**静态时序分析圣经翻译计划——第七章：配置STA环境 （中）**

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

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

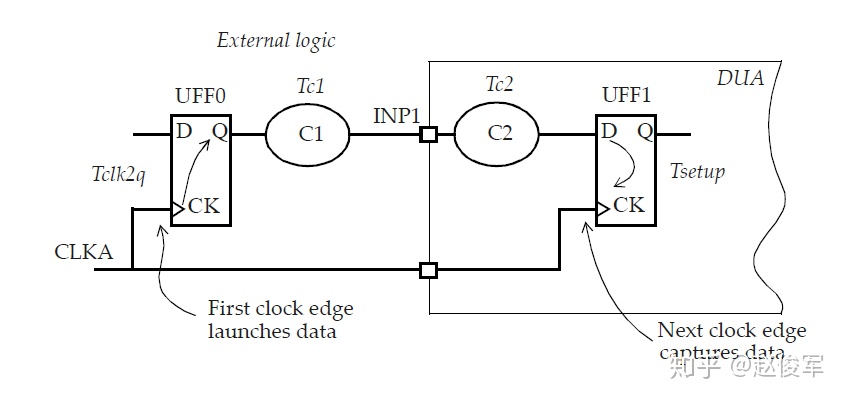
上海交通大学 电子与通信工程硕士在读

23 人赞同了该文章

**7.4 约束输入路径**

本节将介绍输入路径的约束。这里需要注意的一点是，STA无法检查不受约束的路径上的任何时序，因此需要约束所有路径以进行时序分析。在后面的章节中会介绍一些示例，其中一些示例可能并不关心某些逻辑，因而这些输入路径可能可以不用约束。例如，设计人员可能并不在乎一些输入控制信号的时序，因此可能并不需要进行本节中将要介绍的时序检查。但是，本节假定我们要约束全部的输入路径。

图7-21中为待分析设计（DUA）的输入路径。触发器UFF0在设计的外部，并向设计内部的触发器UFF1提供数据。数据通过输入端口INP1连接两个触发器。

图7-21

CLKA的时钟定义指定了时钟周期，这是两个触发器UFF0和UFF1之间可用的总时间。外部逻辑所需的时间为Tclk2q（数据发起触发器UFF0的CK至Q延迟）加上Tc1（通过外部组合逻辑的延迟），因此输入引脚INP1上的延迟定义指定了Tclk2q加上Tc1的外部延迟。并且这个外部延迟是相对于一个时钟指定的，在本示例中为时钟CLKA。

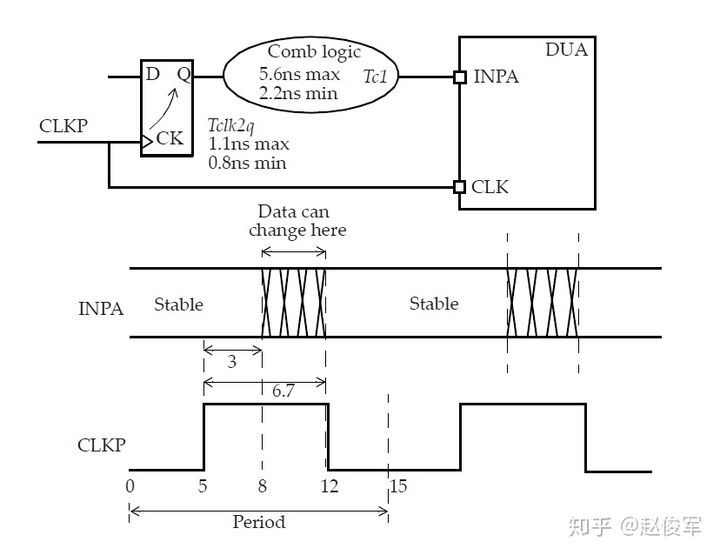
以下是输入延迟的约束：

● **set** Tclk2q 0.9

● **set** Tc1 0.6

● **set\_input\_delay** **-clock** CLKA **-max** [**expr** Tclk2q + Tc1] [**get\_ports** INP1]

该约束指定输入端口INP1的外部延迟为1.5ns，且这是相对于时钟CLKA而言的。假设CLKA的时钟周期为2ns，则INP1引脚的逻辑只有500ps（= 2ns-1.5ns）可以在设计内部中传播。此输入延迟定义意味着输入约束为：Tc2加上触发器UFF1的Tsetup必须小于500ps，才可以确保可靠地捕获到触发器UFF0发起的数据。请注意，上述外部延迟值被指定为了最大值（max）。

图7-22

让我们同时考虑最大和最小延迟情况，如图7-22所示。以下是此示例的约束：

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

● **set\_input\_delay** **-clock** CLKP **-max** 6.7 [**get\_ports** INPA]

● **set\_input\_delay** **-clock** CLKP **-min** 3.0 [**get\_ports** INPA]

INPA的最大和最小延迟是从CLKP到INPA的延迟中得出的，最大和最小延迟分别是最长和最短路径延迟，这些通常也可以对应于最坏情况（worst-case）下的慢速（最大时序工艺角）和最佳情况（best-case）下的快速（最小时序工艺角）。因此，最大延迟对应于最大时序工艺角下的最长路径延迟，最小延迟对应于最小时序工艺角下的最短路径延迟。在我们的示例中，1.1ns和0.8ns是Tck2q的最大和最小延迟值。组合逻辑路径延迟Tc1的最大延迟为5.6ns，最小延迟为2.2ns。INPA上的波形显示了数据到达设计输入端的时间窗口，以及预计达到稳定状态的时间。从CLKP到INPA的最大延迟为6.7ns（= 1.1ns + 5.6ns），最小延迟为3ns（= 0.8ns + 2.2ns），这些延迟是相对于时钟有效沿指定的。在给定外部输入延迟的情况下，设计内部的可用建立时间是慢角（slow corner）下的8.3ns（= 15ns-6.7ns）和快角（fast corner）下的12ns（= 15ns-3.0ns）中的最小值。因此，8.3ns是用来可靠地捕获DUA内部数据的可用时间。

以下是输入约束的更多示例：

● **set\_input\_delay** **-clock** clk\_core 0.5 [**get\_ports** bist\_mode]

● **set\_input\_delay** **-clock** clk\_core 0.5 [**get\_ports** sad\_state]

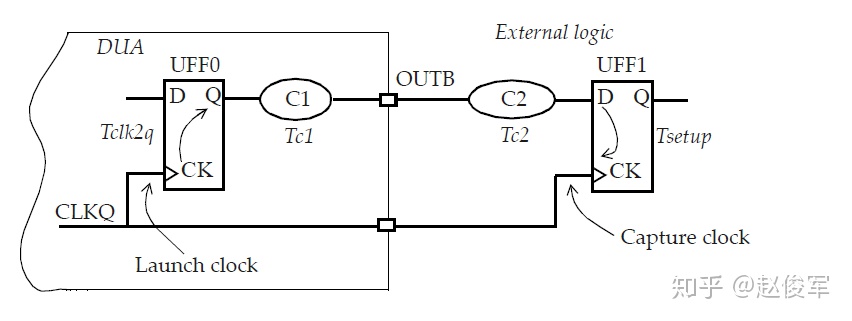
由于未指定**max**或**min**选项，因此500ps这个值将同时用于最大延迟和最小延迟。此外部输入延迟是相对于时钟clk\_core的上升沿指定的（如果输入延迟是相对于时钟的下降沿指定的，则必须使用**-clock\_fall**选项）。

**7.5 约束输出路径**

本节将借助下面的三个例子来介绍输出路径的约束。

**例子A**

图7-23为一条通过待分析设计输出端口的路径示例，其中Tc1和Tc2是通过组合逻辑的延迟。

图7-23

时钟CLKQ的周期定义了触发器UFF0和UFF1之间的总可用时间。外部逻辑的总延迟为Tc2加上Tsetup，此总延迟Tc2 + Tsetup必须作为输出延迟约束的一部分来指定。注意，输出延迟是相对于捕获时钟指定的，数据必须及时到达外部触发器UFF1才能满足其建立时间要求。

● **set** Tc2 3.9

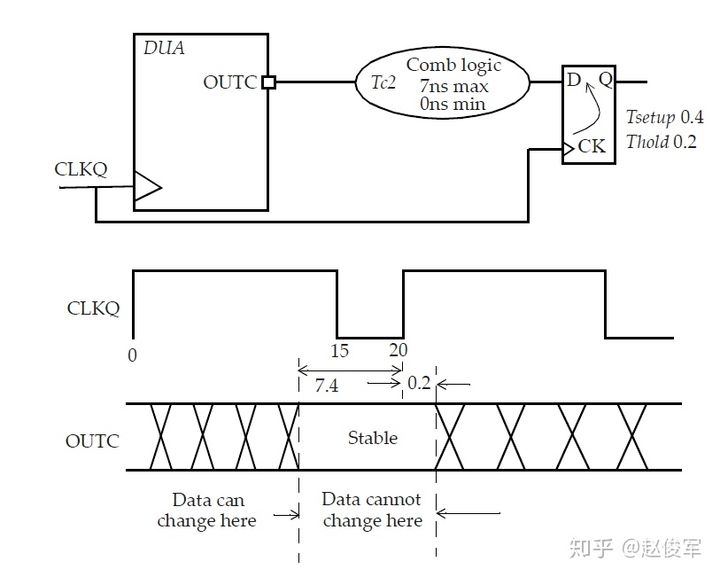
● **set** Tsetup 1.1

● **set\_output\_delay** **-clock** CLKQ **-max** [**expr** Tc2 + Tsetup] [**get\_ports** OUTB]

这指定了相对于时钟边沿的最大外部延迟为Tc2加上Tsetup，即5ns的延迟。最小延迟可以类似地指定。

**例子B**

图7-24给出了同时具有最小和最大延迟的示例。最大路径延迟为7.4ns（=Tc2最大值加上Tsetup = 7 + 0.4），最小路径延迟为-0.2ns（=Tc2最小值减去Thold = 0-0.2）。因此，输出约束为：

图7-24

● **create\_clock** **-period** 20 **-waveform** {0 15} [**get\_ports** CLKQ]

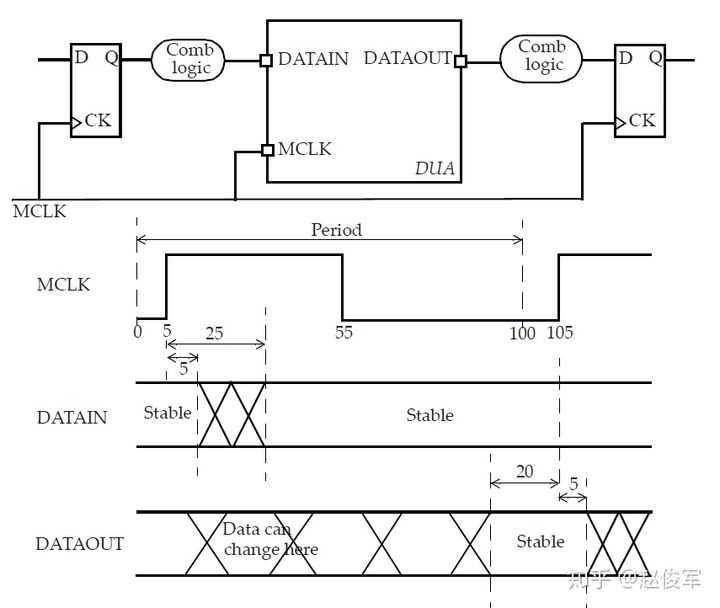
● **set\_output\_delay** **-clock** CLKQ **-min** -0.2 [**get\_ports** OUTC]

● **set\_output\_delay** **-clock** CLKQ **-max** 7.4 [**get\_ports** OUTC]

图7-24中的波形显示了OUTC必须保持稳定状态的时间，以确保外部触发器能够可靠地捕获它。这说明在所需的稳定区域开始之前，数据就必须在输出端口准备就绪，并且必须保持稳定，直到稳定区域结束为止。这同样反映了DUA内部对输出端口OUTC逻辑的时序要求。

**例子C**

这是另一个输入约束和输出约束的示例。该模块具有两个输入端口DATAIN和MCLK，以及一个输出端口DATAOUT。图7-25显示了预期的波形。

图7-25

● **create\_clock** **-period** 100 **-waveform** {5 55} [**get\_ports** MCLK]

● **set\_input\_delay** 25 **-max** **-clock** MCLK [**get\_ports** DATAIN]

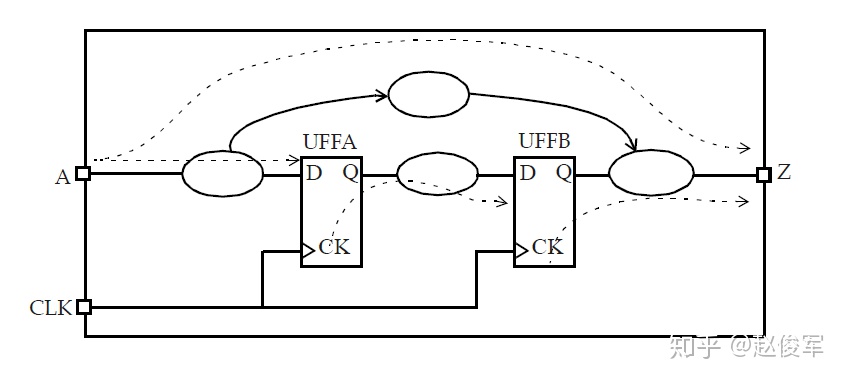
● **set\_input\_delay** 5 **-min** **-clock** MCLK [**get\_ports** DATAIN]

● **set\_output\_delay** 20 **-max** **-clock** MCLK [**get\_ports** DATAOUT]

● **set\_output\_delay** -5 **-min** **-clock** MCLK [**get\_ports** DATAOUT]

**7.6 时序路径组**

设计中的时序路径可以视为路径的集合，每个路径都有一个起点和一个终点。时序路径的示例如下图7-26所示：

图7-26

在STA中，时序路径是根据有效的起点和终点来划分的。有效的起点包括：输入端口或者同步器件（如触发器和存储器）的时钟引脚。有效的终点包括：输出端口或者同步器件的数据输入引脚。因此，有效的时序路径包括：

● 从输入端口到输出端口

● 从输入端口到触发器或存储器的数据输入引脚

● 从一个触发器或存储器的时钟引脚到另一个触发器或存储器的数据输入引脚

● 从一个触发器或存储器的时钟引脚到输出端口

图7-26中的有效时序路径包括：

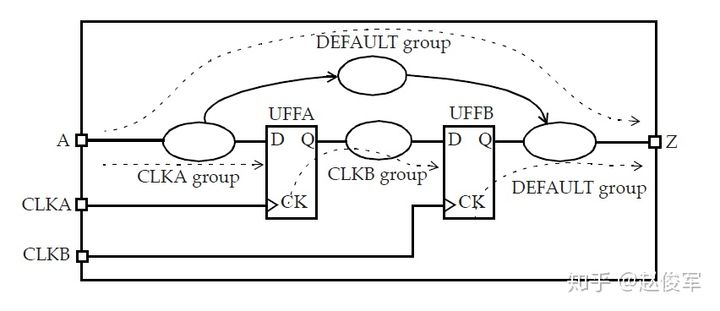
● 输入端口A到输出端口Z

● 输入端口A到触发器UFFA的D引脚

● 触发器UFFA的CK引脚到触发器UFFB的D引脚

● 触发器UFFB的CK引脚到输出端口Z

时序路径可以根据与路径终点相关的时钟分类为不同时序路径组（path groups）。因此，每个时钟都有一组与之相关的时序路径。还有一个默认时序路径组，其中包括了所有非时钟（异步）路径。

图7-27

在图7-27的示例中，时序路径分组为：

● CLKA组：输入端口A到触发器UFFA的D引脚

● CLKB组：触发器UFFA的CK引脚到触发器UFFB的D引脚

● 默认组：输入端口A到输出端口Z、触发器UFFB的CK引脚到输出端口Z

静态时序分析和报告通常分别在每个时序路径组中单独执行。

**7.7 外部属性建模**

尽管**create\_clock**、**set\_input\_delay**和**set\_output\_delay**足以约束设计中用于执行时序分析的所有路径，但这些并不足以获取该模块IO引脚上的准确时序。为了准确地对设计环境进行建模，还需要以下属性。对于输入，需要在输入端口处指定压摆。可以使用以下方式提供此信息：

● **set\_drive**

● **set\_driving\_cell**

● **set\_input\_transition**

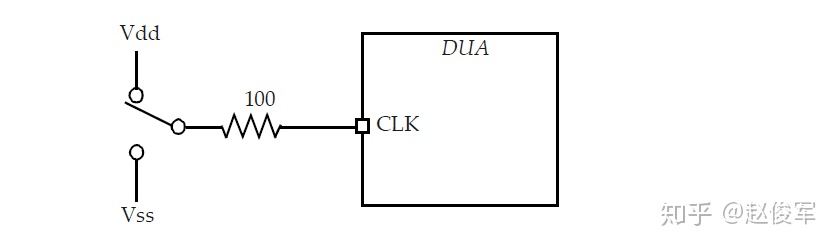
对于输出，需要指定输出引脚的负载电容。可以使用以下命令来指定：

● **set\_load**

**7.7.1 驱动强度建模**

**set\_drive**和**set\_driving\_cell**约束用于对驱动模块输入端口的外部单元的驱动强度进行建模。在没有这些约束的默认情况下，假定所有输入都具有无限的驱动强度，即输入引脚的过渡时间为0。

**set\_drive**明确指定了DUA输入引脚上的驱动电阻值，该电阻值越小，驱动强度越高，电阻值为0表示无限的驱动强度。

图7-28

● **set\_drive** 100 UCLK

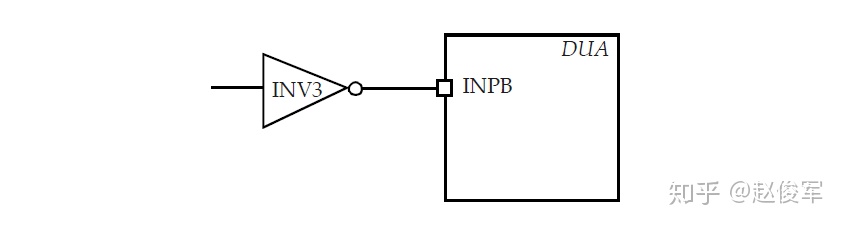
● **set\_drive** **-rise** 3 [**all\_inputs**]

● **set\_drive** **-fall** 2 [**all\_inputs**]

输入端口的驱动强度用于计算第一个单元的过渡时间。指定的驱动强度还可用于计算在任何RC互连情况下从输入端口到第一个单元的延迟值，计算公式如下：

● 延迟值 = （驱动强度 \* 网络负载） + 互连线延迟

**set\_driving\_cell**约束提供了一种更方便、更准确的方法来描述端口的驱动能力。**set\_driving\_cell**可用于指定驱动输入端口的单元类型。

图7-29

● **set\_driving\_cell** **-lib\_cell** INV3 **-library** slow [**get\_ports** INPB]

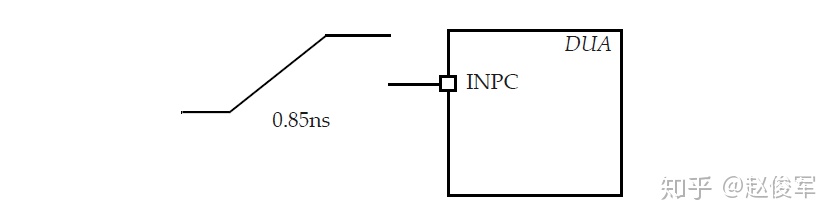
● **set\_driving\_cell** **-lib\_cell** INV2 **-library** tech13g [**all\_inputs**]

● **set\_driving\_cell** **-lib\_cell** BUFFD4 **-library** tech90gwc [**get\_ports** {testmode[3]}]

与**set\_drive**约束一样，**set\_driving\_cell**也可用于计算第一个单元的过渡时间，并在任何互连情况下计算从输入端口到第一个单元的延迟值。

使用**set\_driving\_cell**约束的一个注意点是：由于输入端口上的电容性负载而导致驱动单元的增量延迟被视作为输入上的附加延迟被包括在内。

作为上述方法的替代方法，**set\_input\_transition**约束提供了一种在输入端口表示过渡时间的便捷方法，并且可以指定参考时钟。以下是图7-30中示例的约束以及其它约束示例：

图7-30

● **set\_input\_transition** 0.85 [**get\_ports** INPC]

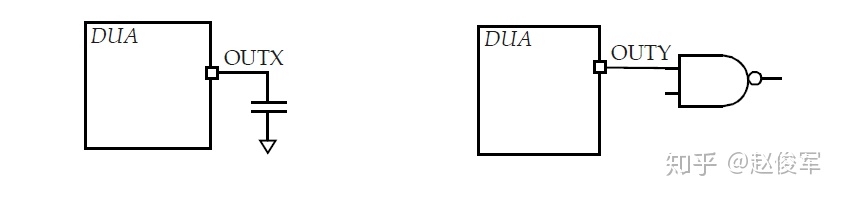
● **set\_input\_transition** 0.6 [**all\_inputs**]

● **set\_input\_transition** 0.25 [**get\_ports** SD\_DIN\*]

总之，设计人员需要指定输入端的压摆值来确定输入路径中第一个单元的延迟。在没有该约束的情况下，将假设为理想过渡值0，这显然是不现实的。

**7.7.2 电容负载建模**

**set\_load**约束在输出端口上设置了电容性负载，以模拟由输出端口驱动的外部负载。默认情况下，端口上的电容性负载为0。可以将负载显式地指定为电容值或某个单元的输入引脚电容。

图7-31

● **set\_load** 5 [**get\_ports** OUTX]

● **set\_load** 25 [**all\_outputs**]

● **set\_load** **-pin\_load** 0.007 [**get\_ports** {shift\_write[31]}]

指定输出上的负载很重要，因为该值会影响驱动输出的单元的延迟。在没有该约束的情况下，将假定负载为0，这显然是不现实的。

**set\_load**约束还可用于在设计中指定内部网络上的负载，以下是一个例子：

● **set\_load** 0.25 [**get\_nets** UCNT5/NET6]

第七章： 配置STA环境 （未完待续......）