**静态时序分析圣经翻译计划——第二章：STA概念 （下）**

[**赵俊军**](https://www.zhihu.com/people/zhao-jun-jun-19)

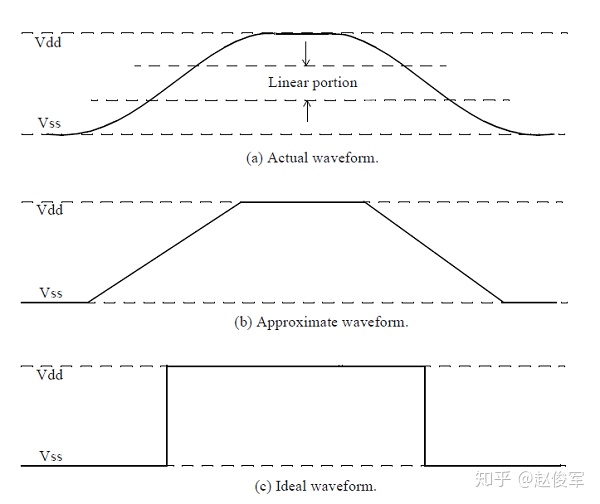
[**​**](https://www.zhihu.com/question/48510028)

上海交通大学 电子与通信工程硕士在读

83 人赞同了该文章

**2.5 波形的压摆**

压摆率（slew rate）的定义是电压转换速率。在静态时序分析中，通常会根据电平转换的快慢来衡量上升波形或下降波形。 压摆（slew）通常是根据转换时间（transition time）来定义的，转换时间是指信号在两个特定电平之间转换所需要的时间。请注意，转换时间实际上就是压摆率的倒数，因此转换时间越大，压摆率就越低，反之亦然。

图2-10

回顾图2-10给出的CMOS单元输出端典型波形：靠近Vdd和Vss两端的波形是渐近的，很难确定过渡时间的确切起点和终点。因此，一般使用指定的阈值电压来规定过渡时间计算的起点和终点。例如，压摆阈值设置可以如下所示：

​ #下降沿的阈值点

​ **slew\_lower\_threshold\_pct\_fall**：30.0；

​ **slew\_upper\_threshold\_pct\_fall**：70.0；

​ #上升沿的阈值点

​ **slew\_lower\_threshold\_pct\_rise**：30.0；

​ **slew\_upper\_threshold\_pct\_rise**：70.0；

以上这些数值的单位同样是Vdd的百分比。阈值设置指定了下降压摆（Fall slew）为下降沿达到Vdd的70％和30％的时间之差。类似地，设置指定了上升压摆（Rise slew）为上升沿达到Vdd的30％和70％的时间之差，如图2-13所示：

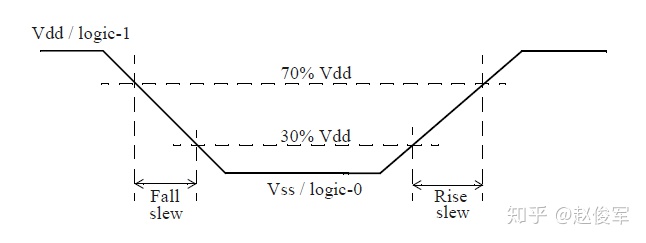
图2-13

图2-14是另一个示例，其中下降沿的压摆测量范围为80％至20％，而上升沿的压摆测量范围为10％至90％，以下是这个例子的阈值设置：

​ #下降沿的阈值点

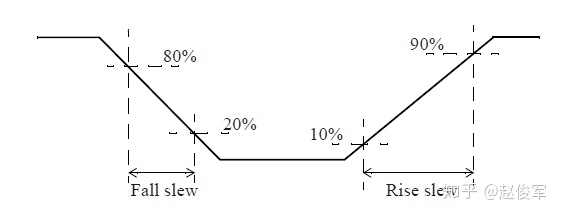
​ **slew\_lower\_threshold\_pct\_fall**：20.0；

​ **slew\_upper\_threshold\_pct\_fall**：80.0；

​ #上升沿的阈值点

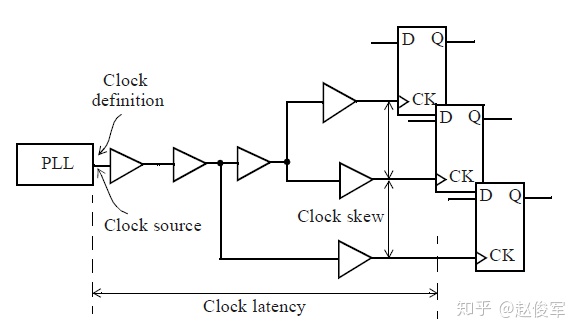
​ **slew\_lower\_threshold\_pct\_rise**：10.0；

​ **slew\_upper\_threshold\_pct\_rise**：90.0；

图2-14

**2.6 信号偏斜**

偏斜（skew）是指两个或多个信号（数据或者时钟）之间的时序之差。例如，如果一个时钟树（clock tree）有500个终点，并且有50ps的偏斜，则意味着最长时钟路径和最短时钟路径之间的延迟差为50ps。如图2-15所示是一个时钟树，时钟树的起点通常是定义时钟的节点，时钟树的终点通常是同步元件（例如触发器）的时钟引脚。时钟延迟（clock latency）是指从时钟源到终点所花费的总时间，时钟偏斜（clock skew）是指到达不同时钟树终点的时间差。

图2-15

理想时钟树是假定时钟源具有无限驱动力，也就是说，时钟可以无延迟地驱动无限个终点。另外，假定时钟树中存在的任何逻辑单元都具有零延迟（zero delay）。 在逻辑设计的早期阶段，STA通常使用理想的时钟树来执行，因此分析的重点是数据路径（data path）。在理想的时钟树中，默认情况下时钟偏斜为0ps，可以使用**set\_clock\_latency**命令显式地指定时钟树的延迟：

**set\_clock\_latency** 2.2 [**get\_clocks** BZCLK]

上述命令规定了时钟树BZCLK的上升沿延迟（rise latency）和下降沿延迟（fall latency）均为2.2ns。注意，如果两个延时值不同，可以使用选项**-rise**和**-fall**来分别指定延时值。

时钟树的时钟偏斜可以借助**set\_clock\_uncertainty**命令显式指定时钟不确定度的值来进行描述：

​**set\_clock\_uncertainty** 0.250 **-setup** [**get\_clocks** BZCLK] ​

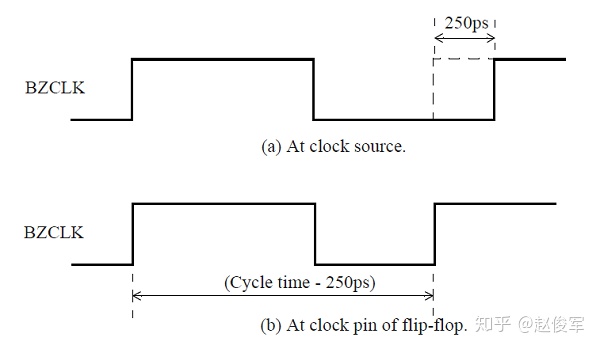
**set\_clock\_uncertainty** 0.100 **-hold** [**get\_clocks** BZCLK]

**set\_clock\_uncertainty**命令为时钟沿的出现指定了一个窗口。时钟边沿时序的不确定性将考虑多个因素，例如时钟周期抖动（jitter）和用于时序验证的额外裕量（slack）。每个实际的时钟源都有一定的抖动量，即一个时间窗口，在该窗口内都可能会出现时钟沿。时钟周期抖动取决于所使用的时钟发生器的类型。实际上是不存在理想时钟的，也就是说，所有时钟都具有一定的抖动量，并且在指定时钟不确定度（clock uncertainty）时应包括时钟周期抖动。

在时钟树被实现（implement）之前，时钟不确定度还必须包括预期的时钟偏斜。

可以为建立时间（setup time）检查和保持时间（hold time）检查指定不同的时钟不确定度。保持时间检查不需要将时钟抖动包括在内，因此通常为保持时间检查指定较小的时钟不确定度。

如下图2-16所示是时钟不确定度为250ps的建立时间检查。图2-16（b）揭示了时钟不确定度是如何从逻辑传播到下一个触发器的耗时中消去的，这相当于要验证设计能够以更高的频率运行。

图2-16

如上所述，**set\_clock\_uncertainty**命令也可以用于建模任何额外时序裕量。例如，设计人员可以在设计过程中依据一定的悲观度而设置50ps的时序裕量，使用**set\_clock\_uncertainty**命令时会将这个值加进去。

总而言之，在实现时钟树之前，**set\_clock\_uncertainty**命令指定的值将包括时钟抖动、时钟偏斜估计值以及额外悲观度。

**set\_clock\_latency** 2.0 [**get\_clocks** USBCLK] ​

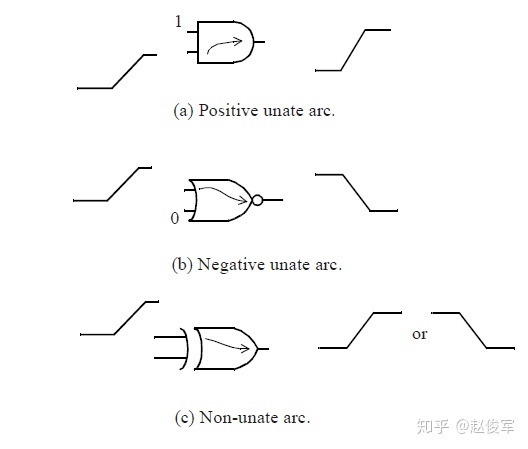
**set\_clock\_uncertainty** 0.2 [**get\_clocks** USBCLK]

上面这条命令中200ps的时钟不确定度可能是由50ps时钟抖动、100ps时钟偏斜以及50ps的额外悲观度组成的。

随后我们将看到**set\_clock\_uncertainty**这条命令是如何影响建立时间和保持时间检查的，我们最好能够将时钟不确定度视为最终计算时序裕量（slack）的补偿（offset）。

**2.7 时序弧**

每个逻辑单元都有多个时序弧（timing arc）。像与门、或门、与非门、加法器这些组合逻辑单元，每个输入引脚到每个输出引脚都存在一条时序弧。而像触发器之类的时序逻辑单元除了有从时钟引脚到输出引脚的时序弧，还有相对于时钟引脚的数据引脚时序约束（timing constraint）。每个时序弧都具有特定的时序敏感（timing sense），即输出如何针对输入的不同跳变类型而变化。如果输入引脚上的上升沿跳变导致输出引脚电平上升（或不变），而输入引脚上的下降沿跳变导致输出引脚电平下降（或不变），则时序弧为正单边（positive unate）类型。 例如,与门和或门的时序弧为正单边类型，如下图2-17（a）所示：

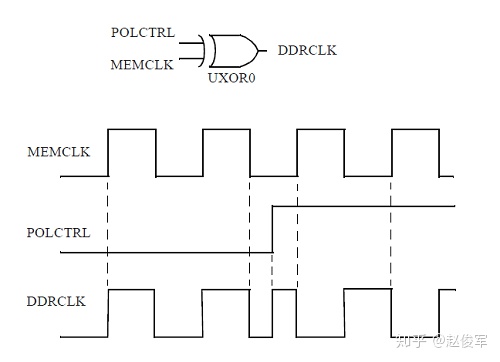
图2-17

如果输入引脚上的上升沿跳变导致输出引脚电平下降（或不变），而输入引脚上的下降沿跳变导致输出引脚电平上升（或不变），则时序弧为负单边（negative unate）类型。例如，与非门和或非门的时序弧为负单边类型，如上图2-17（b）所示。

在非单边（non-unate）时序弧中，仅仅从一个输入引脚的跳变方向是无法确定输出引脚电平将如何跳变的，还要取决于其他输入引脚的状态。例如，异或门中的时序弧是非单边时序弧，如上图2-17（c）所示。

单边性（unateness）对于时序很重要，因为它指定了输入引脚上电平跳变沿将如何通过逻辑单元传播以及将如何出现在逻辑单元的输出引脚上。

可以利用时序弧的非单边性（如异或门）来反转时钟的极性（polarity）。如下图2-18所示，如果输入POLCTRL为逻辑0，则单元UXOR0的输出上的时钟DDRCLK具有与输入时钟MEMCLK相同的极性。如果POLCTRL为逻辑1，则单元UXOR0的输出时钟的极性与输入时钟MEMCLK的极性相反。

图2-18

**2.8 最小与最大时序路径**

逻辑通过逻辑路径传播的总延迟称为路径延迟（path delay），包括了逻辑路径中经过各个逻辑单元（cell）和网络走线（net）的延迟。通常，逻辑想要传递到一个终点可能有不止一条逻辑路径可走，所经过的实际路径取决于逻辑路径上其他输入的状态。图2-19给出了一个例子，由于有多个到达逻辑终点的路径，因此可以得出到达逻辑终点的最大时序和最小时序，对应于最大时序和最小时序的路径分别称为最大路径和最小路径。两个节点之间的最大路径是指延迟最大的路径（也称为最长路径），同样，最小路径是指延迟最小的路径（也称为最短路径）。请注意，最长和最短是指路径上的累积延迟，而不是路径上的逻辑单元个数。

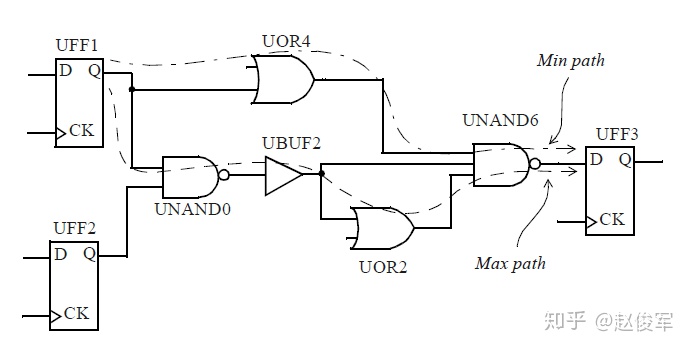
图2-19

图2-19给出了两级触发器之间数据路径的示例。可以看到经过UNAND0、UBUF2、UOR2和UNAND6单元的路径是触发器UFF1和UFF3之间的最大路径，而经过UOR4和UNAND6单元的路径是触发器UFF1和UFF3之间的最小路径。请注意，本例中所说的最小与最大时序路径都是基于终点是触发器UFF3的D引脚。

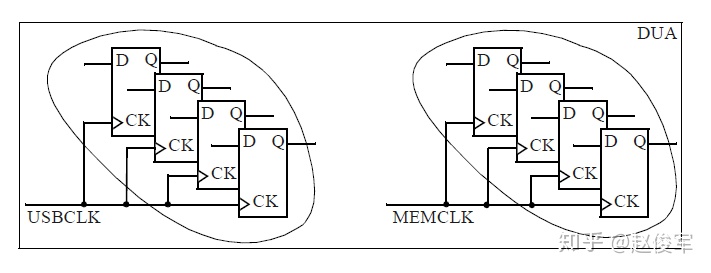
通常也称最大路径为晚路径（late path），称最小路径为早路径（early path）。

当考虑从UFF1到UFF3这样的从触发器到触发器的路径时，其中一个触发器发起（launch）数据，另一个触发器捕获（capture）数据。在这种情况下，由于UFF1发起数据，因此UFF1被称为发起触发器（launch flip-flop），由于UFF3捕获数据，因此UFF3被称为捕获触发器（capture flip-flop）。请注意，“发起”和“触发”不是绝对的，一定是相对于某一条时序路径才能决定触发器到底是发起数据还是捕获数据。例如，UFF3发起的数据如果被下一级触发器捕获了，那么在那条时序路径中UFF3则变为了发起触发器。

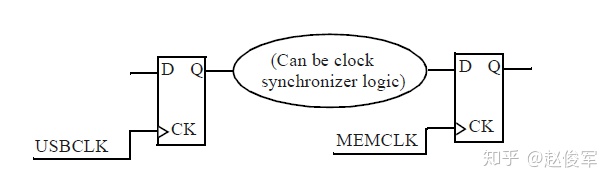
**2.9 时钟域**

在同步逻辑设计中，周期性的时钟信号将计算出的新数据锁存到触发器中。新的输入数据基于的是前一个时钟周期的触发器值，因此锁存到的数据将被用于计算下一个时钟周期的数据。

一个时钟通常驱动许多触发器，由同一时钟驱动的一组触发器称为其时钟域（clock domain）。在典型的设计中，可能有多个时钟域。例如，USBCLK驱动了200个触发器，而时钟MEMCLK驱动了1000个触发器，如图2-20所示。在此示例中，我们称有两个时钟域。

图2-20

需要关注一个问题：两个时钟域是相关的还是彼此独立的？答案取决于是否存在一条从一个时钟域开始并在另一时钟域结束的数据路径，如果没有这样的路径，我们可以肯定地说这两个时钟域彼此独立，这意味着没有时序路径从一个时钟域开始而在另一时钟域结束。

图2-21

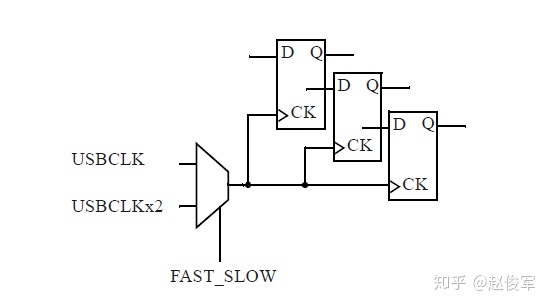
若存在跨时钟域的数据路径（如图2-21所示），则必须确定这些路径是否为真实（real）路径。例如，一个两倍频时钟驱动的触发器发起数据，再由一倍频时钟驱动的触发器捕获数据，这条路径就是一条真实路径。伪路径（false path）的一个例子是设计人员将时钟同步器（clock synchronizer）逻辑明确放置在两个时钟域之间。在这种情况下，即使好像存在从一个时钟域到下一时钟域的时序路径，但这也不是真实的时序路径，因为数据没有被约束要在一个时钟周期之内通过同步器逻辑传播。这样的路径称为伪路径（不是真实的），因为是由时钟同步器来确保数据正确地从一个时钟域传递到另一个时钟域。可以使用**set\_false\_path**命令指定时钟域之间的伪路径，例如：

**set\_false\_path -from** [**get\_clocks** USBCLK] \ ​ **-to** [**get\_clocks** MEMCLK]

虽然无法从图2-21中看到，但实际出现跨时钟域的情况往往是双向的，即从USBCLK时钟域到MEMCLK时钟域，以及从MEMCLK时钟域到USBCLK时钟域，这两种情况都需要在STA中正确理解和处理。

为什么要讨论时钟域之间的路径呢？通常，一个设计中会有多个时钟，并且时钟域之间可能有无数条路径。 分辨出哪些跨时钟域路径是真实的，哪些是伪路径，是时序验证工作的重要组成部分，这使得设计人员可以专注于验证真实的时序路径。

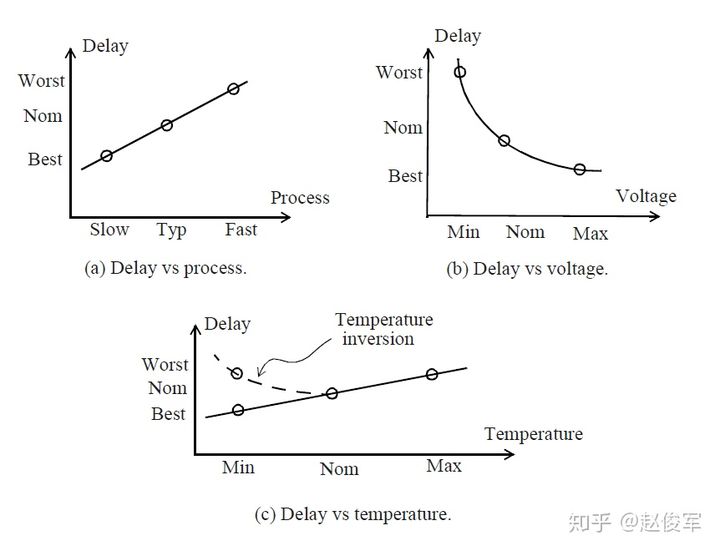
图2-22给出了时钟域的另一个示例，多路复用器（multiplexer）根据设计的工作模式选择时钟源。虽然只有一个时钟域，但却有两个时钟，这两个时钟是互斥的，因为一次只有一个时钟处于有效状态。因此，在这个例子中，USBCLK和USBCLKx2这两个时钟域之间永远不会存在时序路径（假定多路复用器的控制是静态的，并且设计中其余部分也不存在这两个时钟域之间的时序路径）。

图2-22

**2.10 工作条件**

静态时序分析通常是在特定的工作条件（operating condition）下执行的，工作条件定义为工艺（process）、电压（voltage）和温度（temperature）的组合，简称PVT。逻辑单元延迟和互连线的走线延迟是根据特定的工作条件计算的。

半导体代工厂（foundry）为数字设计提供了3种加工工艺模型：慢速（slow）工艺模型，典型（typical）工艺模型和快速（fast）工艺模型，快速和慢速工艺模型代表了半导体代工厂加工的两个极端工艺角（process corner）。对于稳健（robust）的设计，应该在极端加工工艺角以及温度和电源电压都比较极端的环境下对设计进行验证。图2-23（a）展示了逻辑单元延迟如何随工艺角变化；图2-23（b）展示了逻辑单元延迟如何随电源电压变化；图2-23（c）展示了逻辑单元延迟如何随温度变化。因此，为静态时序分析选择特定的工作条件是非常重要的。

图2-23

为STA选择合适的工作条件还需要考虑到可用的单元库（cell library），3种标准的工作条件如下所示：

（1）**WCS**（Worst-Case Slow）：工艺慢（slow）、温度最高（例如125°C）并且电压最低（例如额定1.2V减去10％）。对于使用低电源的纳米技术，可能还有另一个最坏的情况：工艺慢、电压最低并且温度也最低。低温下的延迟并不总是小于高温下的延迟，这是因为对于纳米技术而言，相对于电源的器件阈值电压（Vt）裕度降低了。在这种低电源的情况下，负载较小的逻辑单元的延迟在低温下要高于在高温下的延迟。 对于高Vt（较高阈值，较大延迟）甚至是标准Vt（常规阈值，较低延迟）的单元，情况尤其如此。在较低温度下延迟增加的这种异常行为称为温度反转（temperature inversion），参见图2-23（c）。

（2）**TYP**（Typical）：典型（typical）工艺，温度是额定值（例如25°C），电压是额定值（例如1.2V）。

（3）**BCF**（Best-Case Fast）：工艺快（fast），温度最低（例如-40°C），电压最高（例如额定1.2V加10％）。

功耗分析（power analysis）的工作条件通常不同于静态时序分析所使用的工作条件。 对于功耗分析，工作条件可能是：

（1）**ML**（Maximal Leakage）：工艺快，温度最高（例如125°C），电压也最高（例如1.2V加10％）。该工作条件有最大的漏电功耗（leakage power），对于大多数设计，也有着最大的有效功耗（active power）。

（2）**TL**（Typical Leakage）：经典工艺，温度最高（例如125°C），电压是额定值（例如1.2V）。该工作条件下的漏电功耗比较具有代表性，因为由于正常工作时的功耗，芯片温度往往会更高。

静态时序分析基于的是设计人员所加载（load）和链接（link）的库，可以使用**set\_operating\_conditions**命令明确指定设计的工作条件。

**set\_operating\_conditions** “WCCOM” -library mychip

上述命令使用了在单元库mychip中定义的名为WCCOM的工作条件。

单元库可在各种工作条件下使用，选择何种工作条件进行分析取决于为STA加载的单元库。

第二章：STA概念 （完）