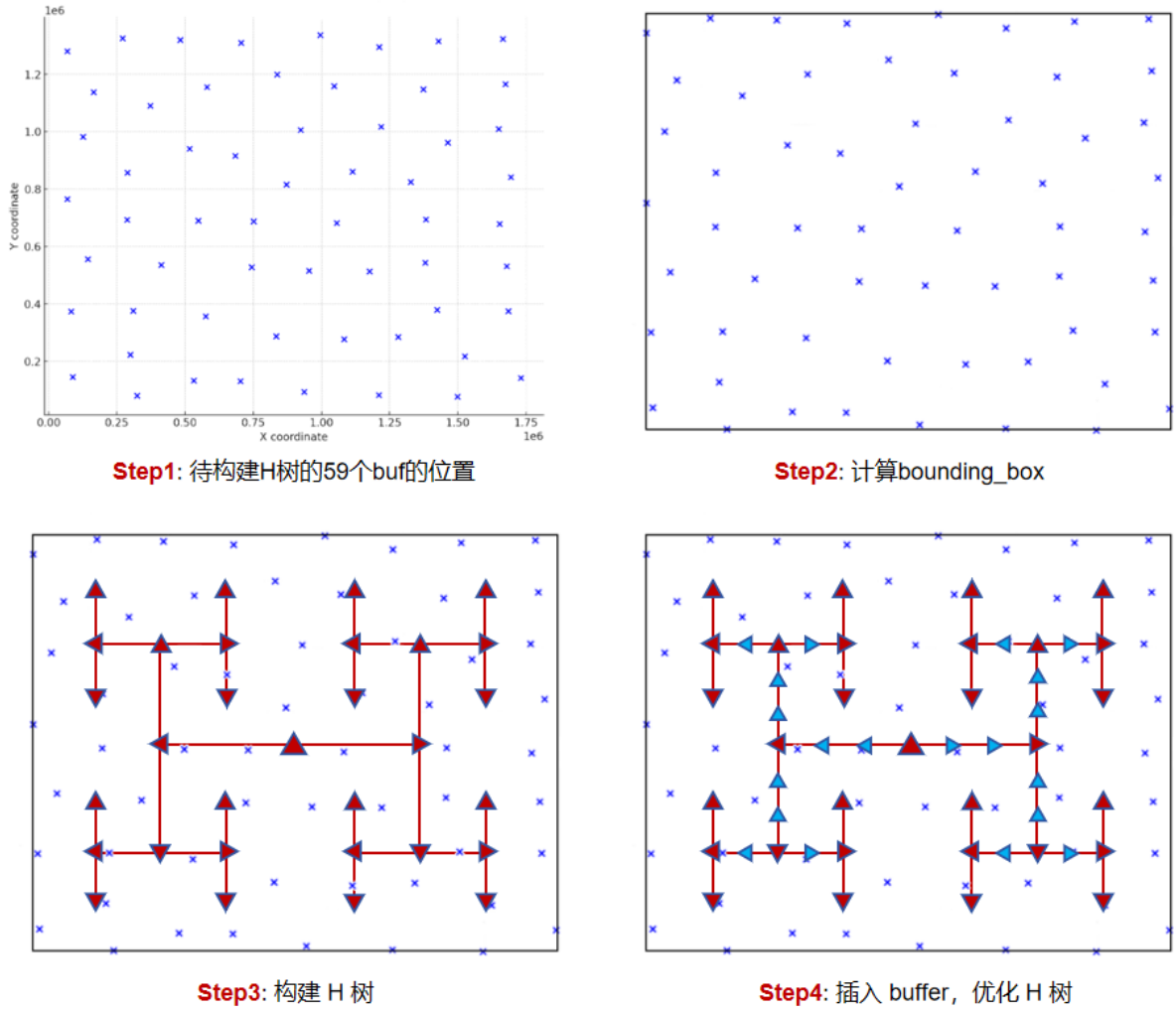


图 12. GH 树构建流程

构建完 GH 树后，需要连接 GH 树和下层聚类构建的时钟树，形成完整的时钟树拓扑结构。使用就近原则进行连接，这样能有效降低 skew。连接完成后，完整的时钟树拓扑结构如图13。

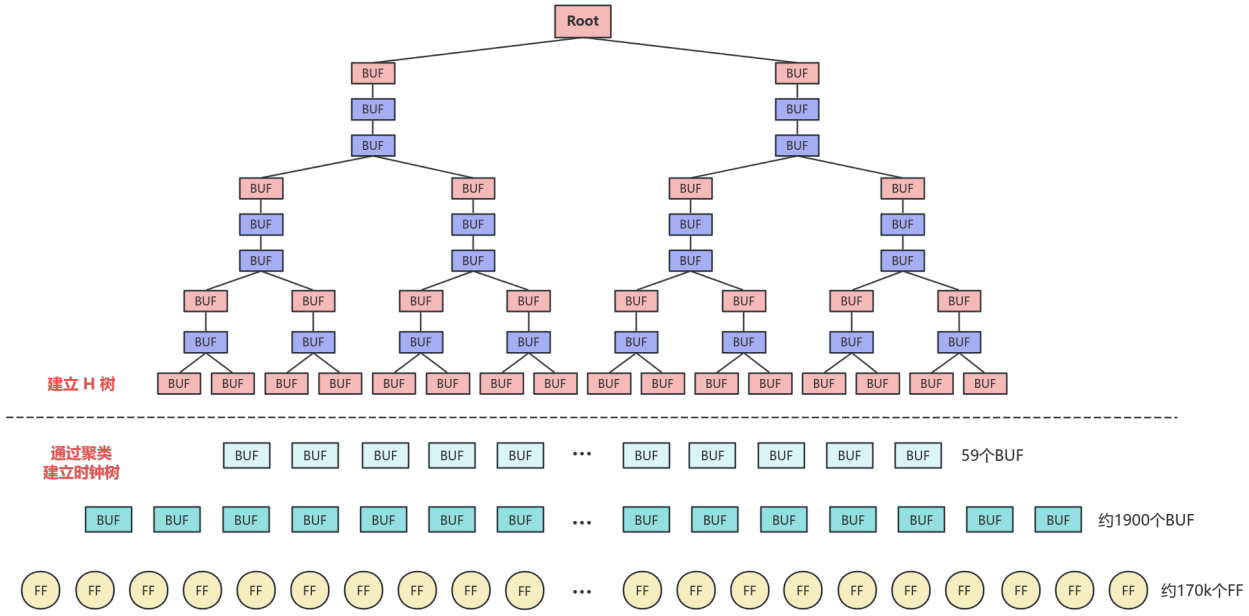


图 13. 完整的时钟树拓扑结构

（4）单元合法化

每次插入 buffer 时，如果检测到与 FF 或之前已插入的 buffer 发生重叠，则进行单元合法化。在寻找合法位置时，使用 “BFS+ 优先队列” 的贪心策略，确保每次找到的新位置与原位置最近。buffer 进行一次移动之后，如果检测到新位置又发生了重叠，则需要继续寻找下一个位置，直到不再发生重叠为止。单元合法化的示意图如图14所示，其中，红色矩形表示产生了重叠，需要进行单元合法化，绿色矩形是进行单元合法化后，buffer 找到的新位置。

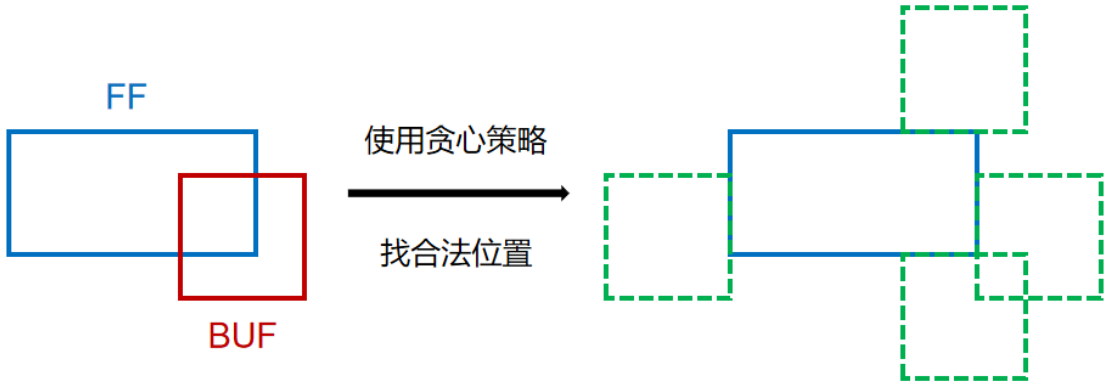


图 14. 单元合法化示意图

4.3 创新点

（1）K-Means 聚类 + GH 树框架

容量约束的 K-means 聚类算法（Capacitated K-means Clustering）是一种改良的 K-means 算法，它通过在聚类过程中添加容量限制来平衡各个簇的大小。这种算法首先选择具有最高权重的数据点作为初始中心，然后将每个数据点的权重除以其到簇中心的距离来计算优先级。数据点根据优先级顺序被分配到最近的簇，如果簇已达到其容量限制，则尝试分配到下一个

最近的未满簇。算法通过算术平均值更新簇中心，确保所有簇不超过指定的容量限制，从而在实现高效聚类的同时，保持簇的规模均衡。

我构建 GH 树的核心思想是先构建一个 H 型骨架结构，通过递归地在水平和垂直方向上交替分割空间来确定各级缓冲器的位置，同时在满足特定条件时进行分支优化以改善信号传输。在构建完 GH 树骨架后，算法采用最小费用流的方法将原始的时钟端点聚类并映射到 GH 树的叶子节点上，通过这种方式既保证了时钟树的结构对称性，又能够根据实际布局优化局部连接，最终通过计算和比较不同方案的平均延迟来选择最优解，实现了结构均衡性和性能优化的统一。

我提出了一种创新的混合式时钟树构建方法，巧妙地结合了 GH 树的结构优势和聚类的灵活性：在上层采用改进的 GH 树算法构建骨干网络，通过动态分支优化和延迟驱动的缓冲器插入策略，确保信号传输的对称性和平衡性；在下层则运用基于最小费用流的智能聚类方法处理实际的时钟端点分布，有效应对不规则布局带来的挑战。两层结构的连接通过一种创新的距离映射算法实现，该算法通过计算簇的几何中心并寻找最近的 GH 树叶节点来建立最优匹配关系，既保持了 GH 树的结构优势，又充分适应了实际布局的不规则性。这种层次化的混合方法不仅克服了单一方法的局限性，还在保证时钟信号质量的同时显著提高了布线效率，为大规模集成电路的时钟树综合提供了一个更加实用和高效的解决方案。

（2）版图划分

我的版图划分算法采用了自适应的网格划分和层次聚类策略：首先根据版图的长宽比例将整个设计区域划分成若干个网格，并将时钟端点（FF）按照其物理位置分配到对应的网格中。然后对每个网格内的端点采用 K-means 聚类方法，聚类数量基于最大扇出限制动态计算。算法会验证每个聚类结果的 RC 延迟是否满足约束，如果存在违例，则通过增加网格列数来细化划分粒度，直到所有子网都满足 RC 约束为止。这种方法既保证了局部连线的延迟控制，又实现了均衡的时钟树分布，同时通过迭代优化的方式自动调整划分粒度，达到了性能和布局质量的平衡。

5 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。本实验有两个输入文件，分别是：problem.def 和 constraints.txt，输入文件的路径是 exp/pre_submit，里面共有三个数据集。在 Linux 系统中，进入项目的根目录，使用指令 make 进行编译，编译完成后会在 bin 目录下生成一个可执行文件 icts，进入 bin 目录下运行这个可执行文件，即可运行本程序。如果需要更改数据集，请在 main.cc 中更改数据集的路径。运行成功后，会在相应数据集的目录下，生成 solution.def。

图15是关于输入文件的说明，图16展示了输入文件经过时钟树综合之后，输出结果的一个简单示例。

```
UNITS DISTANCE MICRONS 1000 ;  #文件中的数值单位为 1nm
DIEAREA ( 0 0 ) ( 0 y ) ( x y ) ( x 0 ) ;  #floorplan 区域为{0 0 x y}
FF ( a b ) ;                   #寄存器 cell FF size 为 a*b
BUF ( c d ) ;                  #buffer cell BUF size 为 c*d
CLK ( e f ) ;                  #clock root 点 CLK 坐标为（e f）
COMPONENTS N ;                 #instance 总数为 N，在输入文件中即 FF 总数
- FF1 FF ( x1 y1 ) ;           #instance FF1 cell 类型为 FF，坐标为（x1 y1）
- FF2 FF ( x2 y2 ) ;           #所有 instance 坐标均为 cell 左下角坐标
- FF3 FF ( x3 y3 ) ;           #所有 instance 方向默认一致，不考虑翻转旋转
.....
END COMPONENTS
```

图 15. 输入文件

输入文件：

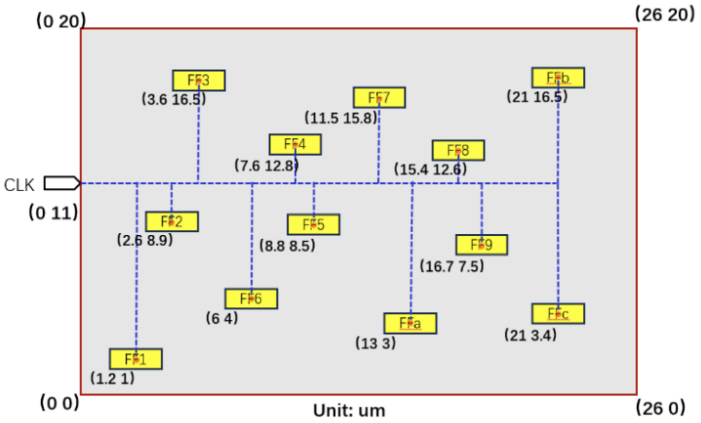
```
UNITS DISTANCE MICRONS 1000 ;
DIEAREA ( 0 0 ) ( 0 20000 ) ( 26000 20000 )
( 26000 0 ) ;
FF ( 2000 1000 ) ;
BUF ( 1000 1000 ) ;
CLK ( 0 11000 ) ;
COMPONENTS 12 ;
- FF1 FF ( 1200 1000 ) ;
- FF2 FF ( 2600 8900 ) ;
- FF3 FF ( 3600 16500 ) ;
- FF4 FF ( 7600 12800 ) ;
- FF5 FF ( 8800 8500 ) ;
- FF6 FF ( 6000 4000 ) ;
- FF7 FF ( 11500 15800 ) ;
- FF8 FF ( 15400 12600 ) ;
- FF9 FF ( 16700 7500 ) ;
- FFa FF ( 13000 3000 ) ;
- FFb FF ( 21000 16500 ) ;
- FFc FF ( 21000 3400 ) ;
END COMPONENTS
```

Unit rc: r 2 Ω/um， c 12pF/um

Max fanout: 4； max rc: 5000 ps

Buffer delay: 100 ps

输入文件图解：



输出文件：

```
UNITS DISTANCE MICRONS 1000 ;
DIEAREA ( 0 0 ) ( 0 20000 ) ( 26000 20000 )
( 26000 0 ) ;
FF ( 2000 1000 ) ;
BUF ( 1000 1000 ) ;
CLK ( 0 11000 ) ;
COMPONENTS 17 ;
- FF1 FF ( 1200 1000 ) ;
- FF2 FF ( 2600 8900 ) ;
- FF3 FF ( 3600 16500 ) ;
- FF4 FF ( 7600 12800 ) ;
- FF5 FF ( 8800 8500 ) ;
- FF6 FF ( 6000 4000 ) ;
- FF7 FF ( 11500 15800 ) ;
- FF8 FF ( 15400 12600 ) ;
- FF9 FF ( 16700 7500 ) ;
- FFa FF ( 13000 3000 ) ;
- FFb FF ( 21000 16500 ) ;
- FFc FF ( 21000 3400 ) ;
- BUF1 BUF ( 10000 10500 ) ;
- BUF2 BUF ( 3300 4700 ) ;
- BUF3 BUF ( 12600 8000 ) ;
- BUF4 BUF ( 20000 10500) ;
- BUF5 BUF ( 8000 16000) ;
END COMPONENTS
NETS 6 ;
- net_clk ( CLK ) ( BUF1 ) ;
- net_buf1 ( BUF1 ) ( BUF2 BUF3 BUF4 BUF5) ;
- net_buf2 ( BUF2 ) ( FF1 FF2 FF6) ;
- net_buf3 ( BUF3 ) ( FF5 FF9 FFa ) ;
- net_buf4 ( BUF4 ) ( FF8 FFb FFc) ;
- net_buf5 ( BUF5 ) ( FF3 FF4 FF7) ;
END NETS
```

输出文件图解：

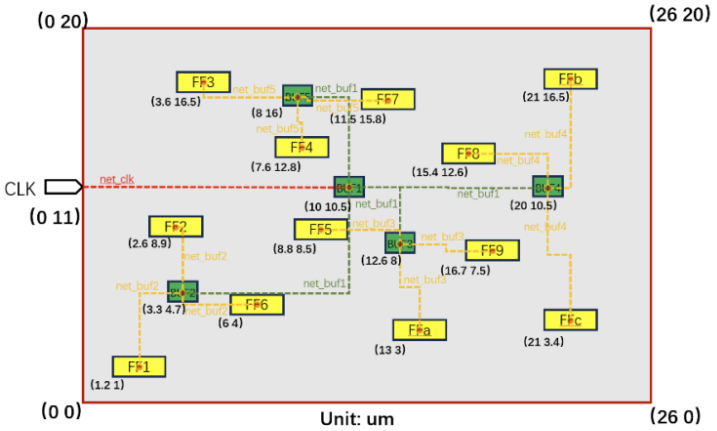


图 16. 示例说明

图17展示了三个数据集的实验结果，对其进行分析，可以看出，在使用较少 buffer 的情况下，对 skew 和 latency 都做到了有效降低。skew 应该越接近 0 越好，但三个数据集的 skew 都没达到 0，是因为时钟树拓扑结构的底层是用聚类算法生成的，这里产生了偏差。

除了 skew、latency 和 buffer 数量得到了有效控制，可以看出，fp，rc，fanout 和 overlap 等约束都满足要求，而且运行时间都比较短，说明我上面提出的框架是有效的。

case 1							
skew	latency	buffer	fp	rc	fanout	overlap	runtime
38.6741	585.1518	6443	0	26.53	0	0	97

case 2							
skew	latency	buffer	fp	rc	fanout	overlap	runtime
63.9372	618.9083	3123	0	0	0	0	133

case 3							
skew	latency	buffer	fp	rc	fanout	overlap	runtime
99.3926	459.1066	3096	0	0	0	0	16

图 17. 实验结果

6 总结与展望

本报告详细探讨了数字集成电路中时钟信号的重要性及其对系统性能的影响。文中讨论了多种时钟树结构，例如 H 树、X 树和平衡树，并分析了通过时钟树综合（CTS）和物理设计来优化时钟网络的方法。尽管传统的基于树的时钟分配方法在工业界得到了广泛应用，但在偏差控制和系统鲁棒性方面存在局限。因此，引入了包括广义 H 树（GH-tree）在内的一些创新方法，这些方法通过改进的拓扑结构和动态规划算法优化了时钟网络的设计，达到了偏差、延迟与功率之间的最佳平衡。

此外，本报告还具体说明了如何结合实际的布局区域、接收器放置和缓冲库来构建时钟树，并通过实验验证了所提方法的有效性。同时，探讨了利用最小费用流和局部聚类技术来优化非均匀布局下的时钟网络设计。报告总结了当前时钟网络设计领域的前沿技术和方法，尤其是在时钟树结构优化和偏差控制方面的进展。

然而，研究中指出，现有方法的复杂度和成本较高，未涉及时钟门控技术，这是减少功耗和提高效率的重要策略，并且目前的方法主要集中在理论和算法研究上，对于大规模商业应用的可行性和实际效果还需进一步验证。未来研究方向包括开发能够根据芯片操作条件自适应调整的时钟网络设计方法，探索将时钟门控技术与现有的时钟网络设计方法相结合，并考虑与机器学习、人工智能等领域的交叉研究，以提升时钟网络设计的自动化和智能化水平。这些研究不仅有助于推动时钟网络设计技术向更高效、更经济、更可靠的方向发展，还能更好地满足现代数字系统对时钟网络性能的严苛要求。

参考文献

[1] Ameer Abdelhadi, Ran Ginosar, Avinoam Kolodny, and Eby G Friedman. Timing–driven variation–aware synthesis of hybrid mesh/tree clock distribution networks. *Integration*, 46(4):382–391, 2013.

[2] Tuck-Boon Chan, Kwangsoo Han, Andrew B Kahng, Jae-Gon Lee, and Siddhartha Nath. Ocv-aware top-level clock tree optimization. In *Proceedings of the 24th edition of the great lakes symposium on VLSI*, pages 33–38, 2014.

[3] Ting-Hai Chao, Yu-Chin Hsu, Jan-Ming Ho, and AB Kahng. Zero skew clock routing with minimum wirelength. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(11):799–814, 1992.

[4] Ying-Yu Chen, Chen Dong, and Deming Chen. Clock tree synthesis under aggressive buffer insertion. In *Proceedings of the 47th Design Automation Conference*, pages 86–89, 2010.

[5] Jason Cong, Andrew B Kahng, Cheng-Kok Koh, and C-W Albert Tsao. Bounded-skew clock and steiner routing. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(3):341–388, 1998.

[6] Danny Dolev, Matthias Függer, Christoph Lenzen, Martin Perner, and Ulrich Schmid. Hex: Scaling honeycombs is easier than scaling clock trees. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 164–175, 2013.

[7] Rickard Ewetz and Cheng-Kok Koh. Cost-effective robustness in clock networks using near-tree structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(4):515–528, 2015.

[8] Matthew R Guthaus, Dennis Sylvester, and Richard B Brown. Clock buffer and wire sizing using sequential programming. In *Proceedings of the 43rd annual Design Automation Conference*, pages 1041–1046, 2006.

[9] Matthew R Guthaus, Gustavo Wilke, and Ricardo Reis. Non-uniform clock mesh optimization with linear programming buffer insertion. In *Proceedings of the 47th Design Automation Conference*, pages 74–79, 2010.

[10] Matthew R Guthaus, Gustavo Wilke, and Ricardo Reis. Revisiting automated physical synthesis of high-performance clock networks. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(2):1–27, 2013.

[11] Kwangsoo Han, Jiajia Li, Andrew B Kahng, Siddhartha Nath, and Jongpil Lee. A global-local optimization framework for simultaneous multi-mode multi-corner clock skew variation reduction. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[12] Andrew B Kahng and C-W Albert Tsao. *Planar-DME: Improved planar zero-skew clock routing with minimum pathlength delay.* UCLA Computer Science Department, 1994.

[13] Youngchan Kim and Taewhan Kim. Algorithm for synthesis and exploration of clock spines. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 263–268. IEEE, 2017.

[14] Bao Liu, Andrew B Kahng, Xu Xu, Jiang Hu, and Ganesh Venkataraman. A global minimum clock distribution network augmentation algorithm for guaranteed clock skew yield. In *2007 Asia and South Pacific Design Automation Conference*, pages 24–31. IEEE, 2007.

[15] I-Min Liu, Tan-Li Chou, Adnan Aziz, and DF Wong. Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion. In *Proceedings of the 2000 international symposium on Physical design*, pages 33–38, 2000.

[16] Ashih D Mehta, Yao-Ping Chen, Noel Menezes, DF Wong, and LT Pilegg. Clustering and load balancing for buffered clock tree synthesis. In *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, pages 217–223. IEEE, 1997.

[17] Anand Rajaram and David Z Pan. Variation tolerant buffered clock network synthesis with cross links. In *Proceedings of the 2006 international symposium on Physical design*, pages 157–164, 2006.

[18] Logan Rakai, Amin Farshidi, Laleh Behjat, and David Westwick. Buffer sizing for clock networks using robust geometric programming considering variations in buffer sizes. In *Proceedings of the 2013 ACM International symposium on Physical Design*, pages 154–161, 2013.

[19] Rajeev R Rao, David Blaauw, Dennis Sylvester, Charles J Alpert, and Sani Nassif. An efficient surface-based low-power buffer insertion algorithm. In *Proceedings of the 2005 international symposium on Physical design*, pages 86–93, 2005.

[20] Phillip J Restle, Timothy G McNamara, David A Webber, Peter J Camporese, Kwok F Eng, Keith A Jenkins, David H Allen, Michael J Rohn, Michael P Quaranta, David W Boerstler, et al. A clock distribution network for microprocessors. *IEEE Journal of Solid-State Circuits*, 36(5):792–799, 2001.

[21] John Reuben, Harish M Kittur, and Mohd Shoaib. A novel clock generation algorithm for system-on-chip based on least common multiple. *Computers & Electrical Engineering*, 40(7):2113–2125, 2014.

[22] John Reuben, Salma Nashit, Harish M Kittur, et al. Capacitance driven clock mesh synthesis to minimize skew and power dissipation. *IEICE Electronics Express*, 10(24):20130850–20130850, 2013.

[23] Rupak Samanta, Jiang Hu, and Peng Li. Discrete buffer and wire sizing for link-based non-tree clock networks. In *Proceedings of the 2008 international symposium on Physical design*, pages 175–181, 2008.

[24] Haihua Su and Sachin S Sapatnekar. Hybrid structured clock network construction. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*, pages 333–336. IEEE, 2001.

[25] Synopsys. *PrimeTime User Guide*, Accessed: 2025.

[26] Simon Tam. Modern clock distribution systems. In *Clocking in Modern VLSI Systems*, pages 9–65. Springer, 2009.

[27] Jeng-Liang Tsai, Tsung-Hao Chen, and CC-P Chen. Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(4):565–572, 2004.

[28] Chung-Wen Albert Tsao and Cheng-Kok Koh. Ust/dme: a clock tree router for general skew constraints. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 7(3):359–379, 2002.

[29] Ashok Vittal and Malgorzata Marek-Sadowska. Low-power buffered clock tree design. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 16(9):965–975, 1997.

[30] Nancy Y Zhou, Phillip Restle, Joseph Palumbo, Joseph Kozhaya, Haifeng Qian, Zhuo Li, Charles J Alpert, and Cliff Sze. Pacman: Driving nonuniform clock grid loads for low-skew robust clock network. In *Proceedings of SLIP (System Level Interconnect Prediction) on System Level Interconnect Prediction Workshop*, pages 1–5, 2014.