**静态时序分析圣经翻译计划——第七章：配置STA环境 （上）**

**[赵俊军](https://www.zhihu.com/people/zhao-jun-jun-19%22%20%5Ct%20%22_blank)**

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

上海交通大学 电子与通信工程硕士在读

30 人赞同了该文章

本章节介绍了如何为静态时序分析配置环境。

● 正确的约束对于分析STA结果很重要，只有准确指定设计环境，STA分析才能够识别出设计中的所有时序问题。STA的准备工作包括设置时钟、指定IO时序特性以及指定伪路径和多周期路径。在继续学习下一章的时序验证之前，请务必全面了解本章节。

**7.1 什么是STA环境**

大部分数字设计是同步的，从前一个时钟周期计算出的数据在时钟有效沿上被锁存在触发器中。请考虑图7-1所示的典型同步设计，假定待分析设计（DUA）会与其它同步设计交互。这意味着DUA从触发器接收数据，并将数据输出到DUA外部的另一个触发器。

图7-1

为了对这种设计执行STA，需要指定触发器的时钟、以及进入设计和退出设计的所有路径的时序约束。

图7-1中的例子假定只有一个时钟，并且C1、C2、C3、C4和C5代表组合逻辑块，其中C1和C5在待分析设计之外。

在典型的设计中，可能存在多个时钟，且许多路径都会从一个时钟域到另一个时钟域。以下各小节将介绍在这种情况下如何配置环境。

**7.2 指定时钟**

要定义时钟，我们需要提供以下信息：

● 时钟源（Clock source）：它可以是设计的端口，也可以是设计内部单元的引脚（通常是时钟生成逻辑的一部分）。

● 周期（Period）：时钟的周期。

● 占空比（Duty cycle）：高电平持续时间（正相位）和低电平持续时间（负相位）。

● 边沿时间（Edge times）：上升沿和下降沿的时刻。

基本定义如图7-2所示。通过定义时钟，所有内部时序路径（触发器到触发器路径）都将受到约束，这意味着可以仅使用时钟约束来分析所有内部路径。时钟约束指定触发器到触发器的路径必须占用一个周期，稍后我们将介绍如何放宽这一要求（一个周期时间）。

图7-2

以下是一个基本的时钟约束规范：

● **create\_clock** **-name** SYSCLK **-period** 20 **-waveform** {0 5} [**get\_ports** SCLK]

该时钟名为SYSCLK，并在端口SCLK上定义。SYSCLK的周期指定为20个单位，如果未指定，默认时间单位为纳秒（通常，时间单位会在技术库中进行指定）。**waveform**中的第一个自变量指定出现上升沿的时刻，第二个自变量指定出现下降沿的时刻。

**waveform**选项中可以指定任意数量的边沿。但是，所有边沿必须在一个周期内。边沿时刻从零时刻之后的第一个上升沿开始，然后是下降沿，然后再是上升沿，以此类推，这意味着**waveform**列表中的所有时刻值必须单调增加。

● **-waveform** {time\_rise time\_fall time\_rise time\_fall ...}

另外，必须指定偶数个边沿时刻。**waveform**选项将指定一个时钟周期内的波形，然后不断重复。

如果未指定任何**waveform**选项，则默认值为：

● **-waveform** {0，period/2}

以下是一个没有使用**waveform**选项的时钟约束示例（见图7-3）。

● **create\_clock** **-period** 5 [**get\_ports SCAN\_CLK**]

在此约束中，由于未指定**-name**选项，因此时钟的名称与端口的名称相同，即SCAN\_CLK。

图7-3

以下是时钟约束的另一个示例，其中波形的边沿在一个周期的中间位置（见图7-4）。

图7-4

● **create\_clock** **-name** BDYCLK **-period** 15 **-waveform** {5 12} [**get\_ports GBLCLK**]

时钟的名称为BDYCLK，并且在端口GBLCLK上定义。实际上，最好将时钟名称与端口名称保持一致。

以下是另一些时钟约束示例：

图7-5

上图7-5（a）中的时钟约束为：

● **create\_clock** **-period** 10 **-waveform** {5 10} [**get\_ports** FCLK]

上图7-5（b）中的时钟约束为：

● **create\_clock** **-period** 125 **-waveform** {100 150} [**get\_ports** ARMCLK]

图7-6

上图7-6（a）中的时钟约束为：

● **create\_clock** **-period** 1.0 **-waveform** {0.5 1.375} [**get\_ports** MAIN\_CLK]

上图7-6（b）中的时钟约束为：

● **create\_clock** **-period** 1.2 **-waveform** {0.3 0.4 0.8 1.0} [**get\_ports** JTAG\_CLK]

还有一些时钟约束如下：

● **create\_clock** **-period** 1.27 **-waveform** {0 0.635} [**get\_ports** clk\_core]

● **create\_clock** **-name** TEST\_CLK **-period** 17 **-waveform** {0 8.5} **-add** [**get\_ports** ip\_io\_clk[0]]

除了上述属性外，还可以在时钟源处指定过渡时间（压摆）。在某些情况下，例如顶层的输入端口或某些PLL的输出端口，工具无法自动计算出过渡时间。在这种情况下，在时钟源处显式地指定过渡时间很有用，这可以使用**set\_clock\_transition**命令来指定。

● **set\_clock\_transition** **-rise** 0.1 [**get\_clocks** CLK\_CONFIG]

● **set\_clock\_transition** **-fall** 0.12 [**get\_clocks** CLK\_CONFIG]

这个约束仅适用于理想时钟，一旦构建了时钟树就将其忽略，因为此时将会使用时钟引脚上的实际过渡时间。如果在输入端口上定义了时钟，也可以使用**set\_input\_transition**命令（参见7.7节）来约束时钟的压摆。

**7.2.1 时钟不确定度**

可以使用**set\_clock\_uncertainty**约束来指定时钟周期的时序不确定度（uncertainty），该不确定度可用于对可能会减少有效时钟周期的各种因素进行建模。 这些因素可能是时钟抖动（jitter）以及可能需要在时序分析中考虑的任何其它悲观度。

● **set\_clock\_uncertainty** **-setup** 0.2 [**get\_clocks** CLK\_CONFIG]

● **set\_clock\_uncertainty** **-hold** 0.05 [**get\_clocks** CLK\_CONFIG]

注意，建立时间检查的时钟不确定度将减少可用的有效时钟周期，如图7-7所示。对于保持时间检查，时钟不确定度将用作需要满足的额外时序裕量。

图7-7

以下命令可用于指定跨时钟边界路径上的时钟不确定度，称为时钟间不确定度（inter-clock uncertainty）。

● **set\_clock\_uncertainty** **-from** VIRTUAL\_SYS\_CLK **-to** SYSCLK **-hold** 0.05

● **set\_clock\_uncertainty** **-from** VIRTUAL\_SYS\_CLK **-to** SYSCLK **-setup** 0.3

● **set\_clock\_uncertainty** **-from** SYS\_CLK **-to** CFG\_CLK **-hold** 0.05

● **set\_clock\_uncertainty** **-from** SYS\_CLK **-to** CFG\_CLK **-setup** 0.1

图7-8中为两个不同时钟域SYS\_CLK和CFG\_CLK之间的路径。根据上述时钟间不确定度的约束，将100ps用作建立时间检查的不确定度，将50ps用作保持时间检查的不确定度。

图7-8

**7.2.2 时钟延迟**

可以使用**set\_clock\_latency**命令指定时钟的延迟。

● **set\_clock\_latency** 1.8 **-rise** [**get\_clocks** MAIN\_CLK]

● **set\_clock\_latency** 2.1 **-fall** [**all\_clocks**]

时钟延迟有两种类型：网络延迟（network latency）和源延迟（source latency）。网络延迟是指从时钟定义点（create\_clock）到触发器时钟引脚的延迟。源延迟，也称为插入延迟（insertion delay），是指从时钟源到时钟定义点的延迟，源延迟可能代表片上或片外延迟，图7-9展示了这两种情况。触发器时钟引脚上的总时钟延迟是源延迟和网络延迟之和。

图7-9

以下是一些指定源延迟和网络延迟的命令示例：

● **set\_clock\_latency** 0.8 [**get\_clocks** CLK\_CONFIG]

● **set\_clock\_latency** 1.9 **-source** [**get\_clocks** SYS\_CLK]

● **set\_clock\_latency** 0.851 **-source** **-min** [**get\_clocks** CFG\_CLK]

● **set\_clock\_latency** 1.322 **-source** **-max** [**get\_clocks** CFG\_CLK]

源延迟和网络延迟之间的一个重要区别是：一旦为设计建立了时钟树，就可以忽略网络延迟（假设指定了**set\_propagated\_clock**命令）。但是，即使在建立时钟树之后，源延迟也会保留。网络延迟是在进行时钟树综合（Clock Tree Synthesis）之前对时钟树延迟的估计值。在时钟树综合完成后，从时钟源到触发器时钟引脚的总时钟延迟是源延迟加上时钟树从时钟定义点到触发器的实际延迟。

下一节将介绍衍生时钟（generated clocks），7.9节将介绍虚拟时钟（virtual clocks）。

**7.3 衍生时钟**

衍生时钟是从主时钟（master clock）派生而来的时钟，主时钟是指使用**create\_clock**命令定义的时钟。

在基于主时钟的设计中生成一个新时钟时，可以将这个新时钟定义为衍生时钟。例如，如果有一个时钟的三分频电路，则将在该电路的输出处定义一个衍生时钟。由于STA不知道分频逻辑输出的时钟周期已更改，更重要的是新的时钟周期是多少，因此需要定义衍生时钟。图7-10给出了衍生时钟示例，该时钟是主时钟CLKP的2分频。

● **create\_clock** **-name** CLKP 10 [**get\_pins** UPLL0/CLKOUT]

● **create\_generated\_clock** **-name** CLKPDIV2 **-source** UPPL0/CLKOUT **-divide\_by** 2 [**get\_pins** UFF0/Q]

图7-10

可以在触发器的输出端口定义一个新时钟是主时钟，而非衍生时钟吗？答案是肯定的，这确实是可能的，但是它也有一些缺点。定义主时钟而不是衍生时钟会创建一个新的时钟域。通常这不是问题，除了在设置STA约束时需要处理更多的时钟域外。相反，将新时钟定义为衍生时钟不会创建新的时钟域，并且衍生时钟会被认为与其主时钟同相，衍生时钟不需要进行额外的约束。因此，尽量将内部新生成的时钟定义为衍生时钟，而不是将其声明为另一个主时钟。

主时钟和衍生时钟之间的另一个重要区别是时钟源的概念。在主时钟中，时钟源位于主时钟的定义点。而在衍生时钟中，时钟源是主时钟的源而不是衍生时钟的源。这意味着在时钟路径报告中，时钟路径的起点始终是主时钟的定义点。这样一来，与定义新的主时钟相比，衍生时钟具有很大优势，因为对于新的主时钟，是不会自动考虑源延迟的。

图7-11给出了一个在两个输入端都有时钟的多路复用器示例，在这种情况下，不必在多路复用器的输出端定义时钟。如果选择信号设置为常数，则多路复用器的输出会自动获取正确的时钟传播。而如果多路复用器的选择端不受约束，则出于STA的目的，两个时钟都将通过多路复用器传播。在这样的情况下，STA会报告出TCLK和TCLKDIV5之间的路径。注意，这样的路径是不可能存在的，因为选择信号只能选择一个多路复用器的时钟输入。在这种情况下，可能需要设置伪路径或指定这两个时钟之间的互斥（exclusive）关系，以避免报告出错误的路径。当然，这假定设计中其它部分的TCLK和TCLKDIV5之间没有路径。

图7-11

如果多路复用器选择信号不是静态不变的并且在运行期间会发生变化，这样会发生什么呢？ 在这种情况下，会对多路复用器输入端进行时钟门控（clock gating）检查，时钟门控检查将在第10章中介绍，这些检查可确保多路复用器输入端的时钟相对于多路复用器选择信号能够安全地切换。

图7-12给出了一个示例，其中时钟SYS\_CLK由触发器的输出进行门控。由于触发器的输出可能不是恒定的，因此处理这种情况的一种方法是在与门单元的输出处定义一个衍生时钟，该时钟与输入时钟相同。

图7-12

● **create\_clock** 0.1 [**get\_ports** SYS\_CLK]

● **create\_generated\_clock** **-name** CORE\_CLK **-divide\_by** 1 **-source** SYS\_CLK [**get\_pins** UAND1/Z]

下一个示例是一个衍生时钟，其频率高于源时钟的频率。波形如图7-13所示：

图7-13

● **create\_clock** **-period** 10 **-waveform** {0 5} [**get\_ports** PCLK]

● **create\_generated\_clock** **-name** PCLK×2 **-source** [**get\_ports** PCLK] **-multiply\_by** 2 [**get\_pins** UCLKMULTREG/Q]

请注意，在主时钟定义中指定了主时钟周期，然后**-multiply\_by**和**-divide\_by**选项指定了衍生时钟的频率。

**时钟门控单元输出端的主时钟示例**

考虑图7-14中所示的时钟门控示例，两个时钟分别输入进一个与门单元中，问题是与门单元的输出是什么呢？如果与门单元的输入均为时钟，则可以安全地在与门单元的输出端定义一个新的主时钟，因为该单元的输出与任何一个输入时钟都没有相位关系的可能性很小。

图7-14

● **create\_clock** **-name** SYS\_CLK **-period** 4 **-waveform** {0 2} [**get\_pins** UFFSYS/Q]

● **create\_clock** **-name** CORE\_CLK **-period** 12 **-waveform** {0 4} [**get\_pins** UFFCORE/Q]

● **create\_clock** **-name** MAIN\_CLK **-period** 12 **-waveform** {0 2} [**get\_pins** UAND2/Z]

在内部引脚上创建时钟的一个缺点是：它会影响路径延迟计算，并迫使设计人员手动计算源延迟。

**使用Edge和Edge\_shift选项生成时钟**

图7-15给出了一个衍生时钟的示例，除两个不同相的时钟外，还会生成一个二分频时钟。各时钟的波形也显示在图中。

图7-15

下面给出了该示例中所有时钟的定义。衍生时钟的定义使用了**-edges**选项，这是定义衍生时钟的另一种方法。该选项采用源主时钟{上升，下降，上升}的边沿列表，以形成新的衍生时钟。主时钟的第一个上升沿是沿1，第一个下降沿是沿2，下一个上升沿是沿3，依此类推。

● **create\_clock** **-period** 2 [**get\_ports** DCLK]

● **create\_generated\_clock** **-name** DCLKDIV2 **-edge** {2 4 6} **-source** DCLK [**get\_pins** UBUF2/Z]

● **create\_generated\_clock** **-name** PH0CLK **-edges** {3 4 7} **-source** DCLK [**get\_pins** UAND0/Z]

● **create\_generated\_clock** **-name** PH1CLK **-edges** {1 2 5} **-source** DCLK [**get\_pins** UAND1/Z]

如果衍生时钟的第一个边沿是下降沿怎么办？考虑如图7-16所示的衍生时钟G3CLK。可以通过指定边沿5、7和10来定义这种衍生时钟，如以下时钟约束所示。注意，1ns时刻的下降沿将被自动推断出来。

图7-16

● **create\_generated\_clock** **-name** G3CLK **-edge** {5 7 10} **-source** DCLK [**get\_pins** UAND0/Z]

**-edge\_shift**选项可与**-edges**选项一起使用，以指定相应边沿的任何偏移以形成新的衍生波形。它指定边沿列表中每个边沿的偏移量（以时间单位）。以下是使用此选项的示例：

● **create\_clock** **-period** 10 **-waveform** {0 5} [**get\_ports** MIICLK]

● **create\_generated\_clock** **-name** MIICLKDIV2 **-source** MIICLK **-edges** {1 3 5} [**get\_pins** UMIICLKREG/Q]

● **create\_generated\_clock** **-name** MIIDIV2 **-source** MIICLK **-edges** {1 1 5} **-edge\_shift** {0 5 0} [**get\_pins** UMIIDIV/Q]

边沿列表中的边沿序列必须以非降序排列，但是同一边沿可重复使用，以指示不同于源时钟占空比的时钟脉冲。上例中的**-edge\_shift**选项通过将源时钟的边沿1移位0ns获得了衍生时钟的第一个边沿，通过将源时钟的边沿1偏移5ns获得了衍生时钟的第二个边沿，而通过将源时钟的边沿5移位0ns获得了衍生时钟的第三个边沿。下图7-17显示了上述波形：

图7-17

**使用Invert选项生成时钟**

这是衍生时钟的另一个示例，这个示例使用了**-invert**选项：

● **create\_clock** **-period** 10 [**get\_ports** CLK]

● **create\_generated\_clock** **-name** NCLKDIV2 **-divide\_by** 2 **-invert** **-source** CLK [**get\_pins** UINVQ/Z]

在所有其它衍生时钟选项都被使用后，**-invert**选项将会对衍生时钟进行反相。图7-18给出了产生这种反相时钟的原理图：

图7-18

**衍生时钟的时钟延迟**

也可以为衍生时钟指定时钟延迟，在衍生时钟上指定的源延迟还包括了从主时钟定义点到衍生时钟定义点的延迟。因此，由衍生时钟驱动的触发器的时钟引脚的总时钟延迟是主时钟源延迟、衍生时钟源延迟和衍生时钟网络延迟的总和。如下图7-19所示：

图7-19

衍生时钟可以将另一个衍生时钟作为其时钟源，即一个衍生时钟也可以具有衍生时钟，以此类推。但是，衍生时钟只能有一个主时钟。在后面的章节中将介绍更多衍生时钟的示例。

**典型的时钟生成方案**

图7-20给出了在典型ASIC设计中如何进行时钟分配（clock distribution）的情形。晶振（Oscillator）在芯片外部产生低频（典型值为10-50 MHz）时钟，片上PLL将其用作参考时钟，以生成高频低抖动时钟（典型值为200-800 MHz）。然后，该PLL的输出时钟被输入到时钟分频器逻辑中，该逻辑产生ASIC所需的时钟。

图7-20

在时钟分配的某些分支上，可能有时钟门控（clock gates）用于在设计的无效部分关闭时钟，以在必要时节省功耗。在PLL的输出端也可以接一个多路复用器，以便在必要时可以绕过PLL。

在进入设计的芯片输入端口处为参考时钟定义了一个主时钟，在PLL的输出处定义了第二个主时钟。PLL的输出时钟与参考时钟没有任何相位关系。因此，PLL的输出时钟不应是参考时钟的衍生时钟。很有可能的是，由时钟分频器逻辑生成的所有时钟都将被指定为PLL输出处主时钟的衍生时钟。

第七章： 配置STA环境 （未完待续......）