**静态时序分析圣经翻译计划——第七章：配置STA环境 （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

20 人赞同了该文章

**7.8 设计规则检查**

STA中两个常用的设计规则是最大过渡时间-**max\_transition**和最大电容-**max\_capacitance**。这些规则将会检查设计中的所有端口和引脚是否满足过渡时间和电容的规定约束。这些规则可以使用以下命令指定：

● **set\_max\_transition**

● **set\_max\_capacitance**

作为STA的一部分，任何设计规则的违例（violation）均以裕量（slack）的形式报告。以下是些例子：

● **set\_max\_transition** 0.6 IOBANK

● **set\_max\_capacitance** 0.5 [**current\_design**]

网络上的电容是通过将所有引脚电容加上任何IO负载再加上网络上的任何互连电容的总和计算得出的。下图7-32为一个示例：

图7-32

● 网络N1的总电容 = UBUF1的A引脚电容 + UOR2的B引脚电容 + OUTP的输出负载电容 + 走线互连电容

​ = 0.05 + 0.03 + 0.07 + 0.02 = 0.17 pF

● 网络N2的总电容 = UBUF2的A引脚电容 + 走线互连电容

​ = 0.03 + 0.04 = 0.07 pF

过渡时间是延迟计算的一部分。对于图7-32中的示例（假设UBUF2单元使用线性延迟模型）：

● UBUF2的A引脚过渡时间 = 2 \* 网络N2的总电容 = 2 \* 0.07 = 0.14ns = 140ps

● 输出端口OUTP过渡时间 = UBUF2的Z引脚的驱动电阻 \* 网络N1的总电容

​ = 1 \* 0.17 = 0.17ns = 170ps

还可以为设计指定其他设计规则检查，比如：**set\_max\_fanout**（指定设计中所有引脚的扇出约束），**set\_max\_area**（用于设计）。但是，这些检查适用于综合（synthesis）而非STA。

**7.9 虚拟时钟**

虚拟时钟（virtual clock）是存在的时钟，但与设计中的任何引脚或端口均不相关。在STA中仅用作参考，以指定相对于时钟的输入延迟和输出延迟。虚拟时钟的示例如图7-33所示。 待分析设计的时钟端为CLK\_CORE，但输入端口ROW\_IN的驱动时钟为CLK\_SAD。在这种情况下，如何指定输入端口ROW\_IN的IO约束呢？同样，在输出端口STATE\_O上也会出现同样的问题。

图7-33

为了处理这种情况，可以在不指定源端口或引脚的情况下来定义虚拟时钟。对于图7-33中的示例，为CLK\_SAD和CLK\_CFG定义了虚拟时钟。

● **create\_clock** **-name** VIRTUAL\_CLK\_SAD **-period** 10 **-waveform** {2 8}

● **create\_clock** **-name** VIRTUAL\_CLK\_CFG **-period** 8 **-waveform** {0 4}

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

定义了这些虚拟时钟后，就可以相对于该虚拟时钟来指定IO约束。

● **set\_input\_delay** **-clock** VIRTUAL\_CLK\_SAD **-max** 2.7 [**get\_ports** ROW\_IN]

● **set\_output\_delay** **-clock** VIRTUAL\_CLK\_CFG **-max** 4.5 [**get\_ports** STATE\_O]

图7-34显示了输入路径上的时序关系。这将待分析设计中的输入路径限制为了5.3ns或更短。

图7-34

图7-35显示了输出路径上的时序关系。这将待分析设计中的输出路径限制为了3.5ns或更短。

图7-35

**-min**选项在**set\_input\_delay**和**set\_output\_delay**约束中指定时，可用于检查最小时序路径。使用虚拟时钟只是约束输入和输出（IO）的一种方法，设计人员还可以选择其它方法来约束IO。

**7.10 完善时序分析**

用于约束分析的四个常用命令是：

● **set\_case\_analysis** ：在单元的引脚或输入端口上指定常量值。

● **set\_disable\_timing** ：中断单元的时序弧。

● **set\_false\_path** ：指定实际不存在的路径，这意味着在STA中不需要检查这些路径。

● **set\_multicycle\_path** ：指定可能花费超过一个时钟周期的路径。

第8章将详细讨论**set\_false\_path**和**set\_multicycle\_path**约束。

**7.10.1 指定无效信号**

在设计中，某些信号在芯片的特定模式下会具有恒定值。例如，如果芯片中具有DFT（可测性设计）逻辑，则在正常功能模式下，芯片的TEST引脚将一直为0。为STA指定这样的常量值通常很有用，除了不必报告任何不相关的路径之外，这还有助于减少分析空间。例如，如果未将TEST引脚设置为常数，则可能会存在一些奇怪的长路径，而这些长路径在功能模式下永远不会存在。通过使用**set\_case\_analysis**约束来指定此类常数信号。

● **set\_case\_analysis** 0 TEST

● **set\_case\_analysis** 0 [**get\_ports** {testmode[3]}]

● **set\_case\_analysis** 0 [**get\_ports** {testmode[2]}]

● **set\_case\_analysis** 0 [**get\_ports** {testmode[1]}]

● **set\_case\_analysis** 0 [**get\_ports** {testmode[0]}]

如果设计具有多种功能模式，而只需要分析其中一种功能模式，则可以使用**set\_case\_analysis**来指定要分析的模式。

● **set\_case\_analysis** 1 func\_mode[0]

● **set\_case\_analysis** 0 func\_mode[1]

● **set\_case\_analysis** 1 func\_mode[2]

注意，可以在设计中的任何引脚上指定**set\_case\_analysis**，这个命令的另一个常见应用是可以在多个时钟上运行的设计，并且时钟的适当选择由多路复用器控制。为了使STA分析更容易并减少CPU运行时间，对每个选择的时钟分别进行STA是十分有用的。图7-36给出了一个多路复用器选择不同时钟的示例：

图7-36

● **set\_case\_analysis** 1 UCORE/UMUX0/CLK\_SEL[0]

● **set\_case\_analysis** 1 UCORE/UMUX1/CLK\_SEL[1]

● **set\_case\_analysis** 0 UCORE/UMUX2/CLK\_SEL[2]

第一个**set\_case\_analysis**选择了PLLdiv16被用于MIICLK，PLLdiv8的时钟路径被阻塞并且不会通过多路复用器传播。因此，没有使用时钟PLLdiv8分析任何时序路径（假设时钟在多路复用器之前不驱动任何触发器）。类似地，最后一个**set\_case\_analysis**选择了SCANCLK 被用于ADCCLK，并阻塞了CLK200的时钟路径。

**7.10.2 中断单元内的时序弧**

每个单元都有从其输入到输出的时序弧，并且时序路径可能会通过这些时序弧中的其中一个。在某些情况下，单元中的一条路径可能无法发生。例如可能有这样一种情况，其中时钟连接到多路复用器的选择端，而多路复用器的输出是数据路径的一部分。在这种情况下，中断多路复用器选择引脚和输出引脚之间的时序弧可能很有用。图7-37为一个示例，通过多路复用器选择端的路径不是有效的数据路径。可以使用**set\_disable\_timing**命令来中断这种时序弧。

图7-37

● **set\_disable\_timing** **-from** S **-to** Z [**get\_cells** UMUX0]

由于时序弧不再存在，因此需要分析的时序路径更少。类似用法的另一个示例是取消触发器的最小时钟脉冲宽度检查。

使用**set\_disable\_timing**命令需要格外小心，因为它会删除通过指定引脚的所有时序路径。在可能的情况下，最好使用**set\_false\_path**和**set\_case\_analysis**命令。

**7.11 点对点约束**

可以通过使用**set\_min\_delay**和**set\_max\_delay**命令来约束点对点路径，这将引脚到引脚之间的路径延迟限制在了命令指定值内。该约束将覆盖所有默认的单周期时序路径以及此类路径的任何多周期路径约束。**set\_max\_delay**约束了指定路径的最大延迟，而**set\_min\_delay**约束了指定路径的最小延迟。

● **set\_max\_delay** 5.0 **-to** UFF0/D

● **set\_max\_delay** 0.6 **-from** UFF2/Q **-to** UFF3/D

● **set\_max\_delay** 0.45 **-from** UMUX0/Z **-through** UAND1/A **-to** UOR0/Z

● **set\_min\_delay** 0.15 **-from** {UAND0/A UXOR1/B} **-to** {UMUX2/SEL}

在上述示例中，需要注意的是，使用非标准的内部引脚将迫使它们成为起点和终点，并在这些点处分割路径。

还可以类似地指定从一个时钟到另一个时钟的点对点约束。

● **set\_max\_delay** 1.2 **-from** [**get\_clocks** SYS\_CLK] **-to** [**get\_clocks** CFG\_CLK]

● **set\_min\_delay** 0.4 **-from** [**get\_clocks** SYS\_CLK] **-to** [**get\_clocks** CFG\_CLK]

如果路径上有多个时序约束，例如时钟频率约束、**set\_max\_delay**和**set\_min\_delay**，则最严格的那个约束是始终用于检查的约束。多个时序约束可能是先应用某些全局（global）约束，然后再应用某些局部（local）约束。

**7.12 路径分段**

路径分段（path segmentation）是指将时序路径分解为可以进行时序分析的较小路径。

时序路径具有起点和终点，可以使用**set\_input\_delay**和**set\_output\_delay**命令在时序路径上创建其它起点和终点。通常在单元的输出引脚上指定**set\_input\_delay**来定义新起点，而通常在单元的输入引脚上指定**set\_output\_delay**来定义新终点。这些约束定义了新的时序路径，它是原始时序路径的子集。

考虑图7-38中所示的路径。为SYSCLK定义时钟后，待分析的时序路径即为从UFF0 / CK到UFF1 / D。如果仅对报告从UAND2 / Z到UAND6 / A的路径延迟感兴趣，则可以使用以下两个命令：

● **set** STARTPOINT [**get\_pins** UAND2/Z]

● **set** ENDPOINT [**get\_pins** UAND6/A]

● **set\_input\_delay** 0 $STARTPOINT

● **set\_output\_delay** 0 $ENDPOINT

图7-38

定义这些约束会导致从UFF0 / CK到UFF1 / D的原始时序路径被分段，并分别在UAND2 / Z和UAND6 / A处创建内部起点和内部终点。现在，时序报告将明确显示此新路径。请注意，还会自动创建另外两条时序路径，一条从UFF0 / CK到UAND2 / Z，另一条从UAND6 / A到UFF1 / D。因此，原始的时序路径已被分为了三个部分，每个部分分别进行时序分析。

**set\_disable\_timing**、**set\_max\_delay**和**set\_min\_delay**命令也会对时序路径进行分段。

第七章： 配置STA环境 （完）