第 3 章
时序分析与约束

在我们学习约束自己的设计之前先来了解时序分析的基础知识。基本上，时序分析有两种：

- 动态时序分析
- 静态时序分析（STA）

动态时序分析是指在输入端输入一组向量，并观察信号到达电路中各个点的时间。通过了解所使用的输入和观察到的信号之间的时间差，我们就知道信号经过特定路径需要多长时间。对于触发器，通过观察输入 D 相对于输入 CLK 的到达时间，我们可以知道特定触发器是否满足建立和保持要求。因此，该过程依赖于时序仿真，也依赖于所施加的激励。但是，静态时序分析通过分析电路拓扑来计算相同的信息，而不需要任何输入向量。

通常，时序分析指的是静态时序分析。在本章中，时序分析也是指静态时序分析，除非另有说明。

3.1 静态时序分析

STA 的概念始于 20 世纪 90 年代中期。从那时起，静态时序分析（STA）已经成为时序分析的首选方法。STA 不依赖于输入向量，STA 的内容包括分析电路拓扑并计算电路中不同信号到达各个点的时间窗口（time window），然后将其与要求信号到达该点的时间进行比较。只要信号到达的时间范围满足要求，从 STA 的角度来看设计就是可用的。STA 在动态时序分析方面变得流行的主要原因是它更为详尽，因为 STA 不依赖于输入向量的完整性。从第 5 章开始，我们将了解 STA 的细节。现在让我们考虑一个非常抽象的电路图，如图 3.1 所示。

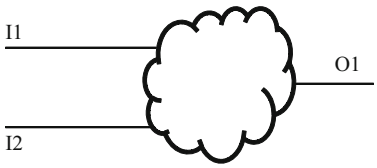

图 3.1　STA 基础知识的示例（抽象）电路

STA 知道什么时候信号可以到达输入 I1 和 I2（以及所有其他输入）。现在，STA 将计算这些信号到达输出 O1（以及所有其他输出）的时间。STA 还知道要求信号到达输出 O1 的时间。通过比较信号到达 O1 的时间和要求时间，STA 工具将报告时序是否满足要求。如果电路中包括一些触发器，那么 STA 也需要计算每个触发器接收其时钟和数据的时间。然后，STA 工具将对每个触发器上数据和时钟的到达时间与该触发器要求建立和保持的时间进行比较。如果数据到达时间不满足触发器建立和保持的时间要求，那么 STA 工具会报告违规。让我们稍微深入了解一下图 3.1 所示的电路。假定图 3.2 是上述电路的一小部分。

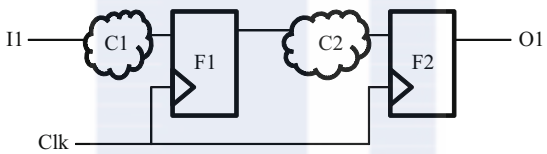

图 3.2　图 3.1 的内部电路

对于这部分电路，在 F1 时钟的每个有效边沿，其数据应该在 F1 建立窗口开始之前到达。另一方面，只有在可靠捕获前一个数据之后，数据才能到达（也就是说，不能违反保持时间）。这取决于触发边沿到达 F1 的时间、组合电路 C1 的延迟，以及输入 I1 的到达时间。

类似地，对于触发器 F2，时序要求将取决于触发边沿到达 F2 时钟引脚的时间、组合电路 C2 的延迟，以及数据从 F1 开始的时间，而这反过来又依赖于 F1 时钟引脚的触发边沿。

在 3.4 节，我们将看到 STA 如何依赖于延迟计算。由于 STA 如此严重地依赖延迟计算，所以只能在得到门级网表后执行它。RTL 可以没有 STA。即使对于门级，随着设计流程的进一步发展，对路由延迟和容性负载的估计变得更加精确了，从而使延迟计算得更准确，STA 也更准确。

3.2　时序约束在 STA 中的作用

STA 工具从相应的设计描述中获取电路描述，HDL 是最常用的形式。它还接受库输入——主要用来了解所依赖技术的特性，如通过特定门的延迟值。

STA 工具需要的另一组输入与输入端各种信号的到达时间和其他特性以及各种输出的时间要求有关。这些输入通过时序约束来提供。时序约束在 STA 期间扮演多个角色。

3.2.1 约束作为声明

某些约束只是工具的声明。该工具不需要验证这些声明的准确性或正确性。该工具仅将其用作自身的输入，以便验证时序是否满足。通常，这是设计外部某些情况的信息——工具无法自行确定。

对于图 3.3 所示的电路，以下是只用作声明的约束的示例：

- 输入 I1 到达边界的时间
- 输入 I2 到达边界的时间
- 输入 I1 和 I2 的过渡时间（transition time，信号的状态从逻辑 0 变为逻辑 1 或从逻辑 1 变为逻辑 0 所需时间）
- 输出 O1 必须驱动的负载

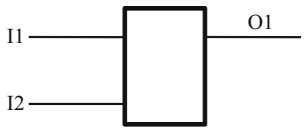

图 3.3 简单的声明类约束

3.2.2 约束作为断言

另一方面，有一些约束充当断言。该工具需要验证设计符合这些约束，这正是 STA 所关心的。在进行各种计算之后，如果工具发现时序满足这些约束，则认为 STA 通过，或认为时序满足要求。相反，如果这些约束不满足要求，则认为 STA 或时序失败。

看一下图 3.3 所示的相同示例电路，断言类约束的一个例子如：

- 输出 O1 在边界处可用的时间

3.2.3 约束作为指令

在其他时候，约束可作为某些工具的指令。这种情况适用于实现工具，如综合或布局布线。这些实现工具将这些约束作为它们试图满足的目标。考虑图 3.4 所示的电路。

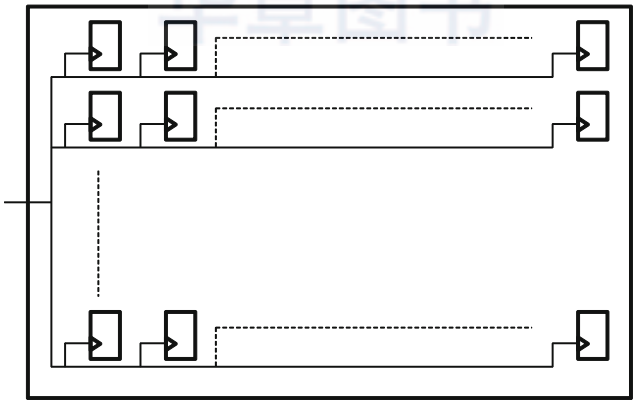

图 3.4 时钟树

对于示例电路，时钟树综合工具现在将提供一个精巧的网络结构，以便 1000 个触发器

中的每一个都可以由该网络驱动。这种时钟树综合工具的示例指令可以实现时钟网络的延迟。时钟树综合工具应该以通过网络的延迟满足规定值的方式来实现树形网络。

3.2.4　约束作为异常

有一些约束做相反的事情，即它们指定约束放宽的范围。假设一些路径被约束以满足某些时序，但是由于某些特殊原因，这些路径并不需要满足这些时序。或者说，即使要求更宽松，它们也可以很好地工作。这些称为时序异常。第 11 章和第 12 章给出了几个时序异常的例子。

3.2.5　约束的角色变化

有时，同一个约束的角色也会发生变化。在设计的一个阶段，它可能是一个声明，而在另一个阶段，它可能变成了一条指令。

再次考虑图 3.4 所示的电路。有一个约束指定了通过该时钟网络的延迟。如果有人想在该时钟网络综合之前进行 STA，那么这个约束（即通过该时钟网络的延迟）只是一个声明。STA 工具应该仅假设通过该时钟网络的延迟是一个指定值。

现在，在时钟树综合期间，同一个约束变成了一个指令。时钟树综合工具需要综合时钟网络，使通过时钟网络的延迟等于指定值。

一旦完成了时钟树综合，STA 就可以通过时钟树网络计算实际延迟。同样的约束就会变得毫无意义了，应该舍弃掉。

因此，相同的约束（即通过时钟树的延迟）：

- 对预布局 STA 而言是一个声明——此处将其视为通过该网络的延迟。
- 对时钟树综合工具而言是一条指令——它尝试实现该时钟网络，使其延迟在指定范围内。
- 对后布局 STA 而言是无意义的。

可以通过图 3.5 来考虑另一个例子。

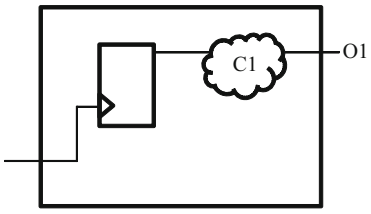

图 3.5　输出所需的时间约束

对于给定电路，将会有一个约束指定输出要求的时间。对于诸如综合的实现工具，该约束作为实现组合电路单元 C1 的指令。C1 应以这样的方式实现，即经过它之后的延迟仍

然允许信号在要求的时间内到达输出 O1。

另一方面，对于 STA 工具而言，该约束可用作断言。STA 工具需要检查在要求的时间内 O1 上的输出是否可用。如果在要求的时间内输出不可用，则应报告 STA 或时序失败。

因此，时序约束可以是输入声明、断言、指令，甚至也可以是约束放宽（称为时序异常）。在大多数情况下，约束本身并不指明其所起的作用。它的作用必须根据上下文来确定。

3.3 STA 中的常见问题

STA 的最大问题是安全感。很多芯片的 STA 干净但操作不正确。一个干净的 STA 使设计人员变得太自信了。

3.3.1 无功能检查

STA 只是时序分析，不包括（或验证）任何其他问题。它根本不进行功能检查。干净的 STA 也不能保证电路能提供所需的功能。干净的 STA 只意味着电路将以指定的频率工作。需要使用仿真、基于检查的断言、FPGA 原型等一些技术来确保功能的正确性。

3.3.2 无声明检查

如 3.2 节所述，有一些约束用于指定设计者的意图，所以被简单地视为声明。STA 工具不会质疑这些代表设计师意图的声明。STA 工具简单地假设这些约束是真实的。在大多数情况下，它们甚至可能没有必要的信息来验证这些约束。如果用户在给出这样的约束时发生错误，那么该工具可能会给出错误的结果。由于用户错误地给出了不正确的约束，所以不满足时序要求的设计可能会使 STA 表面看起来是干净的。即使有些部分的电路其时序特性可能已经确定，但通常用户提供的约束会覆盖隐含推断出的约束。例如，发生的时钟（如时钟分频器）可能会确定时钟的特性。但是，用户指定的特性会将其覆盖，即使用户指定的特性是不正确的。这种情况的细节将在第 6 章进行解释。

3.3.3 要求正确

约束必须处理好平衡，图 3.6 解释了这种平衡。

横轴表示操作频率所需的约束，纵轴表示实际应用于设计中的约束。离原点越远表示约束越严格。45° 的直线表示理想约束。实际定义的约束应与实际所需的相同。

线下面的点代表所应用的约束比要求的更宽松，这意味着 STA 看起来可能是干净的，但是它已经针对比实际要求更宽松的条件进行了处理，因此最终设备可能无法以所需的频率进行工作。

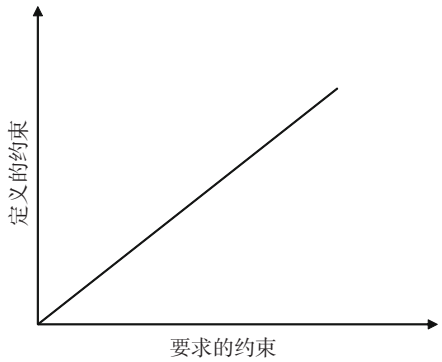

图 3.6　约束精度的要求

一个明显的解决方案是约束的应用应该在始终位于线上或线的上方，这将确保约束比真正需要的更严格。在这里一个干净的 STA 将确保设计的时序是真正干净的，但是，应用更严格的约束有自身的一系列问题。

- 为了满足更严格的时序，这些工具可能要插入驱动能力更高的单元。这些驱动能力更高的单元意味着更高的面积和功率。
- 有时为了满足更严格的约束，可能将更高效的路由资源分配给了这些路径，从而为实际的关键路径留出了较少的高效资源。因此，真正需要更好资源的路径可能会获得较低的优先级，因为另一条路径被给予了不必要的但更严格的约束。
- 严格约束最严重的弊端是时序关闭问题。设计可能不符合时序要求，因为它已经被指定给了比真正需要更严格的约束。试图满足甚至不需要的时序，这可能会带来大量不必要的时间开销。

因此，约束的理想使用应该正好在线上，既不过严也不过松。在拿不准的情况下，必须严一点。

3.3.4　约束中的常见错误

以下是用户在编写约束时可能会犯的一些最常见的错误。

- 不正确的时序异常：时序异常是最容易被误解的约束形式。通过指定这些异常，有时用户放宽了路径约束，但却不应该这样做。
- 不正确的时钟：这个错误可能存在于时钟周期或波形中。但是，更常见的问题通常与时钟产生有关。大部分用户仅将注意力集中在 *divide_by* 和 *multiply_by* 系数上，但是如果没有仔细指定，所产生的时钟波形的正负边沿可能会弄反。
- 与 RTL 的变化不同步：当设计多次重构或重复使用时，RTL 得到了更新。但是，相应的约束不会更新。这可能是因为缺乏知识 / 意识。但是，通常情况下，这是因为用户不容易看到 RTL 的更改与约束中需要的更改的相关性。

在随后的章节中，我们将看到各种开关和选项对许多常用约束的影响。作为必然结果，这些也都指出了用户在编写约束时可能会犯的错误。

3.3.5 好约束的特征

对于任何给定的时序要求，可能有许多种应用约束的方法。一套好的约束应该满足以下条件：

- 首先，这些约束必须明显正确！ 3.3.3 节已经解释了不正确约束的含义。
- 第二个最重要的特征是通过观察约束，应该清楚地理解意图。这有助于检查和找出错误，并且还能使约束不受设计中细微变化的影响。例如，对于图 3.7 所示的电路，一个用户可能会在 Reg1 和 Reg2 之间声明一条虚假路径。另一个用户可能会在 Clock1 和 Clock2 之间声明一条虚假路径。而其他的用户也可能会在异步组中声明两个时钟 Clock1 和 Clock2。尽管从 STA 的角度来看，三者都具有相同的影响，但是从理解意图的角度来看，第三个（异步时钟组）是准确传达约束原因的最好方法（实际上是异常），而第一个（两个寄存器之间的虚假路径）是最差的，它并没有强调虚假路径是由于时钟是异步而引起的，并且它们与两个寄存器之间的实际路径没有任何关系。

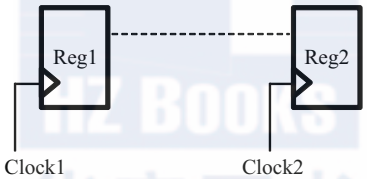

图 3.7 约束的意图

- 对于异常情况，写约束时要特别小心，从而使 RTL 可能出现的细微变化不会导致异常失效。看图 3.8，Clock1 和 Clock2 之间的互斥关系似乎是合理的。但是，如果 RTL 被修改，使两个时钟中的一个在多路复用之前被使用，那么上述的异常就会失效，而且极有可能不被更新。

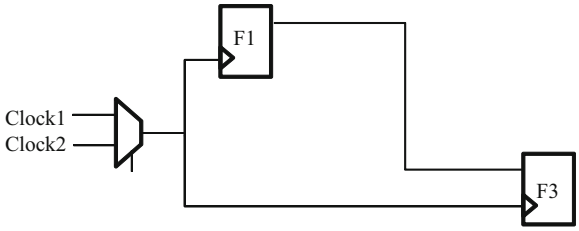

图 3.8 RTL 变化导致的异常失效

- 约束应以能最容易迁移到新技术中的方式来编写。例如，不要根据特定的过渡值规定输入过渡，最好是规定驱动单元。随着技术的变化，将要考虑驱动单元新的驱动能力。
- 最后，约束应该以简明扼要的方式来编写。这对可读性和可重用性有帮助，在大部分工具中，编写简明的约束占用更少的内存（大多数基于 SDC 的工具支持通配符。然而，并不认为通配符的广泛使用是高效的，即使约束变得简明了，这是因为通配符也可以匹配额外的对象。）

编写满足上述特点的约束会更少出错。

3.4　延迟计算与 STA

延迟计算和 STA 是两个截然不同的事情。STA 严重依赖于延迟计算，延迟计算也有除了 STA 以外的其他应用。延迟计算（对于给定的工艺、温度和电压条件）执行以下活动：

- 计算通过特定的门、网络、网段、路径等的延迟。
- 计算特定门输出的压摆率（slew）。
- 当信号通过导线（它反过来又变为下一个门输入的压摆率）时，计算压摆率退化（slew degradation）。

之后，延迟值可能被 STA 工具直接用于分析中。在这种情况下，延迟计算器是 STA 工具本身的一部分。或者，可以将各种延迟值写进 SDF(standard delay format，标准延迟格式）文件中。该 SDF 文件会被读回到 STA 工具中。在这种情况下，延迟计算和 STA 由两种不同的工具完成。

STA 工具的功能可以以如下简化的方式来总结：

1. 收集所有声明类型的约束，如什么时候输入可用。
2. 将一些声明类型的约束传给延迟计算器，如输入过渡时间。
3. 查看电路拓扑以识别各种时序路径。
4. 从延迟计算器中获取路径延迟。
5. 结合 1 和 4 来计算信号到达期望点的时间。
6. 将信号到达时间与断言型约束进行比较，该约束可以告知信号何时可用。
7. 根据实际到达时间和要求时间提供结果。

显然，在很大程度上 STA 的正确性取决于潜在的延迟计算。

3.5　时序路径

时序路径是指信号可以继续穿过，不必等待其他触发条件的路径。沿着时序路径，信

号仅在通过电路元器件时有延迟。

3.5.1 起点和终点

信号时序开始的点称为起点。因此，对于给定电路，所有输入都可以作为起点。信号必须被定时的点称为终点。因此，所有输出都可以作为终点。

在寄存器中，输入 D 必须等待时钟触发的到达。所以，直到 D 到达才能完成的过渡过程现在必须等待。现在进一步传递信号的时序将取决于时钟的到达时间。此处 D 应检查是否满足建立和保持时间要求。因此，时序路径在此结束。所以寄存器也可以作为终点。

类似地，寄存器也可作为起点。信号将从寄存器的 Q 引脚开始，然后向前传递。严格来说，时序从时钟源开始，然后到达触发器的 clock 引脚，接着进入触发器的 Q 引脚，然后继续向前。所以，严格来说，寄存器并不是真正的起点。但是，在大多数实际使用中，把寄存器称为起点。在实际分析中，虽然对于图 3.9 所示的电路，路径跟踪是从时钟源开始的。

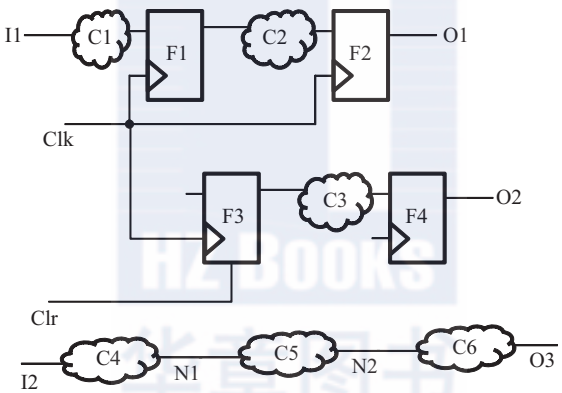

图 3.9 时序路径

起点是：

- 原始输入（I1、Clk、Clr、I2）
- 寄存器（F1、F2、F3、F4）——实际上是这些寄存器的时钟源。

终点是：

- 原始输出（O1、O2、O3）——此处必须检查在要求的时间内信号在这些点可用。
- 寄存器（F1、F2、F3、F4）——此处必须检查 *D*（和其他同步输入）输入满足建立和保持时间的要求。

让我们更仔细地看一下触发器 F3。到达触发器的异步 clear 引脚的信号在此处不需要等待任何其他的触发，它可以简单地通过触发器的 Q 引脚继续向前。因此，时序路径不需要在这里结束。因此，当触发器的 D 引脚作为终点时，异步 clear 或 set 引脚可能不是终点。

3.5.2　打断路径

用户可以通过指定一些启动条件或检查条件在设计的任何位置插入额外的起点和终点。通过指定起点或检查点，路径在该点被打断。

例如，在图 3.9 中，假设一个用户已经指定了从 N1 到 N2 的最大延迟为某个值。在这种情况下，必须在 N2 处进行检查。因此，N2 成为了一个额外的终点，而且它也成为了下一段路径，即 N2 到 O3 的起点。类似地，由于信号跟踪从 N1 开始，所以它变成了起点，而且它也成为前一段路径 I2 到 N1 的终点。

现在，如果用户已经指定 I2 到 N1 的延迟，则 N1 将成为终点。它也将成为下一段路径的起点。但是，I2 只是一个起点，它不会变为终点，因为在它之前没有路径。

再来看触发器 F3，时序路径为 Clr → F3 的 clear 引脚→ F3 的 Q 引脚→ C3 → F4 的 D 引脚。但是，如果在 F3 的 clear 引脚上应用恢复 – 撤销检查，则路径将在该点断开。

3.5.3　功能路径与时序路径

从起点直到终点的过渡过程的路径拓扑称为时序路径，它与功能路径不同。功能路径是指信号流动的拓扑。

例如，对于图 3.9 所示的电路，从 I1 开始的信号将穿过 C1，然后到达 F1 的 D 引脚。之后，当条件正确（即时钟正确触发）时，它将穿过 F1 到达 C2 并一直向前。所以这是一条功能路径。

但是，这不是时序路径。因为，无论信号何时到达 D，我们都不知道它何时会继续通过该触发器。这取决于时钟触发何时到达该触发器。时序路径意味着当信号到达时，我们可以说出路径中下一个过渡过程将会发生的时间。例如，当时钟触发到达时，我们知道下一个过渡过程将发生的触发器输出。

实际的过渡过程可能发生也可能不发生。例如，如果 D 具有相同的值，则 Q 将保持不变。时序路径指示了过渡过程发生的可能性。

因此，时序路径指示过渡过程将会发生的顺序，而不必等待任何其他事件的发生。实际值可能来自其他地方。功能路径表示值将发生变化的顺序，值变化的时序可能仍然受其他事物的控制。有时，时序路径和功能路径可能重叠，如通过组合电路的路径。

3.5.4　时钟路径与数据路径

流入触发器数据引脚的路径称为数据路径。所以数据路径可以是：

- 从一个触发器的输出到另一个触发器的数据输入。
- 从原始端口到触发器的数据引脚。
- 从触发器的一个输出到一个输出端口（流入本设计之外的触发器中）。

流入触发器时钟引脚的路径称为时钟路径。所以时钟路径可以从时钟端口（clock port）

到触发器的时钟端子（clock termina），也可以从时钟分频器或时钟发生器电路的输出到触发器的时钟端子等。

我们使用“触发器”（flop）一词来表示任何同步元素，如内存。

3.6 建立与保持

STA 主要是建立（setup）和保持（hold）分析。在一般术语中，建立是指在时钟边沿到达之前触发器 D（或任何其他同步）引脚上的数据应该稳定的时间。保持是指时钟边沿到达之后数据应保持稳定的时间。

3.6.1 建立分析

在 STA 领域中，建立是指在要求时间之前检查最新数据是否可用。因此，可以在任何终点进行建立检查——而不仅仅是触发器。即使在输出端也要进行建立检查。

建立可以用如下更通用的方式进行定义：数据需要在一些参考事件之前建立并使之可用。对于触发器，参考事件是时钟触发。对于其他终点，参考事件是“期望数据在那一点可用的时间。”

考虑图 3.10 所示的电路。

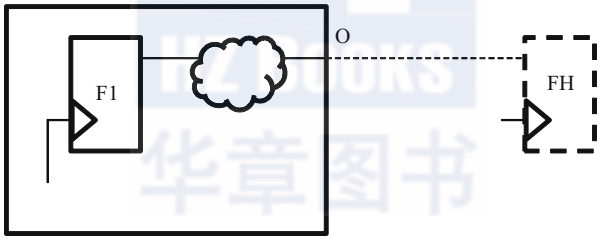

图 3.10 输出的建立检查

对于输出 O，我们假设要求输出在时钟边沿到达后的 6 个时间单位可用。假设时钟周期为 10，这个要求可以看作是假想触发器（FH——虚线所示）的建立要求为 4（10−6），该假想触发器位于输出端口 O 之后。因此，可以在输出端进行建立检查。

3.6.2 保持分析

类似地，保持是指在当前信号的稳定性要求满足之后，检查最新的数据可以干扰当前信号的最早时间。因此，像建立一样，可以在任何终点进行保持检查，包括在输出端——不仅仅是触发器。

保持可以用如下更通用的方式来定义：一些参考事件后，数据需要保持和不干涉。对

于触发器，参考事件是时钟触发。对于其他终点，参考事件是“允许数据在此点变化的时间。”

让我们再次考虑图 3.10 所示的电路。对于输出 O，我们假设输出在时钟边沿到达之后需要保持 2 个时间单位。这个要求可以看作是假想触发器（FH）的保持要求为 2，该假想触发器位于输出端口 O 之后。因此，可以在输出端进行保持检查。

3.6.3　其他分析

建立检查可以确保移动最慢的数据也能及时到达并满足建立的标准。因此，数据路径可以计算最大延迟。因此，它也称为最大分析（max analysis）。由于建立检查考虑了最晚到达的数据，所以也称其为晚期分析（late analysis）。

类似地，保持检查可以确保即使是移动最快的数据也不应该干扰其他数据，同时期望数据保持稳定。因此，数据路径计算最小延迟。因此，它也称为最小分析（min analysis）。由于保持检查考虑了最早到达的数据，所以也称其为早期分析（early analysis）。

STA 的建立 – 保持分析也被称为最小 – 最大分析（min – max analysis）或早期 – 晚期分析（early – late analysis）。更重要的是熟悉这些概念，而不用太担心术语。有时，对同一个概念，不同的工具可能使用不同的术语。有时甚至使用相同的术语来指代不同上下文中的不同概念。

除了建立或保持分析之外，STA 还进行脉冲宽度、恢复、撤销分析等。

3.7　裕度

裕度是指任何超出要求的额外的余地。假设，时刻 6 之前要求信号可用（建立分析），最后一个信号在时刻 4 到达。因此，信号有两个时间单位的额外余量，并且它也不会影响设计的运行。这个 2 被称为建立裕度（setup slack）。

类似地，比如要求信号在时刻 2 之前保持稳定（保持分析）。最早的一个新信号在时刻 5 到达那里。因此，该信号快了 3 个时间单位。这个 3 被称为保持裕度（hold slack）。

建立裕度 = 数据建立要求 – 最后一个到达的信号

保持裕度 = 最早到达的信号 – 数据稳定要求

图 3.11 更好地解释了这一点。

将 E1 作为需要捕获数据的边沿，S 指示在 E1 之前建立要求的时间。所以最迟的数据应该在时刻 S 之前到达终点。我们假设最后一个到来的变化发生在 Ma 时刻，所以建立裕度是 Ma – S 的持续时间（由 S – Ma 测得）。

就像数据必须在 S 之前到达一样，类似地，数据不应该如此早地到达，这样它可能会干扰前一个边沿捕获数据。令 E0 为前一个边沿，假定前一个数据在此处被捕获。为了可靠

地捕获前一个数据，当前数据不应该干扰它，直到满足从 E0 开始的保持要求。H 代表 E0 之后的保持时间，因此当前数据只能在 H 时刻之后到来。我们假定最早一个到来的变化发生在 Mi 时刻，所以保持裕度是 H – Mi 的持续时间（由 Mi – H 测得）。

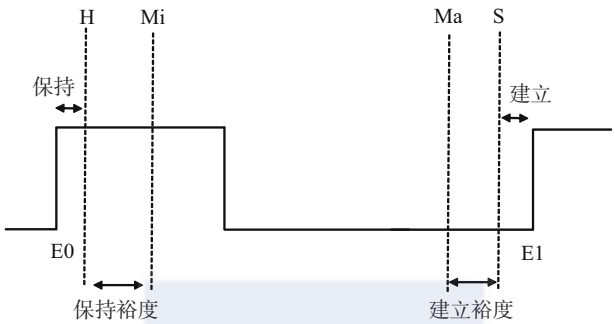

图 3.11　裕度

正裕度意味着时序已经得到满足，而负裕度意味着时序尚未得到满足。根据所用的工具，报告的确切格式将不同。但是，对于任何工具而言，它们都将通过路径跟踪延迟（建立分析的最大值和保持分析的最小值），并跟踪信号要求的时间。然后，它将比较两个数值并提供裕度值（正或负）。路径延迟或要求时间的计算还考虑了时钟路径的延迟，该延迟用于触发起点处的数据启动和终点处的数据捕获。

3.8　片上变化

考虑图 3.12 所示的电路。

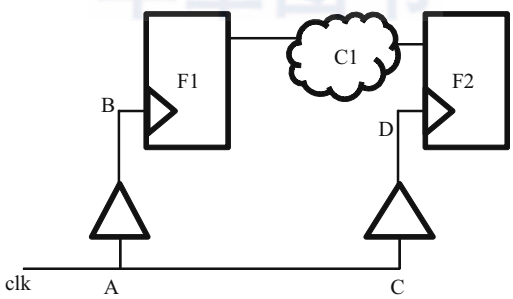

图 3.12　片上变化

我们想在触发器 F2 上进行建立分析，因此必须考虑最慢的数据路径。所以我们会考虑以下的最大延迟：

- 触发器 F1 的 Clk → Q 的延迟

- 组合逻辑 C1
- 互联网络

但是，只有当时钟触发信号到达时，数据才会通过触发器。因此，我们必须用最大延迟考虑到达触发器 F1 的时钟路径。同样的时钟也到达了触发器 F2，并作为建立检查的参考事件。如果我们以最大延迟对待时钟路径，则参考事件也会延迟，这可能会使数据到达的裕度更大。因此，对于触发器 F2 的时钟路径，应考虑最小延迟。

实际上，对于同一个时钟，一段（A → B → F1 的 CLK 引脚）按最慢路径来考虑，而另一段（A → C → D → F2 的时钟引脚）按最快路径来考虑。

对于同一网络不同段间的这种差异处理考虑了同一芯片在不同部分上的任何变化，并称之为片上变化（on-chip variation）。

对于保持分析，数据路径必须是最快的，所以进入 F1 的时钟网络段也将以最快的速度来考虑，而进入 F2 的 CLK 的部分将以最慢的速度来考虑。

这种片上变化减小了裕度。有一段时钟网络被启动触发器和捕获触发器共用（从 Clk 端口到 A）。无论是捕获触发器还是启动触发器，通过此段的时钟将具有相同的延迟。两个触发器共用的这段被认为具有相同的延迟。这样可以防止过度悲观。一些工具将片上变化应用到了整个时钟路径上，包括共用部分。之后，它们应用修正因子来补偿共用段中所考虑的延迟差异。这一点在一个术语中得到了体现，即时钟网络悲观效应降低或时钟树悲观效应降低（clock network pessimism reduction or clock tree pessimism reduction）。

3.9　小结

约束为用户提供了一种方式来指定对 STA 工具以及实现工具的时序目标。STA 用于进行建立和保持分析，并计算正在分析的路径裕度。为了进行这个分析，STA 非常依赖于延迟计算。