一、 作品简介

1.作品简介

我们团队实现的STA算法基于一种拓扑排序的变种，采用这种方法能够减少遍历无法产生违例的路径的次数，降低程序的时间复杂度，同时这种算法有着很高的可并行性，方便以后能将此算法改进为多线程或者是分布式的解决方案。同时，我们在编写程序时采用了模块化的编程方式来适应实际生产中的需求，能够方便的添加各种功能。

二、 实现原理

1. 概念简介和名词解释

为了方便后续描述清晰，先对一些可能出现的名词和一些可能产生歧义的词语做一些定义。

① **结点：与赛题的结点描述相同，为电路中的器件在图的数据结构中的表示。**

② **边：电路中的连线在图的数据结构中的表示**

③ **度：所谓顶点的度(degree)，就是指和该顶点相关联的边数。**

④ **入度（in-degree）：以某顶点为弧头，终止于该顶点的弧的数目称为该顶点的入度。**

⑤ **出度（out-degree）：以某顶点为弧尾，起始于该顶点的弧的数目称为该顶点的出度**

⑥ **拓扑排序： 对一个有向无环图(Directed Acyclic Graph 简称 DAG)G 进行拓扑排序，是将 G 中所有顶点排成一个线性序列，使得图中任意一对顶点 u 和 v，若边<u,v>∈E(G)，则 u 在线性序列中出现在 v 之前。通常，这样的线性序列称为满足拓扑次序(Topological Order)的序列，简称拓扑序列。简单的说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为拓扑排序。**

⑦ **子节点：从该结点出发到达的下一个结点。**

⑧ **父节点：从该结点出发到达的上一个结点。**

2. 基本流程

我们先对算法的基本流程进行介绍，先描述一个整体框架，然后再对算法的每一步骤的原理进行深入分析。

① **从文件中读入网表。**

从给定的文件中读入结点，给每个结点一个编号（ID），再从给定的文件中读

入边，用邻接表的方式储存图，同时计算出边上的延迟（包括 TDM 和 Cable）。

② 从每个 Flip-Flop 结点出发，维护每个 Flip-Flop 结点的父节点集合。

给每一个结点维护一个含有 Flip-Flop 属性的父节点的编号的集合。

我们在这一步便使用了一种拓扑排序的方法，我们从每一个 Flip-Flop 结点开始向它的子节点的集合中添加自己的编号，添加完成后同时将子节点的入度减一，当一个结点的入度减为 0 时，如果他是 non-Flip-Flop 结点则继续向他的子节点的集合中添加自己所拥有集合中的元素，如果是 Flip-Flop 结点则结束。

经过这一步骤后每个 Flip-Flop 结点的集合中就保存了驱动它的 Flip-Flop 结点的编号。

③ 寻找时钟

通过上一步的操作，我们获得了每一个 Flip-Flop 结点上一个 Flip-Flop 结点的集合，通过对这些结点的集合运算可以推断出哪些 Flip-Flop 结点是时钟器件，或者在第②步的过程中就可以推断完成，以减少寻找可能的时钟器件的事件。

④ 处理时钟延迟

沿着时钟结点向下修改结点的时钟延迟，计算出每个结点的时钟延迟。

⑤ 维护每个结点的最大（最小）到达时间

在这一步我们需要维护信号到达每一个结点的 arrive time。

这一步所用方法与第②步所用方法类似，不过在这一步我们需要向子节点集合里添加的是从当前结点的最大（最小）到达时间所对应的路径出发到达子结点的路径，这样我们就能保证在该结点入度减为 0 时，所有到达该节点的路径中的最大（最小）到达时间所对应的路径一定包含在这个结点所维护的路径中。

⑥ 对可能出现的违例路径进行处理

在第⑤步操作结束后，就能算出到达当前结点的最大（最小）到达时间，如果最大（最小）到达时间不会产生违例，则次大（次小）到达时间也不会产生违例，则以此类推到达该结点的所有路径都不会产生违例，所以对于该结点的违例查找结束。如果最大（最小）到达时间产生了违例则次大（次小）到达时间也可能产生违例，需要进一步处理。

⑦ 拓展每个结点的次大（次小）到达时间

如果当前结点的最大（最小）路径产生了违例，则需要查找次大（次小）路径，次大（次小）路径来源可能有两种情况：

1. 到达该结点最大（最小）路径的上一个结点（即该结点最大（最小）路径的对应父节点）的次大路径。
2. 由其他结点到达该结点的最大（最小路径）。

其中第 2 种情况已经存在于该结点维护的路径集合中，第 1 种情况是与该问题结构相同的递归子问题，所以我们只需要处理第 2 种情况。

⑧ 处理次大路径的问题

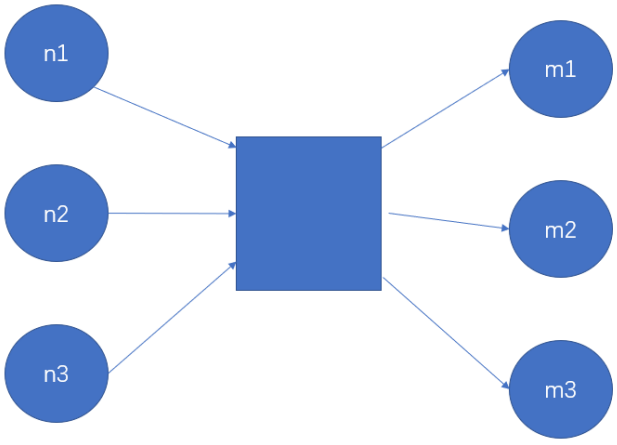
从第⑦步中已得知次大路径的来源，该问题是一个递归的问题，我们只需要从递归的终点（也就是最大（最小）路径的起点）出发，对该条路径上的每一个结点删除集合中对应的最大（最小）到达时间的路径，将次大（次小）的路径添加到子结点的集合中去。在这步完成后可以保证次大（次小）路径被保存在产生违例的结点的集合中。

如果次大（次小）路径不再产生违例则对于该节点的查询结束，否则回到第⑤步。

⑨ 打印违例，清理空间

3. 原理分析

①维护最大（最小）到达时间的优点



对于图中的情况如果采用穷尽搜索的方法，我们需要对左右两边每一对结点进行时序分析不论有没有违例都需要计算 n*m 次，而采用维护最大（最小）路径的的方法最好情况只需要计算 m 次这对于违例较少的项目来说提升是十分巨大的，尤其是在 n 和 m 的数目很大的时候。

如果将程序设置成不论到达时间为多少都视作是违例的话，该算法将退化为一种搜素算法，所以该算法的时间复杂度下限是搜素算法。

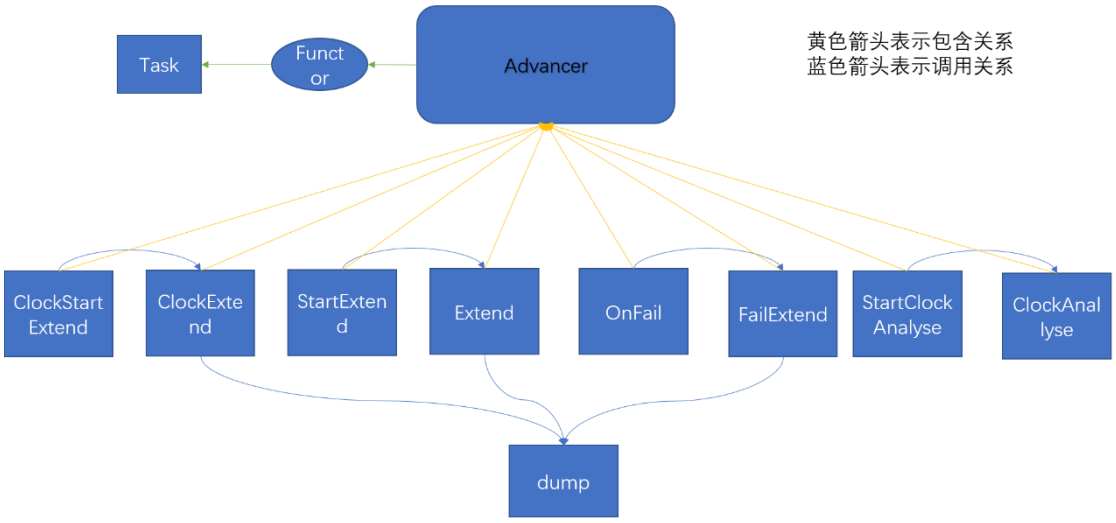
②可并行性

拓扑排序算法具有天生的可并行性，甚至许多线程调度算法内部都是由拓扑排序实现的，所以该算法有很大的潜力运用于分布式处理或者多线程处理机上，这对于越来越大的问题规模是十分友好的。

4. 流程实现

① 数据结构说明

1. Graph 保存了题目中的图结构，包括结点信息和网表连线信息，以及 TDM 和 clock 频率等信息。
2. Task 为了之后并行处理预留的接口，后续可能通过它来实现算法的并行计算。（目前只能单线程运行）
3. Advancer 算法运行的主要场所，代码中用它来开辟运行所需要的资源，内部采用模块化的功能设计，方便后续修改和添加功能。
4. PathRecord 保存路径的数据结构。
5. SafeMultiSet 每个结点用来维护的集合。



6. ClockStartExtend 用来从起始时钟器件出发确定每个器件的时钟路径的模块。
7. ClockExtend 用来从时钟器件出发确定每个器件的时钟路径的模块。
8. StartExtend 用来从起始器件出发确定每个器件的最大（最小）到达时间的模块。

9. Extend 用来从器件出发确定每个器件的最大（最小）到达时间的模块。

10. OnFail 决定产生违例后对于该结点的处理方法。

11. FailExtend 对产生违例的路径上的结点的集合进行操作。

② 执行流程

根据 2 中的基本流程对对应的结点调用相应 Start 模块到 Task 中，模块运行过程中会调用其他的模块，由 Task 对模块进行调度直到完成所有流程。

5.时空复杂度证明

我们定义结点的个数为 V，边的个数为 E，违例数为 N，违例路径的平均长度为 l，违例路径上的结点的平均出度是 o。

① 在这个算法中寻找路径需要对每一个结点开始运行，时间复杂度为 O(V)。

② 对于每一个结点要对它的每一个子节点处理，也就是对每一条边进行处理，时间复杂度为 O(E)。

③ 当每个结点入度减为 0 时，需要对该节点进行判断是否出现违例，所有结点处理一遍的时间复杂度为 O(V)。

④ 对每一条违例路径的处理需要对违例路径的每一个结点的出边进行操作，时间复杂度为 O(N*l*o)。

总时间复杂度为：

$$\boldsymbol{O(V)+O(E)+O(V)+O(l*o*N)=O(N+E)}$$

每个结点保存的路径数目不会超过 E，则

空间复杂度为：

$$\boldsymbol{O(E)}$$