静态时序分析（Static Timing Analysis）基础及应用

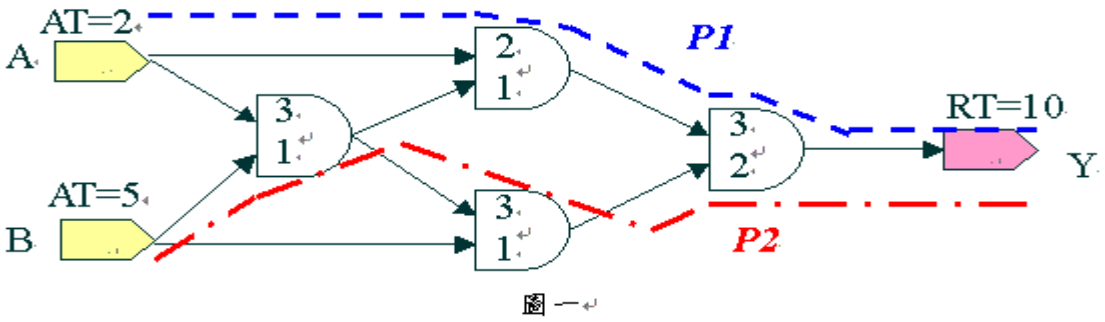
◎陈麒旭

前言

在制程进入深次微米世代之后，芯片（IC）设计的高复杂度及系统单芯片（SOC）设计方式兴起。此一趋势使得如何确保 IC 质量成为今日所有设计从业人员不得不面临之重大课题。静态时序分析（Static Timing Analysis 简称 STA）经由完整的分析方式判断 IC 是否能够在使用者的时序环境下正常工作，对确保 IC 质量之课题，提供一个不错的解决方案。然而，对于许多 IC 设计者而言，STA 是个既熟悉却又陌生的名词。本文将力求以简单叙述及图例说明的方式，对 STA 的基础概念及其在 IC 设计流程中的应用做详尽的介绍。

什么是 STA?

STA 的简单定义如下：**套用特定的时序模型（Timing Model），针对特定电路分析其是否违反设计者给定的时序限制（Timing Constraint）**。以分析的方式区分，可分为 Path-Based 及 Block-Based 两种。



图一

先来看看 Path-Based 这种分析方式。如图一所示，信号从 A 点及 B 点输入，经由 4 个逻辑闸组成的电路到达输出 Y 点。套用的 Timing Model 标示在各逻辑闸上，对于所有输入端到输出端都可以找到相对应的延迟时间。而使用者给定的 Timing Constraint 为：

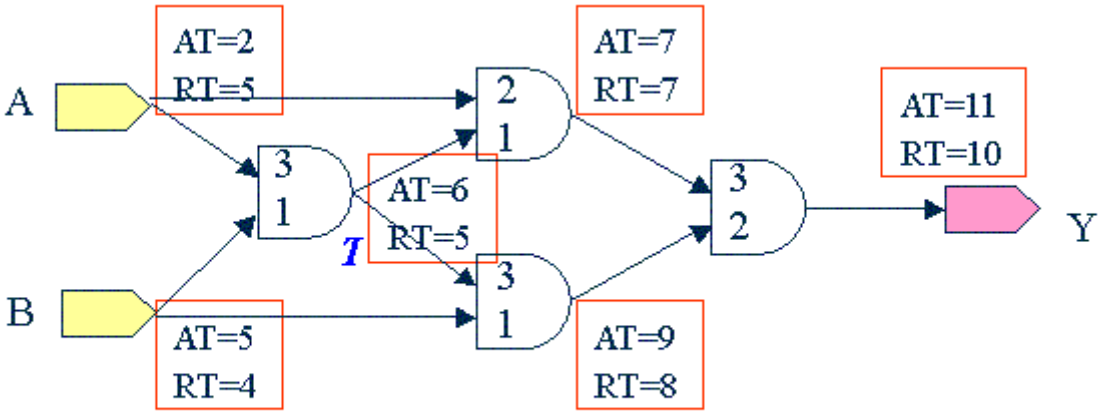
1. 信号 A 到达电路输入端的时间点为 2（AT=2，AT 为 Arrival Time）。
2. 信号 B 到达电路输入端的时间点为 5（AT=5）。
3. 信号必须在时间点 10 之前到达输出端 Y（RT=10，RT 为 Required Time）。

现在我们针对 P1 及 P2 两条路径（Path）来做分析。P1 的起始点为 A，信号到达时间点为 2。经过第 1 个逻辑闸之后，由于此闸有 2 单位的延迟时间，所以信号到达此闸输出的时间点为 4（2+2）。依此类推，信号经由 P1 到达输出 Y 的时间点为 7（2+2+3）。在和上述第三项 Timing Constraint 比对之后，我们可以得知对 P1 这个路径而言，时序（Timing）是满足使用者要求的。

按照同样的方式可以得到信号经由路径 B 到达输出 Y 的时间点为 11（5+1+3+2），照样和上述第三项 Timing Constraint 比对，我们可以得知对 P2 这个路径而言，Timing 是不满足使用者要求的。

对图一的设计而言，总共有 6 个信号路径。对于采用 Path-Based 分析方式的 STA 软件来说，它会对这 6 个信号路径作逐一的分析，然后记录下结果。IC 设计者藉由检视其分析报告的方式来判断所设计的电路是否符合给定的 Timing Constraint。由于最常用来做静态时序分析验证核可（STA Signoff）的 EDA 软件 PrimeTime™ 采用 Path-Based 的分析方式，所以本文将以 Path-Based 的分析方式介绍为主。

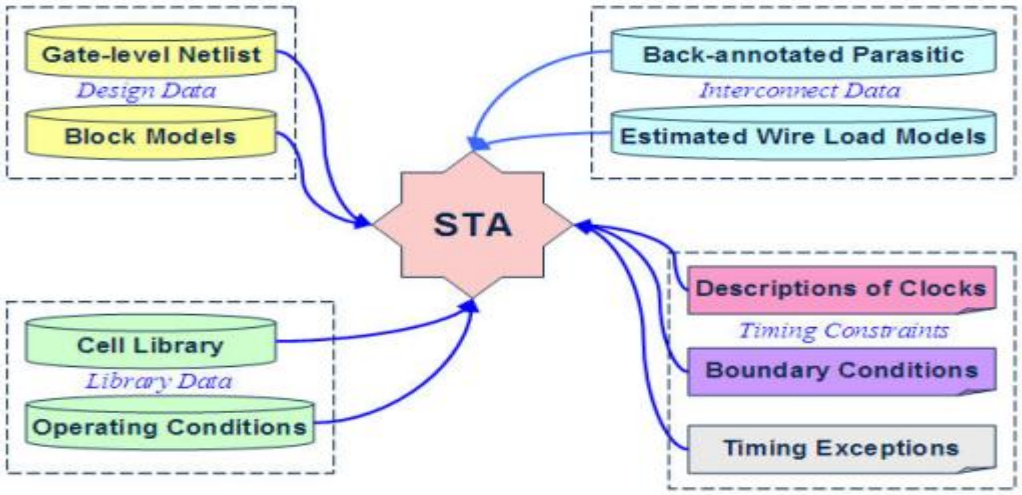
再来看看 Block-Based 的分析方式。此时时序信息（Timing Information）的储存不再是以路径为单位，而是以电路节点（Node）为单位。由 Timing Constraint 我们仅能得知 A 节点的 AT 为 2，B 节点的 AT 为 5 以及 Y 节点的 RT 为 10。Block-Based 的分析方式会找出每个节点的 AT 和 RT，然后比对这两个数值。当 RT 的值大于 AT 时表示信号比 Timing Constrain 中要求的时间还早到达，如此则 Timing 是满足的，反之则不满足。



图二

STA 资料准备

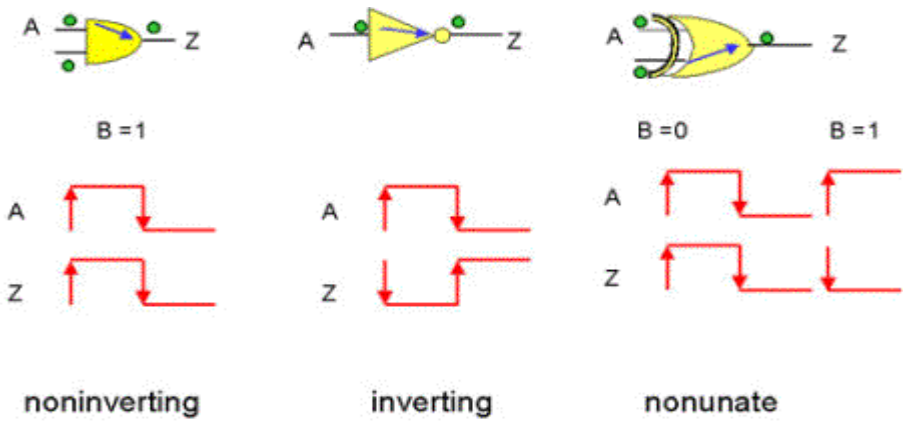
在做 STA 之前，我们必须对其准备工作有充分的了解。STA 所需的资料如图三所示，以下我们分项说明。其中 Design Data 部分，由于 Block Model 和 STA 软件相关性太高，我们不在此加以说明，请直接参阅您 STA 软件的使用手册。



图三

Library Data：

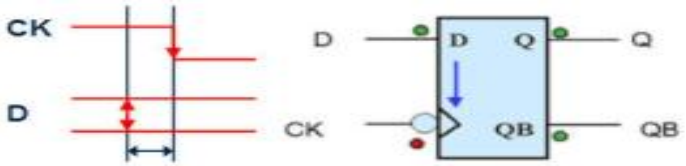
STA 所需要的 Timing Model 就存放在标准组件库（Cell Library）中。这些必要的时序信息是以 Timing Arc 的方式呈现在标准组件库中。Timing Arc 定义逻辑闸任两个端点之间的时序关系，其种类有 Combinational Timing Arc、Setup Timing Arc、Hold Timing Arc、Edge Timing Arc、Preset and Clear Timing Arc、Recovery Timing Arc、Removal Timing Arc、Three State Enable & Disable Timing Arc、Width Timing Arc。其中第 1、4、5、8 项定义时序延迟，其它各项则是定义时序检查。



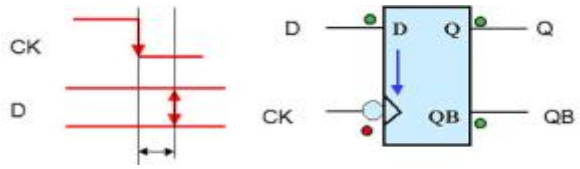
图四

Combinational Timing Arc 是最基本的 Timing Arc。Timing Arc 如果不特别宣告的话，就是属于此类。如图四所示，他定义了从特定输入到特定输出（A 到 Z）的延迟时间。Combinational Timing Arc 的 Sense 有三种，分别是 inverting（或 negative unate），non-inverting（或 positive unate）以及 non-unate。当 Timing Arc 相关之特定输出（图四 Z）信号变化方向和特定输入（图四 A）信号变化方向相反（如输入由 0 变 1，输出由 1 变 0），则此 Timing Arc 为 inverting sense。反之，输出输入信

号变化方向一致的话，则此 Timing Arc 为 non-inverting sense。当特定输出无法由特定输入单独决定时，此 Timing Arc 为 non-unate。

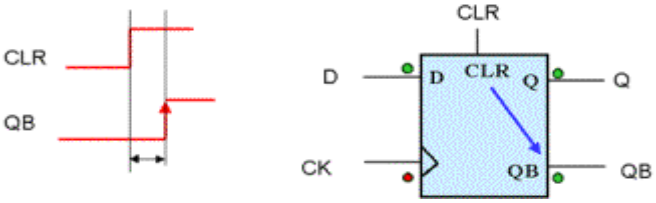


图五

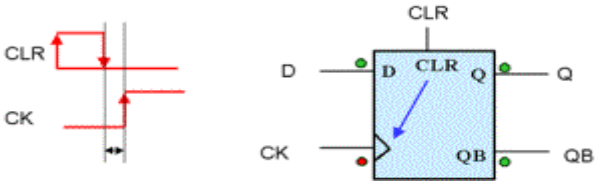


图六

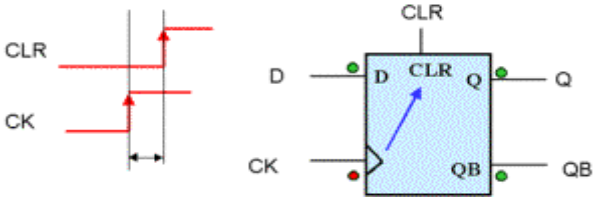
图七



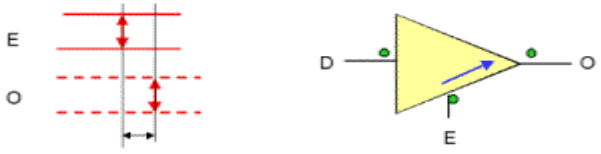
图八



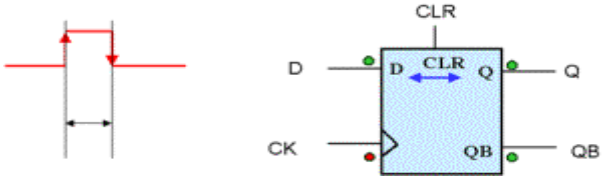
图九



图十



图十一



图十二

其它的 Timing Arc 说明如下。

Setup Timing Arc：定义序向组件（Sequential Cell，如 Flip-Flop、Latch 等）所需的 Setup Time，依据 Clock 上升或下降分为 2 类（图五）。

Hold Timing Arc：定义序向组件所需的 Hold Time，依据 Clock 上升或下降分为 2 类（图六）。

Edge Timing Arc：定义序向组件 Clock Active Edge 到数据输出的延迟时间，依据 Clock 上升或下降分为 2 类（图七）。

Preset and Clear Timing Arc：定义序向组件清除信号（Preset 或 Clear）发生后，数据被清除的速度，依据清除信号上升或下降及是 Preset 或 Clear 分为 4 类（图八）。这个 Timing Arc 通常会被取消掉，因为它会造成信号路径产生回路，这对 STA 而言是不允许的。

Recovery Timing Arc：定义序向组件 Clock Active Edge 之前，清除信号不准启动的时间，依据 Clock 上升或下降分为 2 类（图九）。

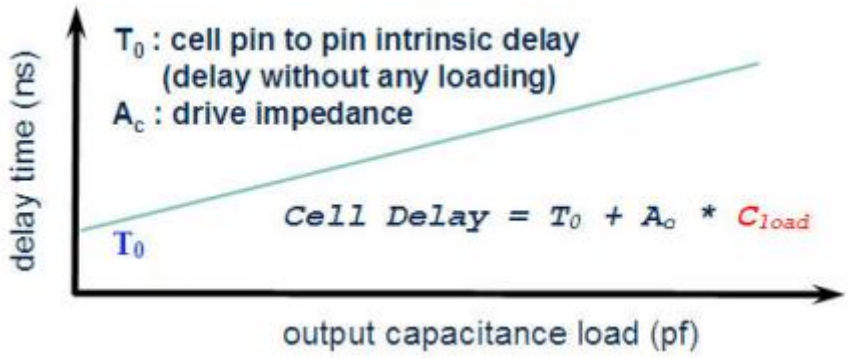
Removal Timing Arc：定义序向组件 Clock Active Edge 之后，清除信号不准启动的时间，依据 Clock 上升或下降分为 2 类（图十）。

Three State Enable & Disable Timing Arc：定义 Tri-State 组件致能信号（Enable）到输出的延迟时间，依据 Enable 或 Disable 分为 2 类。（图十一）

Width Timing Arc：定义信号需维持稳定的最短时间，依据信号维持在 0 或 1 的位准分为 2 类。（图十二）

上文列出了标准组件库内时序模型的项目，但对其量化的数据却没有加以说明。接下来，我们就来看看到底这些时序信息的确实数值是如何定义在标准组件库中的。

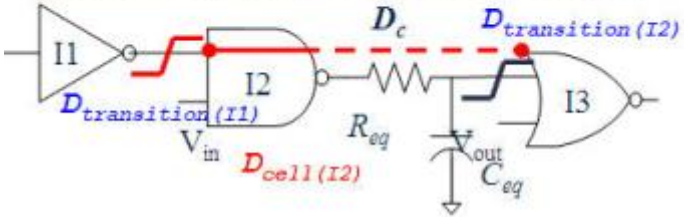
以 Combinational Timing Arc 为例，信号从输入到输出的延迟时间可以描述成以输入的转换时间（Transition Time）和输出的负载为变量的函数。描述的方式可以是线性的方式，如图十三所示。也可以将这 2 个变量当成指针，建立时序表格（Timing Table），让 STA 软件可以查询出正确的延迟时间。这种以表格描述的方式会比上述线性描述的方式准确许多，因此现今市面上大部分的标准组件库皆采用产生时序表格的方式来建立 Timing Model。



图十三

我们举个简单的例子来说明 STA 软件如何从时序表格计算出组件延迟时间。（图十四）

Output Capacitance	Input Transition		
	0	0.5	1
0.1	0.123	0.234	0.456
0.2	0.222	0.432	0.801



图十四

组件延迟时间（D_{delay}）：输入达逻辑 1 位准 50%到输出达逻辑 1 位准 50%的时间。

组件转换时间（$D_{transition}$）：输出达逻辑 1 位准 20%（80%）到 80%（20%）的时间。

$$\Rightarrow D_{cell}(I2) = f(D_{transition}(I1), C_{eq}) = f(0.5, 0.2) = 0.432$$
$$\Rightarrow D_{transition}(I2) = g(D_{transition}(I1), C_{eq})$$

当输入的转换时间为 0.5，输出负载为 0.2 时，可由图十四的时序表格查得组件 I2 的延迟时间为 0.432。而由于表格的大小有限，对于无法直接由表格查询到的延迟时间（如输入转换时间 0.25，输出负载 0.15），STA 软件会利用线性内插或外插的方式计算延迟时间。

对于其它的 Timing Arc，不管是时序延迟或时序检查，其相对应的时序数值计算和上例的计算方式是一样的。

接下来我们说明操作环境（Operating Condition）对时序的影响。操作环境指的是制程（Process）、电压（Voltage）、温度（Temperature）三项因子。这三项因子通常会被简称为 PVT，其对时序的影响可用下方线性方程式来描述。其中 nom_process、nom_voltage 及 nom_temperature 会定义在标准组件库中，代表建立时序表格时的操作环境。

$$D_{new} = D(1+\Delta_P K_P)(1+\Delta_V K_V)(1+\Delta_T K_T)$$

$$\Delta_P = Process - nom\_process$$

$$\Delta_V = Voltage - nom\_voltage$$

$$\Delta_T = Temperature - nom\_temperature$$

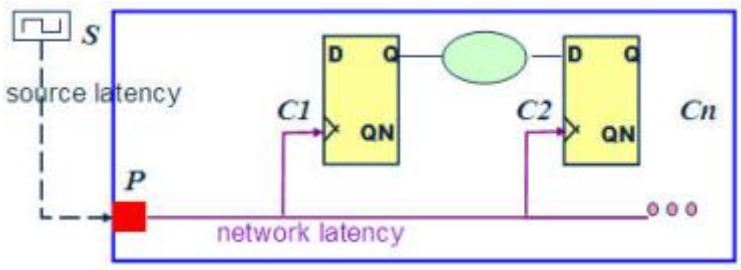
Interconnect Data：

在「什么是 STA」段落的例子中，为了方便说明，我们并没有把逻辑闸和逻辑闸间的联机延迟（Interconnect Delay）考虑在内。事实上，许多 DSM IC 设计之时序表现是由联机延迟主导的，其重要性不容我们忽视。

联机延迟依照布局与绕线（P&R）前后有不同的考虑。在布局与绕线前，组件在芯片中摆放的位置尚未确定，所以联机延迟是一个预估值。而在布局与绕线之后，联机延迟则是根据实际绕线计算出来的。对布局与绕线之前的联机延迟，通常是用 Wireload Model 来预估。Wireload Model 根据芯片面积的预估大小及联机驱动组件数目（Fan-out）的多寡来决定联机的电阻和电容值，STA 软件则利用这些电阻电容值计算出联机延迟。在布局与绕线之后，可以利用电阻电容萃取（RC Extraction）软件将绕线图形转换成实际的电阻电容电路，然后贴回（Back-annotate）STA 软件计算联机延迟。

Timing Constraints：

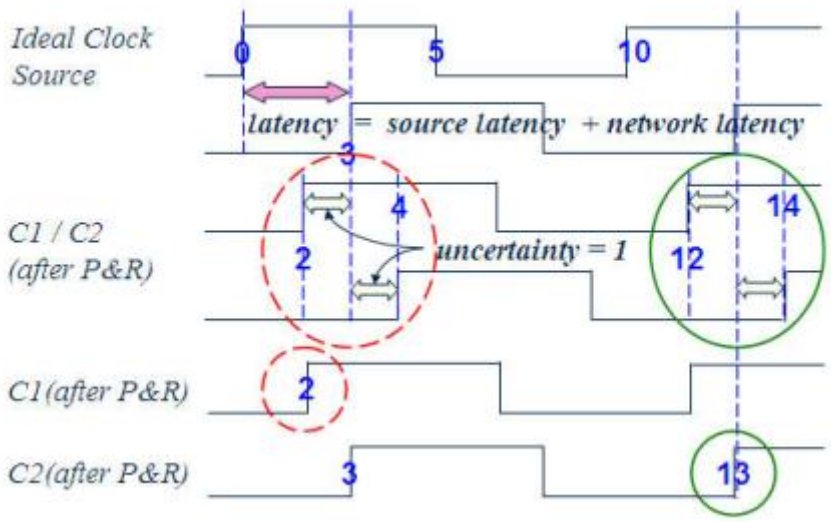
Timing Constraint 为使用者所给定，用来检验设计电路时序的准则。其中最重要的一项就是频率（Clock）的描述。对于一个同步电路而言，缓存器和缓存器之间的路径延迟时间必须小于一个 Clock 周期（Period），也就是说，当我们确认了 Clock 规格，所有缓存器间的路径的 Timing Constraint 就会自动给定了。



图十五

Clock 规格包含波形、Latency 及 Uncertainty 的定义。波形定义一个 Clock 的周期及信号上升缘及下降缘的时间点。Latency 定义从 Clock 来源到序向组件 Clock 输入端的延迟时间。Uncertainty 则定义 Clock 信号到序向组件 Clock 输入端可能早到或晚到的时间。

如果上面的文字让你有不知所云的感觉，那底下看图说故事的解说也许会让你有比较清晰的概念。在图十五的电路中，左边的正反器（Flip-Flop）在第一个 Clock 上升缘时会丢出数据，此数据会在第二个 Clock 上升缘让右边的 Flip-Flop 撷取。要分析右边的 Flip-Flop 能否正确撷取数据就必须知道第一个 Clock 上升缘到达节点 C1 的时间点和第二个上升缘到达节点 C2 的时间点。假设在时间点为 0 的时候，Clock 信号由 S 点出发，经过一段时间（source latency，1 个时间单位，仿真芯片外的 Clock 延迟时间，例如板子上的绕线产生的信号延迟时间）到达电路的 Clock 输入端点 P，接下来再经过一段时间（芯片内 Clock 绕线造成的信号延迟时间），Clock 信号分别到达 C1 和 C2 节点。如果电路已经进行布局与绕线，输入端点 P 到 C1 和 C2 的信号延迟时间可由联机上的寄生电阻电容计算得来。比方说，经过计算发现信号由 P 传递到 C1 需要 1 个时间单位，由 P 传递到 C2 需 2 个时间单位，则 Clock 信号第一个上升缘到达 C1 和第二个上升缘到达 C2 的时间点就会如图十六下方两列所示，分别为时间点 2 和 13（因为加上了 1 个时间单位的 source latency）。



图十六

在布局与绕线之前，我们无法准确得知 P 到 C1 和 C2 的信号延迟时间，仅能先做个预估。图十五的 network latency 及上文提到的 Uncertainty 就是用来做此种预估的。先假设我们拥有某种完美的布局与绕线软件可以让 Clock 输入端点 P 到所有 Flip-Flop 的 Clock 输入端的信号延迟时间一模一样，那么我们只要知道这个信号延迟时间就可以得到 Clock 信号到达 C1 和 C2 的时间点了。这个信号延迟时间可以藉由电路特性（如预估面积大小，Flip-Flop 数目等）来做预估，

而这个预估值就是所谓的 network latency。如果这种完美的软件存在的话，那 Clock 的上升缘到达 C1 和 C2 的时间点就可以由 Latency（source latency + network latency）计算出来。

很不幸的，世界上没有这么完美的软件，在布局与绕线后 Clock 输入端点 P 到所有 Flip-Flop 的 Clock 输入端的信号延迟时间不会完全一样。也就是说 Clock 的某个上升缘不会同时到达 C1 和 C2。因此我们要对上述的预估值做些修正，加入 Uncertainty 的描述来定义 Clock 上升缘左右移动的可能范围。在图十六中，Uncertainty 为 1 个时间单位，所以 Clock 第一个上升缘会在时间点 3（因为 Latency 为 3）左右 1 时间单位范围内（也就是时间点 2 到时间点 4）到达 C1，。第二个上升缘则会在时间点 12 到 14 的范围内到达 C2。

除了 Clock 之外，对于电路其它输出输入端点及其周边的环境（Boundary Condition）也要加以描述。在说明 Boundary Condition 之前，我们得对路径（Path）有更进一步的了解。上文曾提及 STA 会将电路中所有的 Path 找出来加以分析，但 Path 的定义是什么呢？

Path 根据起点及终点可以分为 4 种：

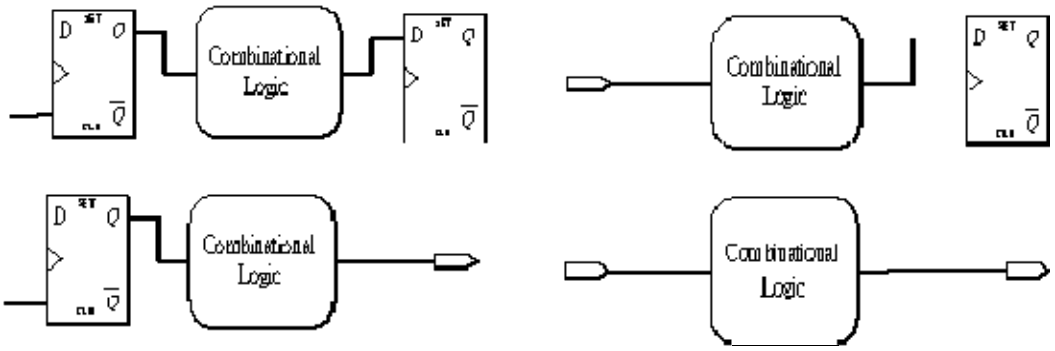
1. 由 Flip-Flop Clock 输入到 Flip-Flop 数据输入（图十七左上）。

2. 由主要输入（Primary Input，简称 PI）到 Flip-Flop 数据输入（图十七右上）。

3. 由 Flip-Flop Clock 输入到主要输出（Primary Output，简称 PO）（图十七左下）。

4. 由主要输入到主要输出（图十七右下）。

当 Clock 规格确定了之后，第 1 种 Path 的时序限制（Timing Constraint）就自动的给定了。为了给定其它 3 种 Path 的时序限制，我们必须定义 Boundary Condition。



图十七

一般来说，我们会定义下列的 Boundary Condition：

1. Driving Cell：定义输入端点的推动能力（图十八）。

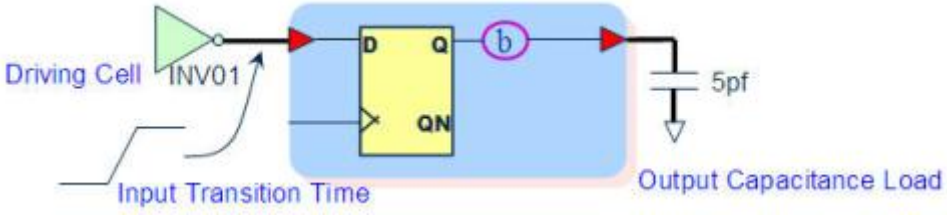
2. Input Transition Time：定义输入端点的转换时间（图十八）。

3. Output Capacitance Load：定义输出负载（图十八）。

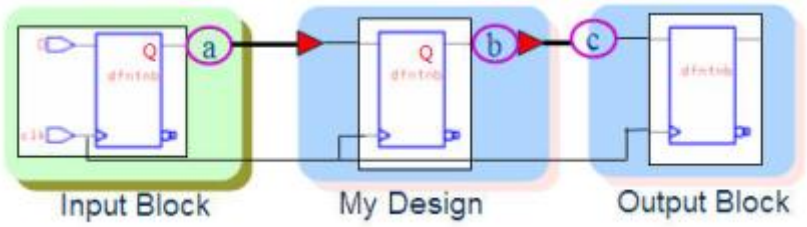
4. Input Delay：输入端点相对于某个 Clock 领域的延迟时间。（图十九，$Delay_{clk-Q}$ + a）

5. Output Delay：自输出端点往外看相对于某个 Clock 领域的延迟时间。（图十九，c）

在这些 Boundary Condition 定义之后，上述 4 种 Path 事实上都可看成是第 1 种 Path（Flip-Flop 到 Flip-Flop）。也就是说，加上 Boundary Condition 后，只要 Clock 给定，所有 Path 的 Timing Constraint 就会自动给定。。



图十八

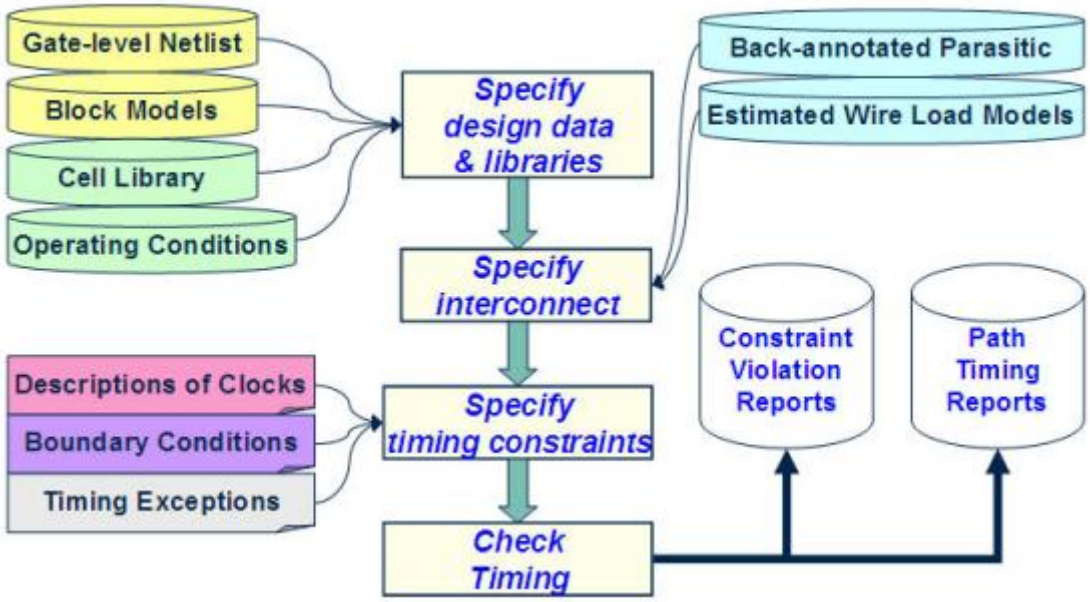


图十九

由于每个 Path 都有 Timing Constraint，所以时序分析都能够进行。但在某些情况下，有些 Path 的分析可能没有意义，因此妳会想忽略这些 Path 的分析。或是有些 Path 分析的方式不一样，妳会想指定这些 Path 的分析方式。此时就要设定一些 Timing Exception，如 False Path 和 Multi-cycle Path 等等来处理非一般性的时序分析。

STA 流程及分析方式

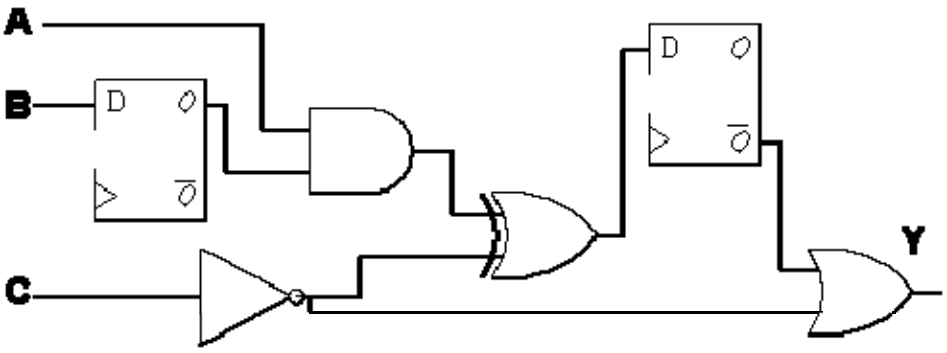
STA 的流程如图二十所示，而其分析验证的项目就是我们前文提及之时序检查相关的 Timing Arc，如 Setup Time、Hold Time 等等。以下我们针对 Setup Time 举 1 实际范例来说明 STA 的分析方式。



图二十

Setup Time

设计电路如图二十一所示，时序模型（Timing Model）及时序限制（Timing Constraint）如下：



图二十一

所有逻辑闸在输出信号上升时最长的延迟时间为 3ns，最短为 2ns。

所有逻辑闸在输出信号上升时最长的延迟时间为 2ns，最短为 1ns。

所有联机（Net）最长的延迟时间为 2ns，最短为 1ns。

所有 Flip-Flop Clock 到 Q 的延迟时间为 3ns。

所有 Flip-Flop 的 Setup Time 为 1ns（T_s）。

所有 Flip-Flop 的 Hold Time 为 1ns（T_h）。

Clock 周期为 14ns（D_{clkp}）。

Clock source latency 为 2ns（D_{clks}）。

Clock network latency 为 3ns（D_{clkn}）。

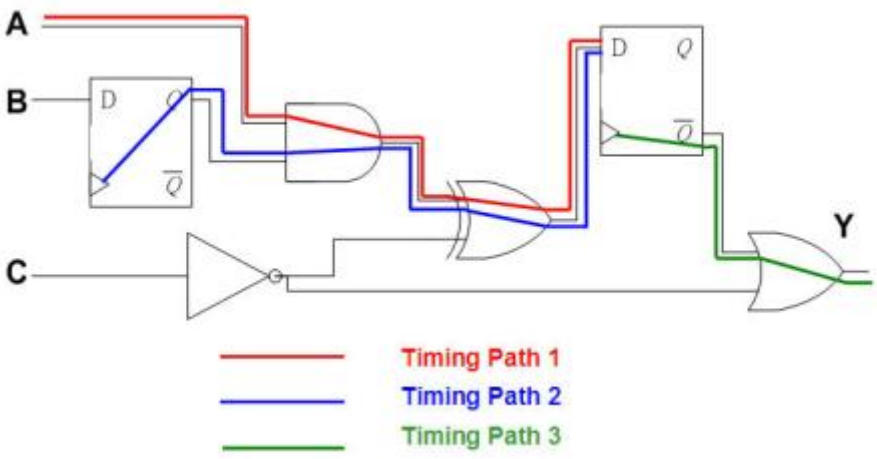
Clock uncertainty 为 1ns（D_{clku}）。

B 及 C 的 input delay 皆为 1ns（D_a、D_b、D_c）。

Y 的 output delay 为 3ns（D_Y）。

接下来，我们以 Step-By-Step 的方式说明时序分析的方式。

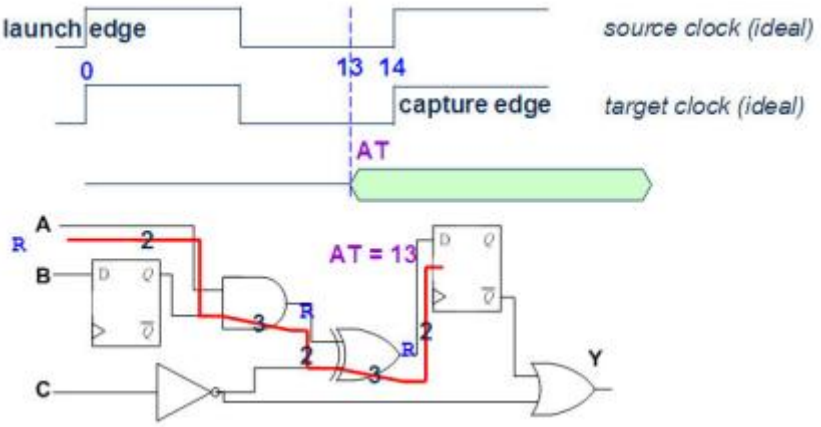
1. 首先找出所有 Timing Path，我们只列出具代表性的 3 条 Timing Path 来加以说明。



图二十二

2. 假设输入 A 信号由 0 变 1，计算第 1 条 Path 终点信号到达的时间（Arrival Time 简称 AT）。

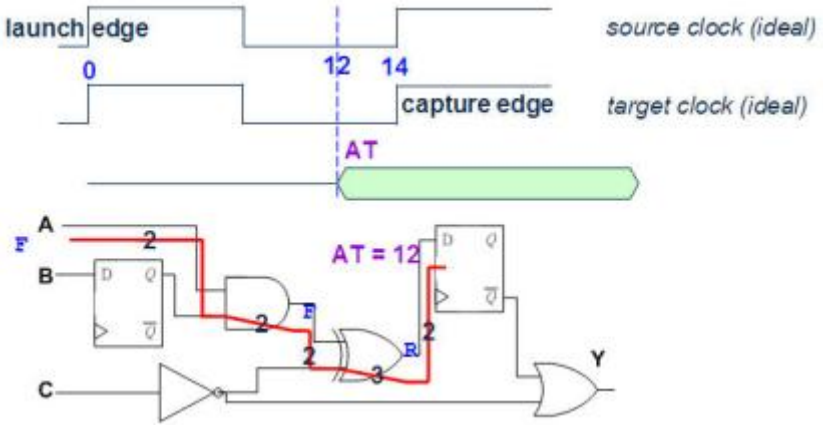
$$AT = D_c + 2 + 3 + 2 + 3 + 2 = 13ns$$



图二十三

3. 假设输入 A 信号由 1 变 0，计算第 1 条 Path 终点 AT。

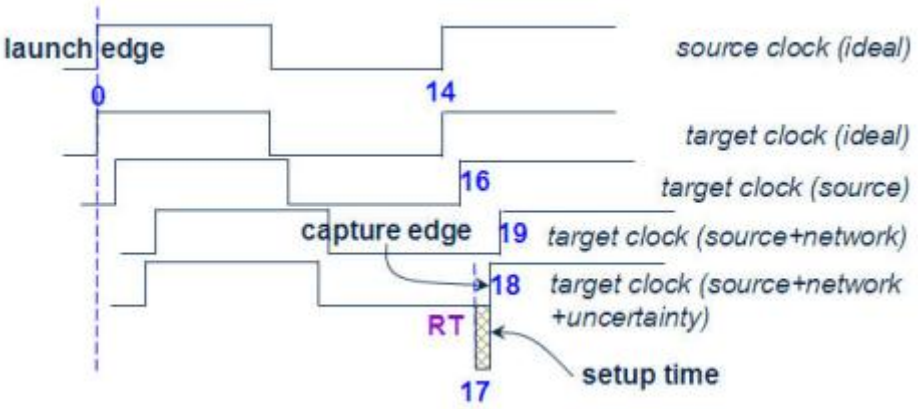
$$AT = D_c + 2 + 2 + 2 + 3 + 2 = 12ns$$



图二十四

4. 计算第 1 条 Path 终点的需求时间（Required Time，简称 RT）。

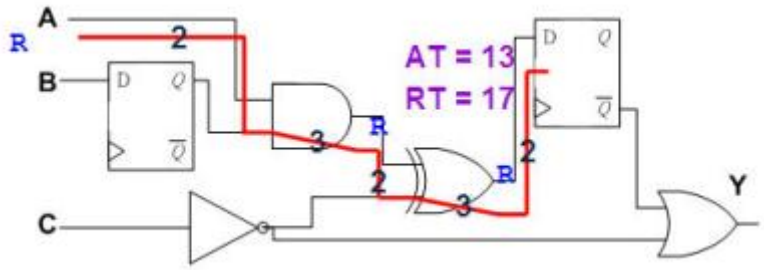
$$RT = D_{clk} + D_{clk} + D_{clk} - D_{clk} - T_s = 14 + 2 + 3 - 1 - 1 = 17ns$$



图二十五

5. 假设输入 A 信号由 0 变 1，计算第 1 条 Path 终点的 Slack。Slack 等于 RT 和 AT 的差值，对于 Setup Time 验证来说等于 RT - AT，对于 Hold Time 验证来说等于 AT - RT。在此 Setup Time 范例中，Slack 为正，表示信号实际到达 Path 终点时间比必须到达的时间还早，因此 Timing 是满足的。

$$Slack = RT - AT = 17 - 13 = 4ns$$

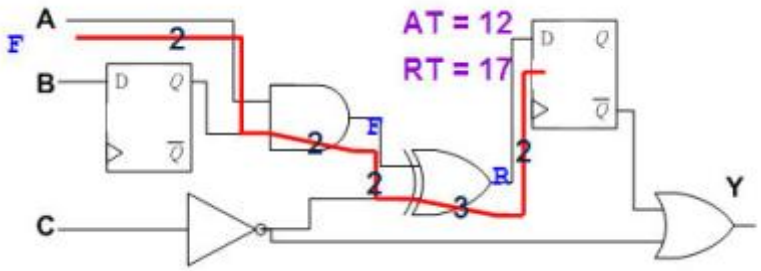


图二十六

6. 假设输入 A 信号由 1 变 0，计算第 1 条 Path 终点的 Slack。Slack 为正，因此 Timing 是满足的。

$$Slack = RT - AT = 17 - 12 = 5ns$$

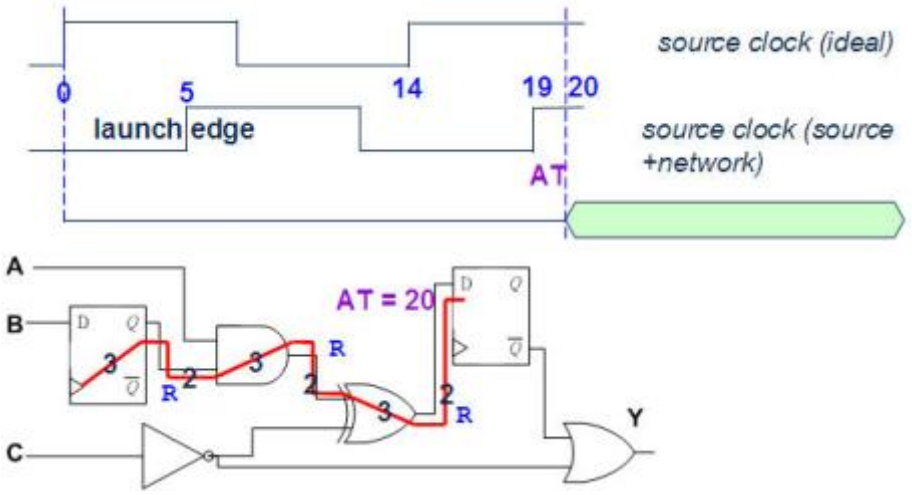
综合 5 和 6，第 1 条 Path 的 Timing 是符合规格的，其 Slack 为 4ns（取较差状况）。



图二十七

7. 假设前级 Flip-Flop 的信号由 0 变 1，计算第 2 条 Path 终点的 AT。

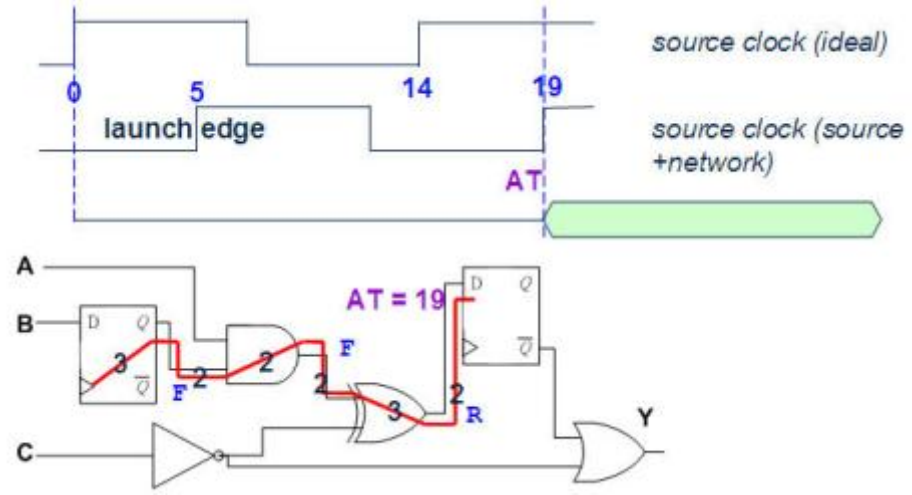
$$AT = D_{clk} + D_{clkq} + 3 + 2 + 3 + 2 + 3 + 2 = 20ns$$



图二十八

8. 假设前级 Flip-Flop 的信号由 1 变 0，计算第 2 条 Path 终点的 AT。

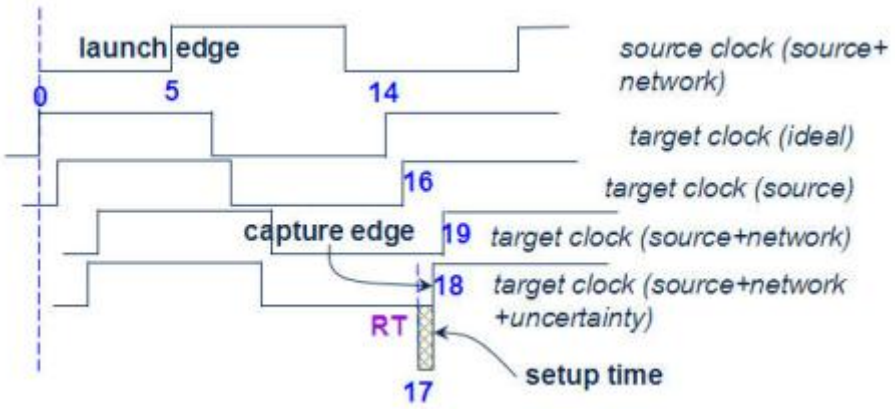
$$AT = D_{clk} + D_{clk} + 3 + 2 + 2 + 2 - 3 + 2 = 20ns$$



图二十九

9. 计算第 2 条 Path 终点的 RT。

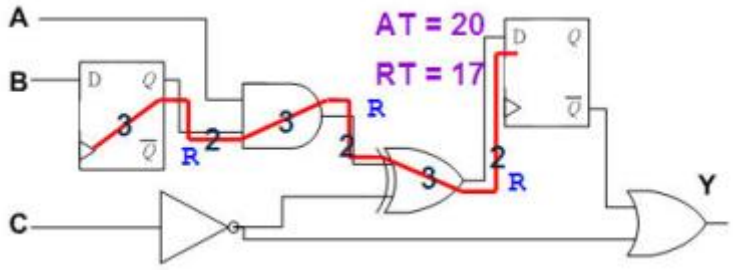
$$RT = D_{clk} + D_{clk} + D_{clk} - D_{clk} - T_s = 14 + 2 + 3 - 1 - 1 = 17ns$$



图三十

10. 假设前级 Flip-Flop 的信号由 0 变 1，计算第 2 条 Path 终点的 Slack。Slack 为负，因此 Timing 不满足。

$$Slack = RT - AT = 17 - 20 = -3$$

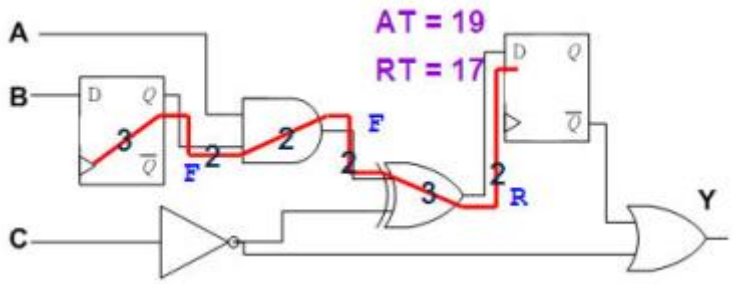


图三十一

11. 假设前级 Flip-Flop 的信号由 1 变 0，计算第 2 条 Path 终点的 Slack。Slack 为负，因此 Timing 不满足。

$$Slack = RT - AT = 17 - 19 = -2$$

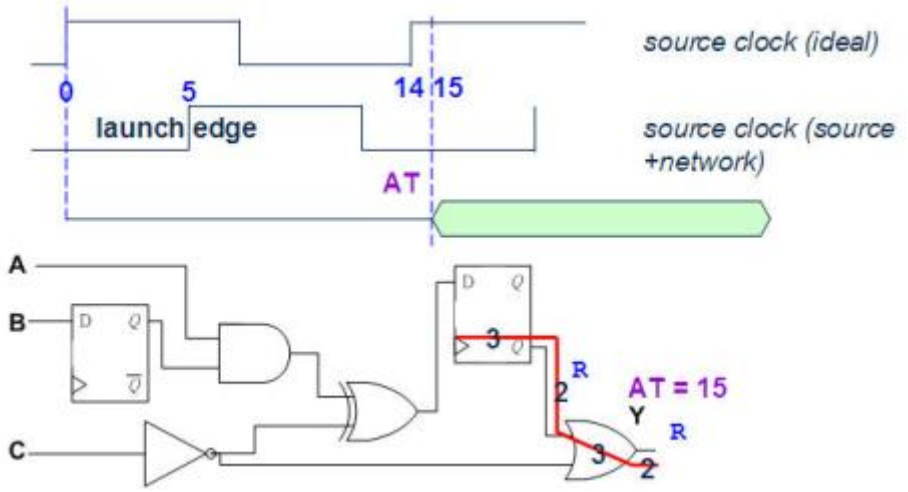
综合 10 和 11，第 2 条 Path 的 Timing 不满足，其 Slack 为-3。



图三十二

12. 假设前级 Flip-Flop 的信号由 0 变 1，计算第 3 条 Path 终点的 AT。

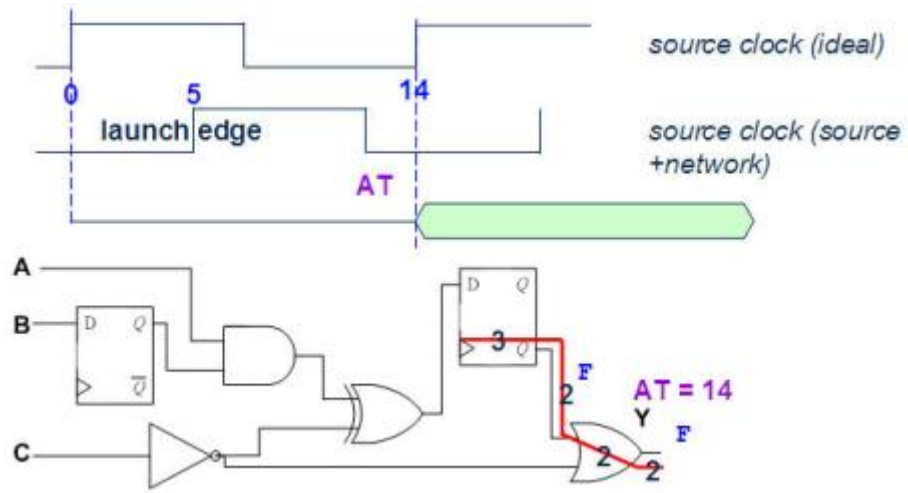
$$AT = D_{clk} + D_{clk} + 3 + 2 + 3 + 2 = 15ns$$



图三十三

13. 假设前级 Flip-Flop 的信号由 1 变 0，计算第 3 条 Path 终点的 AT。

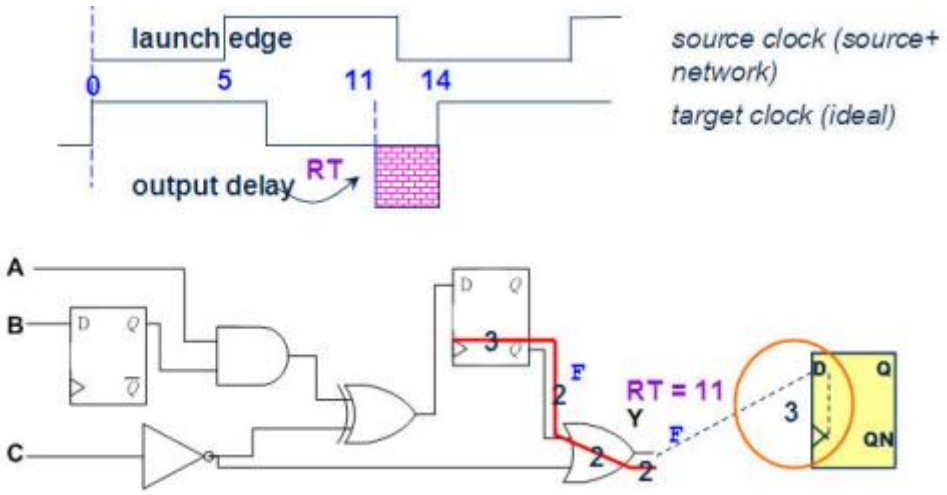
$$AT = D_{clk} + D_{clk} + 3 + 2 + 2 + 2 = 14ns$$



图三十四

14. 计算第 3 条 Path 终点的 RT。

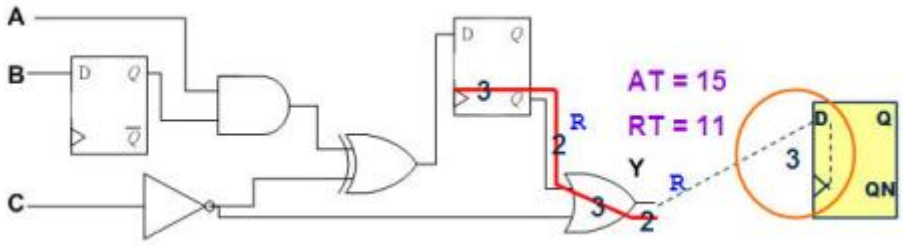
$$RT = D_{clk} - D_t = 14 - 3 = 11ns$$



图三十五

15. 假设前级 Flip-Flop 的信号由 0 变 1，计算第 3 条 Path 终点的 Slack。Slack 为负，因此 Timing 不满足。

$$Slack = RT - AT = 11 - 15 = -4$$

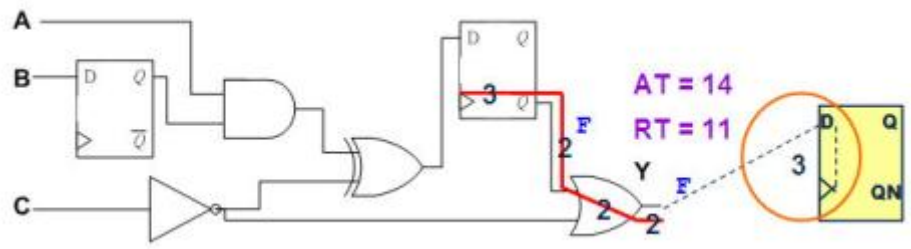


图三十六

16. 假设前级 Flip-Flop 的信号由 1 变 0，计算第 3 条 Path 终点的 Slack。Slack 为负，因此 Timing 不满足。

$$Slack = RT - AT = 11 - 14 = -3$$

综合 15 和 16，第 3 条 Path Timing 不符合规格，其 Slack 为-4。



图三十七

综合上述分析结果，此电路的时序不符合规格，其 Critical Path 是 Path3，Slack 为-4。

总结

本文先对 STA 的概念做概念性的介绍，在下集的文章中，将对 STA 在实际 IC 设计流程中的应用举一范例说明，请各位拭目以待。

静态时序分析（Static Timing Analysis）基础及应用（下）

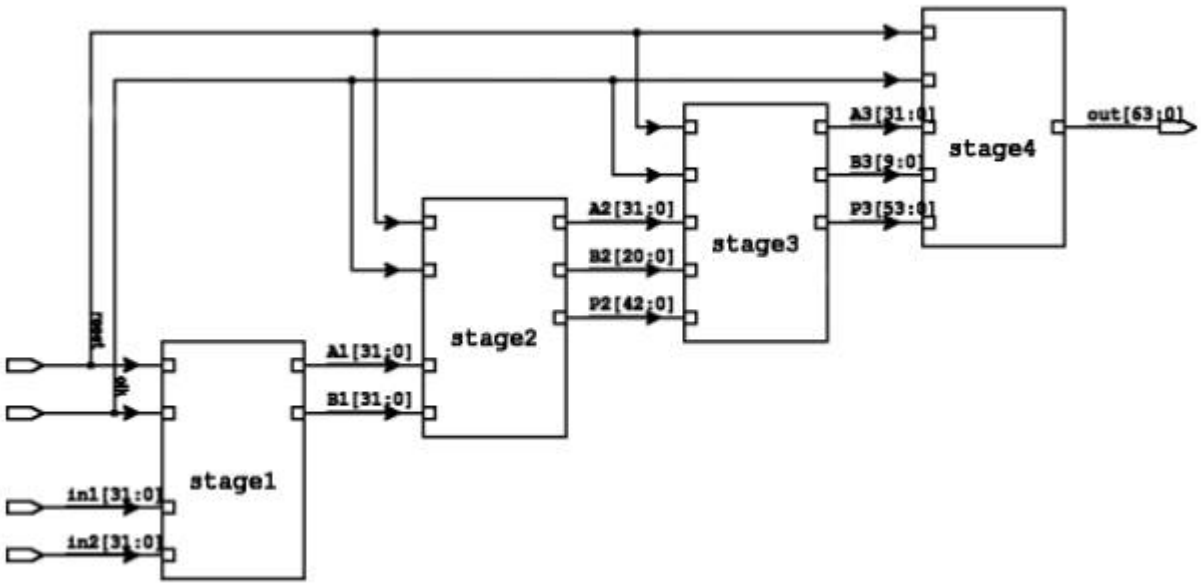
◎陈麒旭

前言

在制程进入深次微米世代之后，芯片（IC）设计的高复杂度及系统单芯片（SOC）设计方式兴起。此一趋势使得如何确保 IC 质量成为今日所有设计从业人员不得不面临之重大课题。静态时序分析（Static Timing Analysis 简称 STA）经由完整的分析方式判断 IC 是否能够在使用者指定的时序下正常工作，对确保 IC 质量之课题，提供一个不错的解决方案。在「静态时序分析（Static Timing Analysis）基础及应用（上）」一文中笔者以简单叙述及图例说明的方式，对 STA 的基础概念做了详尽的说明。接下来，就让我们藉由实际设计范例来了解 STA 在设计流程的应用。

设计范例说明

设计范例为一个 32bit x 32bit 的 Pipeline 乘法器，其架构如图一所示。Pipeline 共分 3 级，电路之输出输入端皆有缓存器储存运算数值。



图一

依据 Cell-based 设计的方式，首先以硬件描述语言设计图一之电路。接下来实作此电路，进行合成（Synthesis）及布局与绕线（P&R）。并在实作的各步骤后进行静态时序分析，确认时序规格是否满足。实作及验证所用到的软件及设计数据库如下所示：

合成：Synopsys™ Design Compiler

布局与绕线：Synopsys™ Astro

设计数据库：Artisan™ 0.18um Cell Library

在接下来的文章中，各位将会看到静态时序分析在实作过程中的应用。藉由实际产生的数据了解在不同实做步骤上时序分析的差异。

时序限制（Timing Constraint）

要作静态时序分析，首先要有时序限制。此设计范例的时序限制如下所述。（→ 后为设定时序限制之 SDC 指令）

1 频率规格（Clock Specification）

1.1 周期：6ns →

```
create_clock -name "MY_CLOCK" -period 6 -waveform {0 3} [get_ports {clk}]
```

1.2 Source Latency：1ns →

```
set_clock_latency -source 1 [get_clocks {MY_CLOCK}]
```

1.3 Network Latency：1ns →

```
set_clock_latency 1 [get_clocks {MY_CLOCK}]
```

1.4 Skew：0.5ns →

```
set_clock_uncertainty 0.5 [get_clocks {MY_CLOCK}]
```

2 周边状况（Boundary Condition）

2.1 输入延迟（Input Delay）：1.2ns →

```
set allin_except_CLK [remove_from_collection [all_inputs] [get_ports clk] ]
set_input_delay $I_DELAY -clock MY_CLOCK $allin_except_CLK
```

2.2 输出延迟（Output Delay）：1.2ns →

```
set_output_delay $O_DELAY -clock MY_CLOCK [all_outputs]
```

2.3 输出负载（Output Loading）：0.5pF →

```
set_load $O_LOAD 0.5 [all_outputs]
```

3 时序例外（Timing Exception）：无

合成软件之时序报告

当 Synopsys™ Design Compiler 将电路合成完毕后，执行下面指令可以产生时序报告：

```
report_timing -path full -delay max -max_paths 10 -input_pins \
-nets -transition_time -capacitance > timing_syn.txt
```

时序报告会储存在 timing_syn.txt 此档案中。在档案的开头不远处，会列出此电路最有可能不符合时序规格的路径（Critical Path）。例如：

Startpoint: **S2/B2_reg_0_**

(rising edge-triggered flip-flop clocked by MY_CLOCK)

Endpoint: **S3/P3_reg_47_**

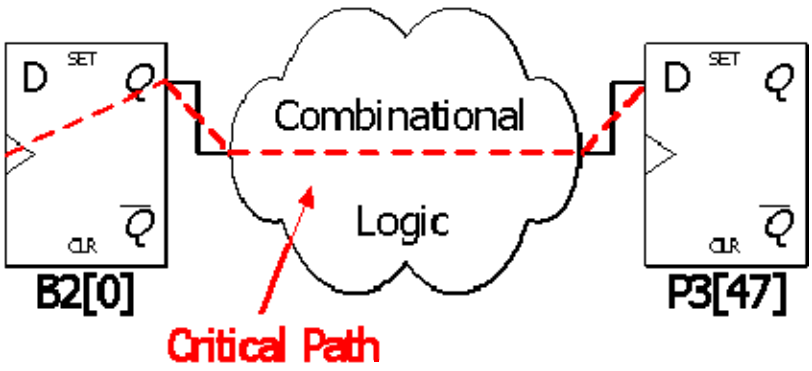
(rising edge-triggered flip-flop clocked by MY_CLOCK)

Path Group: MY_CLOCK

Path Type: max

在这个例子中，Critical Path 的起点 Flip-Flop 是第 2 个 Pipeline Stage 内的 B2 缓存器的第 0 个位，终点 Flip-Flop 则是第 3 个 Pipeline Stage 内的 P3 缓存器的第 47 个位（图二）。

在 Critical Path 报告的下方会有 Wire Load Model 的信息，此范例使用的是 UMC18_Conservative Model。这个 Model 会以较悲观的方式预估联机的延迟时间（Interconnect Delay）。



图二

继续往下检视档案，你会看到 Critical Path 的详细时序信息。例如：

```
   Point                                  Fanout      Cap     Trans
Incr      Path

   ---------------------------------------------------------------
--------------

   clock MY_CLOCK (rise edge)                                        0.00
0.00
```

```
   clock network delay (ideal)                                    2.00
2.00

   S2/B2_reg_0_/CK (DFFHQX4)                              0.00
0.00     2.00r
S2/B2_reg_0_/Q (DFFHQX4)                               0.16     0.30
2.30r
S2/n36 (net)                          1        0.03             0.00
2.30r
S2/U10/A (BUFX20)                                      0.16     0.00
2.30r
S2/U10/Y (BUFX20)                                      0.23     0.21
2.51r
...

    ...

   S3/add_106/U0_5_47/A (XNOR2X2)                        0.18
0.00     7.74f

   S3/add_106/U0_5_47/Y (XNOR2X2)                        0.12
0.22     7.96f

   S3/add_106/SUM[47] (net)              1        0.01
0.00     7.96f

   S3/add_106/SUM[47] (stage3_DW01_add_54_0)
0.00     7.96f

   S3/N94 (net)                                  0.01
0.00     7.96f
S3/P3_reg_47_/D (DFFTRXL)                              0.12     0.00
7.96f

   data arrival time
7.96



   clock MY_CLOCK (rise edge)                                     6.00
6.00

   clock network delay (ideal)
2.00      8.00
```

```
   clock uncertainty
-0.50      7.50

   S3/P3_reg_47_/CK (DFFTRXL)
0.00      7.50r

   library setup time
-0.28      7.22

   data required time
7.22

   ----------------------------------------------------------------
---------------

   data required time
7.22

   data arrival time
-7.96

   ----------------------------------------------------------------
---------------

   slack (VIOLATED)
-0.74
```

先由左往右看，第一个直行 Point 标示出路径中的节点，节点可以是组件的输出入端点，也可以是组件间的联机（Net）。第二个直行 Fanout 标示节点推动的组件个数。第三个直行 Cap 标示出节点推动的负载。第四个直行 Trans 标示出节点上信号的转换时间（Transition Time）。第五个直行 Incr 标示出节点造成的延迟时间。最后一个直行 Path 则是自路径起点到到此节点为止的总延迟时间。

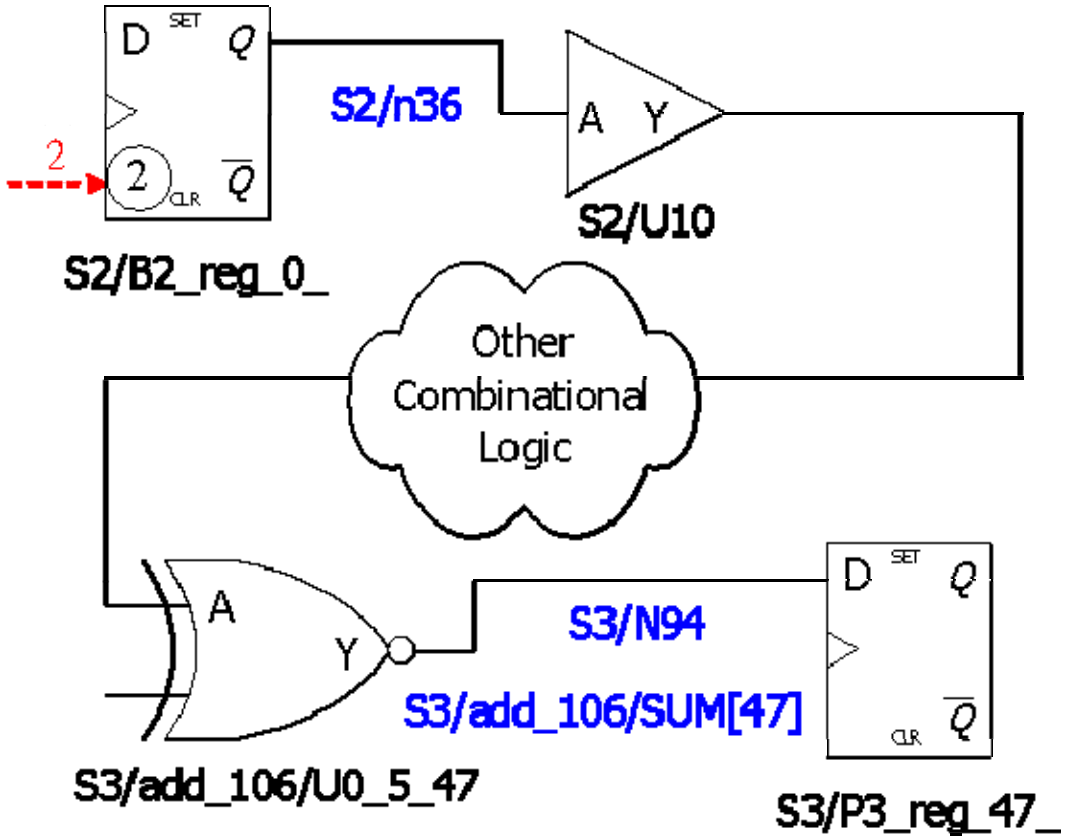
再来我们由上往下检视 Critical Path 的时序信息。

```
   clock              network              delay              (ideal)
2.00      2.00
```

此处的 2ns 的 clock network delay 是由我们给定的时序限制计算而来的，因为我们给定了各 1ns 的 source latency 及 network latency，加起来共有 2ns。

```
   S2/B2_reg_0_/CK (DFFHQX4)                                    0.00
0.00      2.00 r
```

此行表示 Critical Path 的起点为 S2 Instance 下的 B2_reg_0_这个 instance 的 CK 端点。由于有 2ns 的 network delay，所以频率信号到达此节点的时间为 2ns（图三）。至于 0ns 的 Transition Time 则是因为我们没有在频率规格中定义其数值，合成软件的会假设是一个 0ns Transition Time 的理想波形。最右边的 r 是因为这个 Flip-Flop 是正缘触发，所以以 r 表示。如果是 f 就是负缘触发。



图三

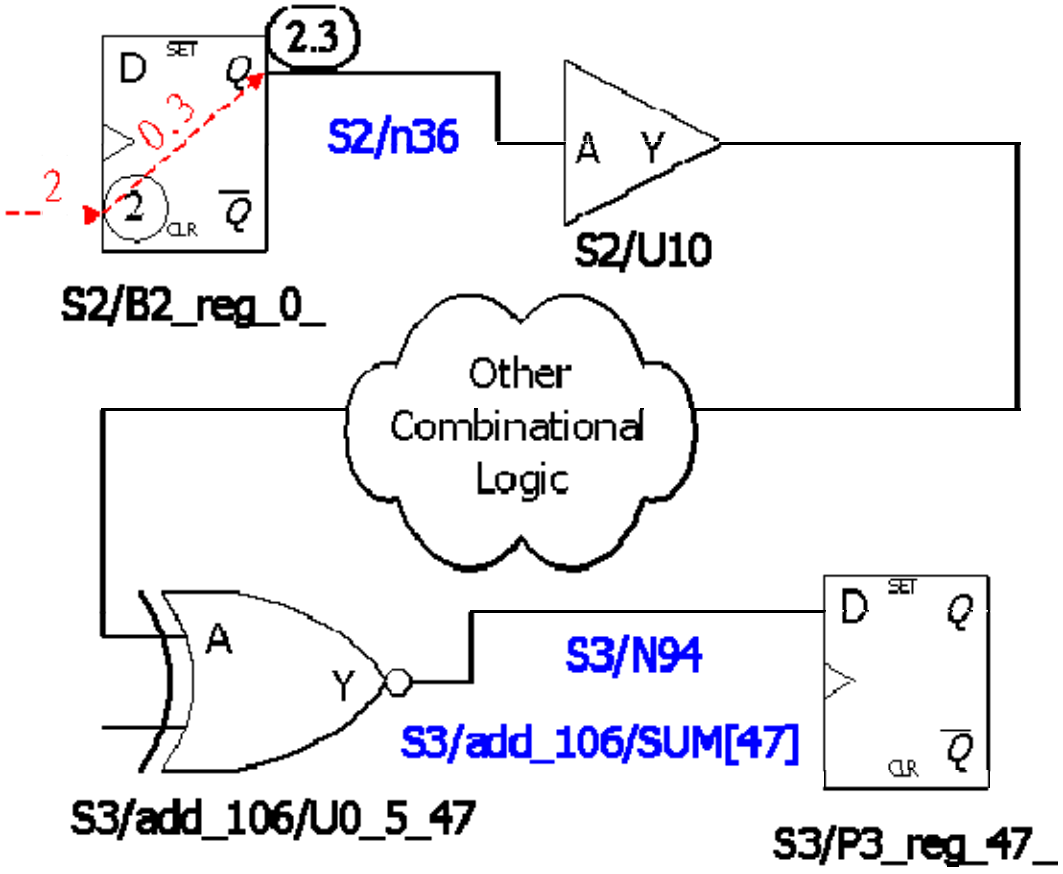
```
   S2/B2_reg_0_/Q (DFFHQX4)                                0.16
0.30       2.30 r
```

接着信号自起点开始向终点传递，这一行表示路径起点的 Flip-Flop 从 CK 端点到 Q 端点的时间延迟为 0.3ns，且在此节点的 Transition Time 为 0.16ns。所以信号到达此节点的时间为 2+0.3=2.3ns（图四）。最右边显示 r 是因为 Q 端点从 0 变化到 1 时的延迟时间比 1 变化到 0 时的延迟时间还长，如果状况相反的话，最右边会标示 f。以上数值是由组件库（Cell Library）里的时序表（Timing Table)查出来的，其计算的方式请参照「静态时序分析(Static Timing Analysis)基础及应用（上）」。

```
   S2/n36 (net)                                      1          0.03
0.00       2.30 r
S2/U10/A (BUFX20)                                  0.16       0.00
```

```
2.30 r
```

這兩行和上一行最右邊的 Path 字段都一样，这是因为其实它们是同一个节点，所以信号到达时间一样。仔细的读者这时候可能会有个疑问，Flip-Flop 的 Q 输出端和后面 Buffer 的输入端 A 信号到达时间应该有一个联机延迟（Interconnect Delay）的差距吧？想法上是没错，但因为 Design Compiler 这个合成器将联机延迟的时间合并到组件延迟（Cell Dealy）的时间内计算，所以从时序报告中看不到延迟时间的信息。也就是说，如果 Point 栏是 net 的话，各位只需去检视 Fanout 和 Cap 字段即可。S2/n36 这个 net 只有推动一个 Buffer，其 Fanout 为 1。负载则是 Buffer 的输入负载和预估联机负载的总和，其值为 0.03pF。



图四

```
   S2/U10/Y (BUFX20)                                        0.23
0.21      2.51 r
```

這一行是描述 Buffer 从输入端到输出端的时间延迟，其值为 0.21，所以信号到达 Buffer 输出端的时间为 2.3+0.21=2.51ns（图五）。

接下来是一堆类似的组件时序信息，我们略过它们不讨论，直接跳到最后面几个组件。

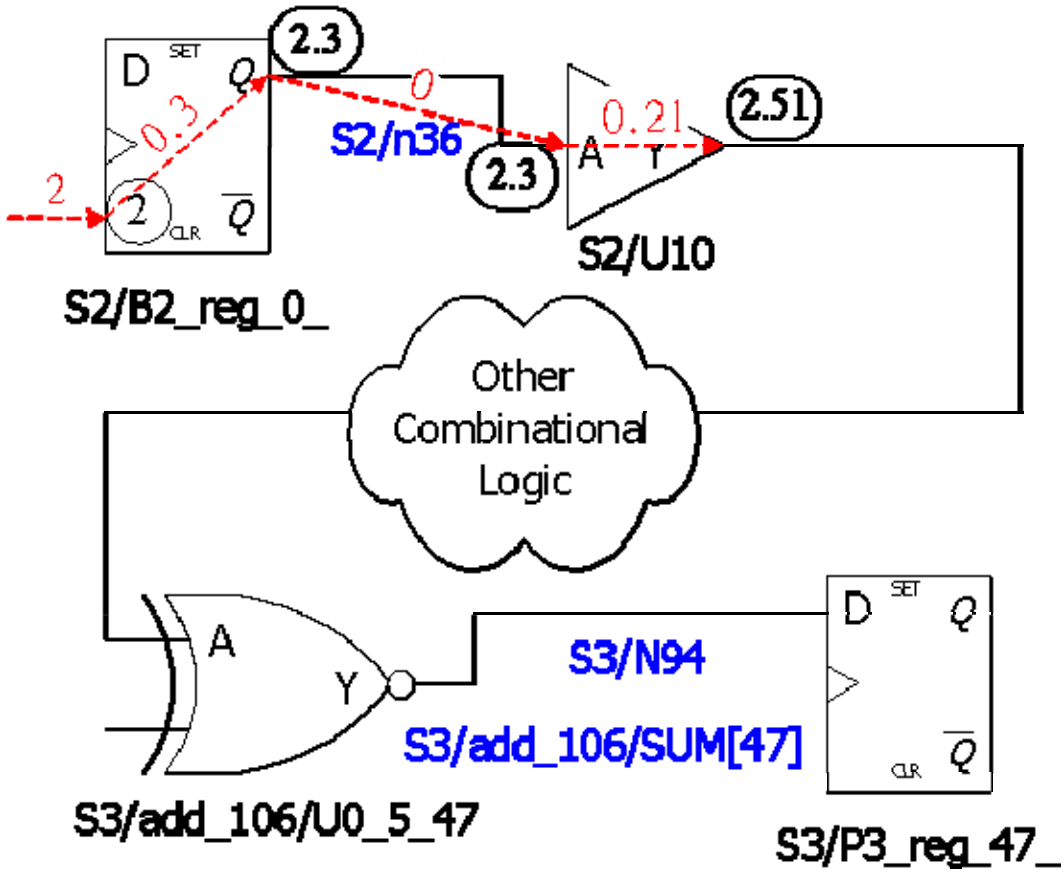
```
   S3/add_106/U0_5_47/A (XNOR2X2)                          0.18
0.00      7.74 f

   S3/add_106/U0_5_47/Y (XNOR2X2)                          0.12
0.22      7.96 f
```

这是到 Critical Path 终点前的最后一个组件，信号到达的时间是 7.96ns。各位可以看到最右边的标示已经变成 f 了，这表示信号由 1 变 0 的状况组件延迟时间较长。

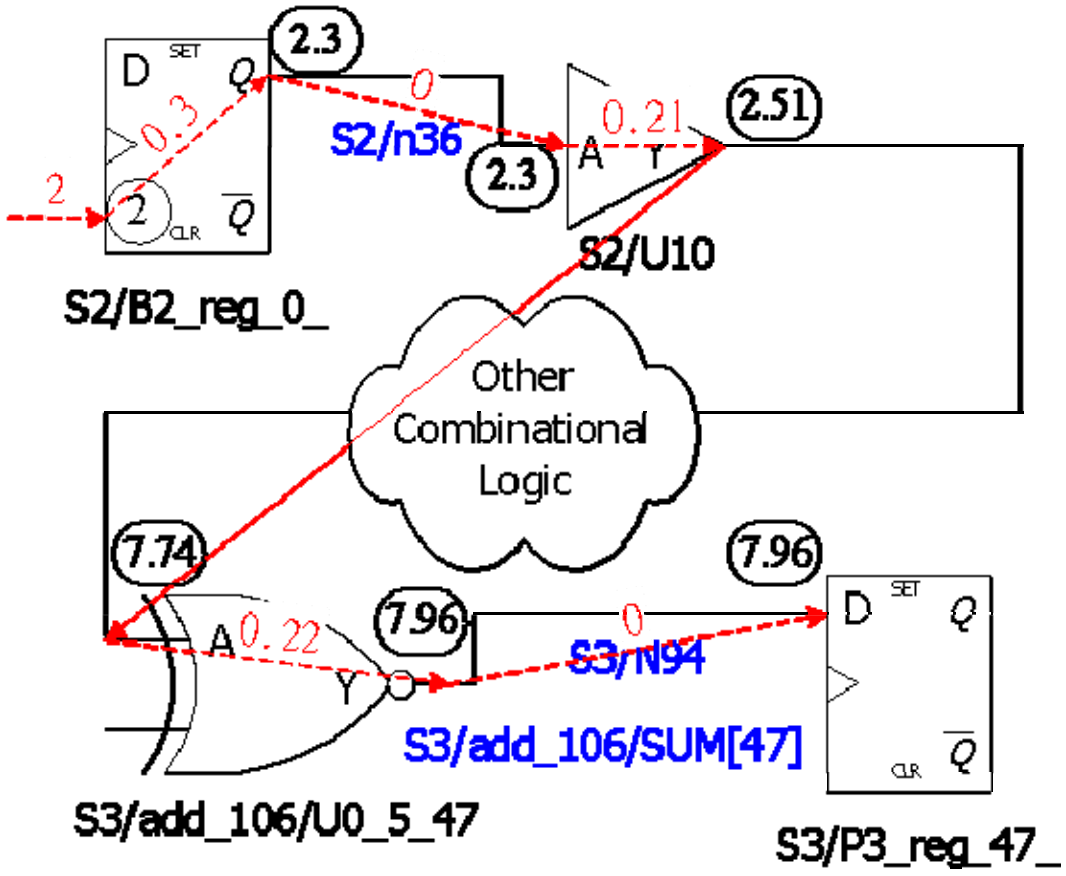
```
   S3/add_106/SUM[47] (net)              1       0.01
0.00      7.96 f

   S3/add_106/SUM[47] (stage3_DW01_add_54_0)
0.00      7.96 f

   S3/N94 (net)                                            0.01
0.00      7.96 f
S3/P3_reg_47_/D (DFFTRXL)                          0.12     0.00
7.96 f
data                          arrival                     time
7.96
```

這幾行都是同一個節點的時序資訊，只是逻辑阶层（Logic Hierarchy）不同。

信号最后到达 Critical Path 终点的时间为 7.96ns(图六)。以上是 Arrival Time（AT）的计算，接下来我们看 Required Time（RT）的计算。


D
SET
Q
0.3
2
2
CLR
Q
2.3
0
S2/n36
2.3
A
0.21
Y
2.51
S2/U10
S2/B2_reg_0_
Other
Combinational
Logic
A
Y
S3/N94
S3/add_106/SUM[47]
S3/add_106/U0_5_47
D
SET
Q
CLR
Q
S3/P3_reg_47_

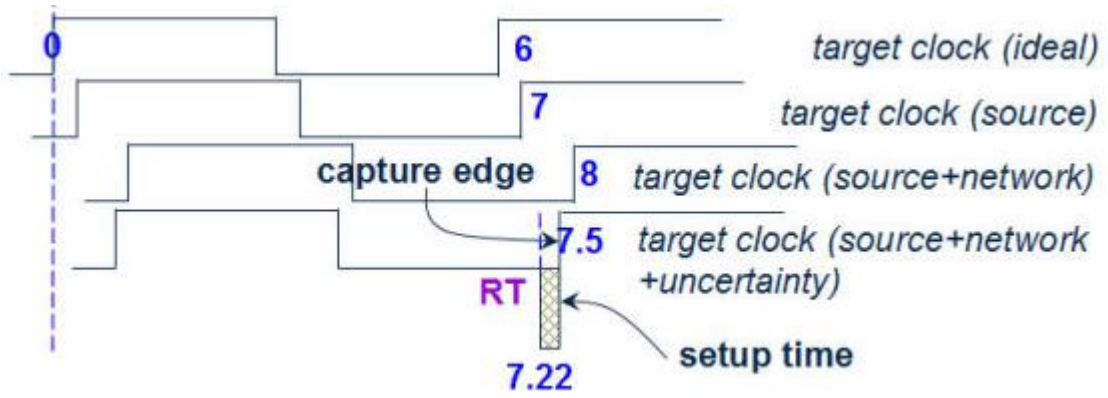
图五



图六

```
   clock MY_CLOCK (rise edge)                                  6.00
6.00
clock network delay (ideal)                                    2.00
8.00
clock uncertainty                                             -0.50
7.50
S3/P3_reg_47_/CK (DFFTRXL)                                     0.00
7.50 r
library setup time                                            -0.28
7.22
data                          required                         time
7.22
```

Critical Path 终点的 Flip-Flop 的频率输入一样有 2ns 的 network delay，所以本来 1 个频率周期后（6ns）要抓取资料就变成了 6+2=8ns 后抓取数据，也就是 Required Time（RT）变成 8ns。但因为我们的频率规格有 0.5ns 的不确定性（clock uncertainty），以最坏状况考虑，频率提早了 0.5ns 到，则 RT 变为 8-0.5=7.5ns。再考虑组件库中定义的 Setup Time 0.28ns，RT 就变为 7.5-0.28=7.22ns（图七）。



图七

```
   data                              required                              time
7.22
data                              arrival                               time
-7.96
-------------------------------------------------------------------------
-----------
slack                                                                   (VIOLATED)
-0.74
```

有了 RT 和 AT 就可以計算 slack，這個例子的 slack 值是負的，也就是無法達到时序规格。由于我们只是要以实例说明 STA，时序规格不符合也无所谓。实际制作芯片时，这当然是不允许的。

布局完成后之时序报告

在布局完成后，组件摆放的位置已经固定，组件间的绕线及其联机延迟（Interconnect Delay）也就可以大致上预估出来。此时估计的联机延迟会比合成时的 Wire Load Model 准确许多。以 Astro 这个布局与绕线软件来说，布局时的绕线预估叫做 Virtual Route（VR）。VR 会假设两个组件间以最短的可行距离去绕线。各位要注意的是「可行」两字，两组件间不见得所有区域都可以拿来做绕线，Astro 的 VR 会自动避开这些区域。布局后的时序报告和合成时的很类似，也会先标示出 Critical Path，然后紧接着是其时序信息。

```
   *   Start point : S3.B3_reg_4_/CK

   *   ( Rising edge-triggered flipflop clocked by MY_CLOCK )

   *   End point   : S4.out_reg_63_/D

   *   ( Rising edge-triggered flipflop clocked by MY_CLOCK at CK )
```

```
   *   Clock Group : MY_CLOCK

   *   Delay Type  : Max

   *   Slack       : -1.0682  (VIOLATED)
```

各位可以看到，此时的 Critical Path 和合成时的不一样。这是因为绕线预估不同，造成联机负载及联机延迟时间的估计值不一样，再加上布局与绕线软件会对时序做最佳化，才会有如此结果。我们先来看看布局后 Critical Path 的时序信息。

```
   Port/Pin
Cap    Fanout Trans.     Incr       Arri       Master/Net

   ------------------------------------------------------------------
---------------

   Rising edge of clock MY_CLOCK                0.0000    0.0000

   Clock Source delay                           1.0000    1.0000

   Clock Network delay                          1.0000    2.0000

   ------------------------------------------------------------------
---------------
S3.B3_reg_4_/CK

   0.0000        0.0000               2.0000 r   DFFTRX4
S3.B3_reg_4_/Q
0.1501 15     0.5663     0.5307 2.5307 r   B3[4]
S4.mult_123.B3_reg_4_ASTipoInst106/A
0.5839     0.0294     2.5602 r   BUFX1
S4.mult_123.B3_reg_4_ASTipoInst106/Y
0.0231 5      0.3842     0.3252     2.8853 r
   S4.mult_123.B3_4_ASThfnNet53
...

   ...

                  S4.add_133.U155/A
               0.3328      0.0006     8.0590 f   XNOR2X2

   S4.add_133.U155/Y

   0.0030 1      0.0894     0.2341     8.2931 f   S4.N109
```

```
   S4.out_reg_63_/D

                    0.0895      0.0001     8.2932 f   DFFTRXL

   ---------------------------------------------------------------
---------------

   Rising edge of clock MY_CLOCK       6.0000     6.0000

   Clock Source delay                  1.0000     7.0000

   Clock Network delay                 1.0000     8.0000

   Clock Skew                       0.5000     7.5000

   Setup time                       0.2749     7.2251

   ---------------------------------------------------------------
---------------

   Required time                                  7.2251

   Arrival time                                   8.2932

   ---------------------------------------------------------------
---------------

   Slack                                         -1.0682  (VIOLATED)
```

这份报告和合成时的报告格式略有差异。合成后报告的 Point 字段此时拆成 Port/Pin 和 Master/Net 字段。我们从最上面一列开始检视此报告，Critical Path 的起点为 S3.B3_reg_4_的 CK 频率输入端点，其信号到达的时间为频率规格所描述的 2ns（Source Latency + Network Latency）。接下来信号走到 S3.B3_reg_4_的数据输出端点 Q，花费了 0.5307ns，也就是在 2.5307ns 时到达此节点。继续往下走，信号会经由绕线接到 S4.mult_123.B3_reg_4_ASTipoInst106（通常这种名字的组件是布局与绕线软件在做时序最佳化时加入的缓冲器或反相器）这个组件的输入端点 A，花费了 0.0294ns，也就是在 2.5602ns 时到达此节点。这段绕线软件会给予一个辨识的名称，就是所谓的 Net Name，各位可以在报告最右边的字段查询到 Net 的信息。在合成软件的报告中，由于 Net 的联机延迟被内涵在组件的延迟中，所以永远为零。在布局与绕线软件的报告中，则会像这样清楚的列出联机的延迟。

报告的后半部和合成时候的报告一样，我们就不加赘述。由于 slack 值为负，在布局之后，时序规格不符合的状况仍旧没有改变。

绕线完成后之时序报告

在绕线完成后，不但组件的位置已经固定，所有的绕线的大小形状也都已经确定。此时的时序报告会更接近实际芯片的时序特性。在理想的状况下，布局后的 VR 预估应该和实际绕线一致。但事情通常没有这么简单，VR 并没有考虑到绕线壅塞的问题，实际绕线时可能在某些区域走了太多的绕线，会导致有些联机没办法沿着 VR 规划的路径走。此时布局与绕线软件会强迫这些联机绕道通行，而这些绕道通行的联机，其联机距离一定会比 VR 规划的路径还要更长。这也意味着会有更多的联机负载，因此造成更多的联机延迟，导致时序上的特性变差。接下来，我们就来看看此范例绕线后的时序报告。

```
   *   Start point : S2.A2_reg_9_/CK

   *   ( Rising edge-triggered flipflop clocked by MY_CLOCK )

   *   End point   : S3.P3_reg_32_/D

   *   ( Rising edge-triggered flipflop clocked by MY_CLOCK at CK )

   *   Clock Group : MY_CLOCK

   *   Delay Type  : Max

   *   Slack      : -0.2721  (VIOLATED)

   ****************************************************************
***************

   Port/Pin           Cap   Fanout    Trans.      Incr      Arri
Master/Net

   ----------------------------------------------------------------
---------------

   Rising edge of clock MY_CLOCK                 0.0000    0.0000

   Clock Source delay                            1.0000    1.0000

   Clock Network delay  (propagated)             0.9998    1.9998

   ----------------------------------------------------------------
---------------
S2.A2_reg_9_/CK                  0.1451    0.0000    1.9998 r
DFFHQX4
```

```
S2.A2_reg_9_/Q     0.0278     1    0.1576    0.3282    2.3280 r
A2[9]

   ...

   S3.P3_reg_32_/D                    0.1001    0.0002    8.0062 f
DFFTRXL

   ---------------------------------------------------------------
----------------

   Rising edge of clock MY_CLOCK                6.0000   6.0000

   Clock Source delay                           1.0000    7.0000

   Clock Network delay  (propagated)            0.9854    7.9854

   Clock Skew                                   0.0000    7.9854

   Setup time                                   0.2513    7.7341

   ---------------------------------------------------------------
----------------

   Required time                                          7.7341

   Arrival time                                           8.0062

   ---------------------------------------------------------------
----------------

   Slack                                                 -0.2721
(VIOLATED)
```

各位可以看到，Critical Path 又换了，原因和布局时的原因类似，一是绕线估算不一致，二是时序最佳化的影响。时序报告的格式和布局后的报告一样，但内容有一重大的差异，就是频率的信息。在布局之前，频率的信息是由频率规格得来的，这只是一个预估值或称目标值。通常我们会在布局后，合成频率树（Clock Tree）来达到频率规格。要完全百分之百达到频率规格是件很不容易的事，所以通常绕线时的频率信息会和频率规格定义的有些许差异，这会影响到静态时序分析的结果。

以上面的例子来说，Critical Path 起点的 Flip-Flop 其 Clock Network Delay(Latency)变为 0.9998ns 而不再是频率规格中的 1ns。而终点的 Flip-Flop 其 Clock Network Delay（Latency）则变为 0.9854ns，原来 0.5ns 的 Clock Skew

则被舍弃不用。如此，计算出来的 slack 值为-0.2721。很不幸，这仍然是一个负值，时序规格仍然无法达成。

比较布局和绕线后的 slack 值，各位会发现布局时的时序特性比绕线后的时序特性优良，这和我们在本小节第一段的结论似乎有所抵触。造成这个现象的原因有二。首先，布局与绕线软件为避免绕线后的时序和布局后的时序差异太大，会在布局时用比较悲观的方式计算时序相关信息。第二个原因则是在绕线时我们又做了很多最佳化的动作以求达到符合时序规格，所以时序特性比较好。不过，这个例子似乎无法用布局绕线软件内建的最佳化功能来达到时序规格。必须再回到之前的合成步骤、硬件描述语言撰写步骤，甚至更早的架构规划步骤来修改。

总结

本文对 STA 在实际 IC 设计流程中的应用举一范例说明，希望各位在具有实际数据的范例导引下，能对 STA 的应用有更深一步的认识。(设计服务组/陈麒旭)