【赛题解析】统计静态时序分析算法

一、所需知识背景

1. 专业背景不限。本赛题核心在读入数据建立时序图的基础上，考虑时序图中各节点的时序波动进行静态时序分析，适合微电子、计算机、数学等各专业，具有概率统计分布基础知识和编程能力的本科生、硕士生、博士生同学参加。

2. 参赛者必须使用 C++语言编程实现本赛题，不得使用除 STL 外的其他库。

二、赛题背景

静态时序分析(Static Timing Analysis, STA)是集成电路设计中的关键步骤，贯穿逻辑综合和物理实现中的各个阶段，用于检查是否存在建立时间违例和保持时间违例，对保障芯片性能指标和避免功能错误具有关键作用。随着先进工艺的演进，工艺参数偏差对电路时序的影响逐渐不可忽视，导致电路中器件和互连线的延时不再是确定值，而是存在统计波动。为了描述该波动，需要在传统的确定型静态时序分析(Deterministic STA, DSTA)的基础上根据设计经验人为引入一个悲观量，即通过统计静态时序分析(Statistical STA, SSTA)检查时序违例。和传统 DSTA 相比，SSTA 不但仍然需要考虑电路规模增长时的算法复杂度，而且还需要保证统计分析时的计算精度。

三、赛题描述

本赛题提供以下输入文件，均为 csv 文件格式。

1) 时序图文件：罗列时序图中所有节点间的连接关系和连接节点的边上的统计时序信息，包括延时均值(mean)和标准差(sigma)；
2) 建立时间检查文件：罗列时序图中所有寄存器的建立时间；
3) 时序路径起点(startpoint)文件：罗列所有时序路径起点，一般为寄存器时钟端(CK)；
4) 时序路径终点(endpoint)文件：罗列所有时序路径终点，一般为寄存器数据输入端(D)。

参赛者通过上述文件建立电路时序图并进行 SSTA，计算输出以下结果：

1) 考虑延时统计分布，计算时序路径终点(endpoint)文件中所有 endpoint 的建立时间裕量(slack)。在此过程中，考虑延时统计分布的峰度(skewness)，从而计算获得更准确的建立时间裕量(slack)是本赛题的一个加分项；
2) 计算时序图中的所有节点的全局裕量(global slack)。这是本赛题的另一个加分项。

四、赛题解析

1. 时序路径的建立时间裕量(slack)定义

STA 包括建立时间检查和保持时间检查两方面，本赛题仅作建立时间检查。对于图 1 所示的发送寄存器(launch flop, UFF0)和接收寄存器(capture flop, UFF1)之间的时序路径，定义建立时间裕量(slack)为式所示的所需时间(T_{rt}, required time)和到达时间(T_{at}, arrival time)之差。当 slack 为正值时，判定为满足时序约束，否则判定为时序违例。T_{rt} 可由式计算获得，其中 $T_{capture}$ 表示时钟信号(CLKM)到达 UFF1 时钟端(CK)所需的延时，T_{cycle} 表示时钟周期（本赛题中该值定为 10），T_{setup} 表示接收寄存器的建立时间；T_{at} 可由式计算获得，其中 T_{launch} 表示时钟信号(CLM)到达 UFF0 时钟端(CK)所需的延时，T_{ck2q} 表示 UFF0 的延时，T_{dp} 表示 UFF0 和 UFF1 之间的数据通路的延时。

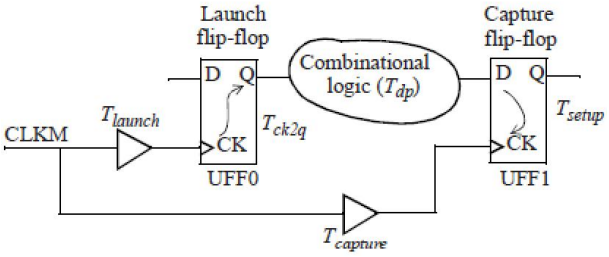

图 1 时序路径示例

$$slack = T_{rt} - T_{at} \tag{1}$$

$$T_{rt} = T_{capture} + T_{cycle} - T_{setup} \tag{2}$$

$$T_{at} = T_{launch} + T_{ck\,2q} + T_{dp} \tag{3}$$

在本赛题中，将延时当作统计量进行 SSTA，即根据各延时统计部分的均值(mean)和标准差(sigma)，分别计算 T_{rt} 和 T_{at} 的均值(mean)和标准差(sigma)，最终获得最坏情况下的 slack。本赛题中，需考虑 T_{launch}、$T_{capture}$、T_{ck2q} 和 T_{dp} 的延时波动，从而获得 slack 的波动结果。对于延时波动或 slack 波动，其最坏/最好情况可分别表示为 mean+N×sigma 和 mean-N×sigma，其中 N 为正整数，本赛题中 N 取 3。。

2. 时序路径的建立时间裕量(slack)计算

当以特定接收寄存器的数据输入端(D)作为时序路径终点(endpoint)时，本赛题要求计算此 endpoint 的 slack。实际以此为 endpoint 的时序路径不止一条，此 endpoint 的 slack 指的是所有以此为 endpoint 的时序路径的 slack 的最坏值，即值最小（负值同样取最小值）的 slack。可以采用邻接矩阵存储时序图，并通过深度搜索算法(DFS)或广度搜索算法(BFS)遍历时序图，找出 endpoint 对应所有时序

路径的 startpoint。

在如图 2 所示的时序图中，以寄存器 DFF3 的 D 端为 endpoint 的时序路径的 startpoint 包括寄存器 DFF1 和 DFF2 的 CK 端，可分别计算获得 T_{at}（DFF3 的 D 端）和 T_{rt}（DFF3 的 CK 端）的输出上升最大值(MaxRiseMean)、输出下降最大值(MaxFallMean)、输出上升最小值(MinRiseMean)、和输出下降最小值(MinFallMean)。

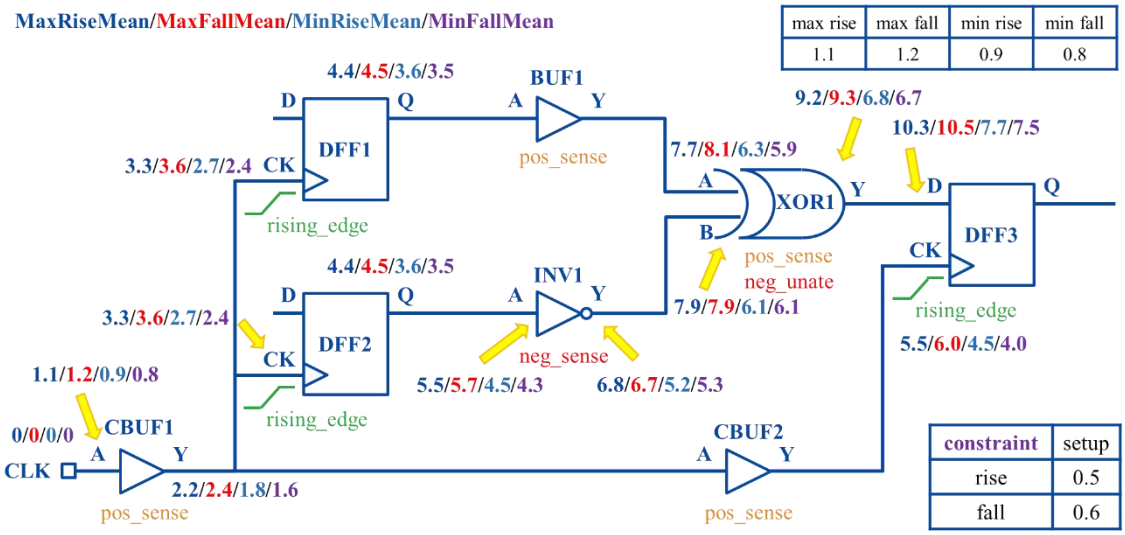

图 2 时序图示例

当计算获得了 T_{at} 和 T_{rt} 之后，可由式分别计算获得输出上升建立时间裕量(SlackRise)和输出下降建立时间裕量(SlackFall)如表 1 所示。

表 1 示例时序图的建立时间裕量

ClkPeriod	MinRiseMean (DFF3/CK)	MaxRiseMean (DFF3/D)	RiseConstraint (DFF3/CK→DFF3/D)	SlackRise (DFF3/D)
10	4.5	-10.3	**-0.5**	**3.7**

ClkPeriod	MinRiseMean (DFF3/CK)	MaxFallMean (DFF3/D)	FallConstraint (DFF3/CK→DFF3/D)	SlackFall (DFF3/D)
10	4.5	-10.5	**-0.6**	**3.4**

在此关注图 2 时序图中 XOR1 单元 Z 端处 T_{at} 的计算。首先，XOR1 单元具有两条时序弧(timing arc)，即 A→Z 和 B→Z，由于 T_{at} 需要按最坏情况进行计算，因此需要考虑 A→Z 和 B→Z 两条时序弧(timing arc)的延时；其次，由于 XOR1 单元的功能是异或，因此 A→Z 和 B→Z 这两条时序弧既不是正单极性(positive unate)也不是负单极性(negative unate)的，被称为非单极性(non-unate)的，需要分解为正单极性(positive unate)和负单极性(negative unate)这两类情况分别计算。因此，对于 XOR1 单元 Z 端处的 T_{at} 的输出上升最大值(MaxRiseMean)和输出下降最大值(MaxFallMean)，需要计算 2×2=4 种情况并取最坏情况。由于此处是建立时间检查中的 T_{at} 计算，因此最坏情况为最大值(max)，具体如表 2 所示。

表 2 XOR1 单元 Z 端的最坏(max) T_{at}

arc	unate	max rise mean	max fall mean
A→Z	pos	**8.8**	**9.3**
A→Z	neg	**9.2**	**8.9**
B→Z	pos	**9.0**	**9.1**
B→Z	neg	**9.0**	**9.1**
max		**9.2**	**9.3**

对于 SSTA，需要处理两类原子(atom)操作，包括求和(sum)和求最大值(max)。与 sum 操作不同，max 是非线性操作，因此对于多个随机量（例如表 2 中的 4 种不同的 T_{at}），哪怕是互相独立的高斯量，也会导致其新的随机量 max 不再服从高斯分布(skewness=0)，而是需要在新的随机量 max 的均值(mean)和标准差(sigma)之外，进一步考虑其峰度(skewness)及其对随机量 max 分布的影响。

3. 全局裕量计算

本赛题要求计算时序图中的所有节点的全局裕量(global slack)，即经过该节点的所有时序路径的 slack 的最坏值，或者说经过该节点的所有时序路径的值最小（负值同样取最小值）的 slack。注意 global slack 的计算发生在计算完所有 endpoint slack 之后，此时已经对时序图全图节点进行过遍历并计算了其 T_{at}，因此可利用每个节点的 T_{at}，避免重复遍历时序图各节点。但同时，由于每个 endpoint slack 的计算仅考虑最坏情况，因此需要注意到在计算各节点最坏情况的 T_{at} 时，需要兼顾计算 global slack 所需的其他 T_{at}。

参考资料

[1] Bhasker J, Chadha R. “Static timing analysis for nanometer designs: A practical approach,” Springer Science & Business Media, 2009.

[2] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, “Statistical Timing Analysis: From Basic Principles to State of the Art,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 4, pp. 589–607, Apr. 2008

[3] A. Devgan and C. Kashyap, “Block-based static timing analysis with uncertainty,” in ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486), Nov. 2003, pp. 607–614.

[4] C. S. Amin et al., “Statistical Static Timing Analysis: How Simple Can We Get?,” in Proceedings of the 42Nd Annual Design Automation Conference, in DAC ’05. New York, NY, USA: ACM, 2005, pp. 652–657.

[5] C. Visweswariah et al., “First-Order Incremental Block-Based Statistical Timing Analysis,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 10, pp. 2170–2180, Oct. 2006.

[6] T.-W. Huang and M. D. F. Wong, “OpenTimer: A high-performance timing analysis tool,” in 2015

IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2015, pp. 895–902.

[7] T.-W. Huang, G. Guo, C.-X. Lin, and M. D. F. Wong, “OpenTimer v2: A New Parallel Incremental Timing Analysis Engine,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1–1, 2020.