静态时序分析

IC芯片设计中的静态时序分析实践

读书笔记

# 一、引言

# 二、概念

## 2.1、时序分析（Timing Analysis）通常指两种分析方法

静态时序分析（Static Timing Analysis）和时序仿真分析（Timing Simulation）。

## 2.2、CMOS单元建模

标准单元的一个输出引脚可驱动多个后续单元。输出引脚自身电容，连接后续单元的导线电容以及后续单元的输入电容之和构成标准单元的驱动电容。驱动电容的大小将影响标准单元的时序。

标准单元输出电路中的上拉和下拉结构在对驱动电容充放电时，等效为电阻。上拉结构越大，等效上拉电阻越小，驱动能力越强；同理，下拉结构越大，等效下拉电阻越小，驱动能力越强。

## **2.3、电平翻转波形**

设，上拉等效电阻Rdh

下拉等效电阻Rdl

电源电压Vdd

公共端电压Vss

则：

输出由“0”转换为“1”时，输出端电压为：

Vout = Vdd \* [1 - e-t/(Rdh \* Cload)], 电流方向是从电源Vdd经Rdh到Cload。

输出由“1”转换为“0”时，输出端电压为：

Vout = Vdd \* e-t/(Rdl \* Cload), 电流方向是从Cload经Rdh到Vss。

## **2.4、传播延时（Propagation Delay）**

对于标准单元，输入、输出信号翻转并达到电源幅值的同一百分值为参考时间点时，输出信号对于输入信号的延迟时间是传播延时。通常单元內部路劲越短，传播延时越小。如果上、下拉结构不同，信号上升、下降延迟不同。

## 2.5、转换时间（Translation Time）和波形转换率（Slew Rate）

信号电平上升或下降所需要的时间为转换时间，转换时间的倒数是波形转换率。通常驱动能力越强转换时间越短，转换率越高。

## **2.6、信号之间的偏移（Skew）以及自身的频率稳定性（Jitter）**

多个信号之间的时序差称为偏移，信号可以是数据（Data Skew）也可以是时钟（Clock Skew）。Skew通常由信号路径长短不一致引起。

单一信号的周期或频率不稳定而产生Jitter。

## **2.7、时序弧（Timing Arc）**

某一个单元的输入与输出的时序关系。

## **2.8、最小和最大时序路径**

数据通路上，信号通过逻辑路径的总时间被称为路径延迟。有最大路径延迟的路径叫作最大路径、晚路径；有最小路径延迟的路径叫作最小路径、早路径。

## **2.9、时钟域**

由同一时钟驱动的一组触发器被称为该时钟的时钟域。

## **2.10、工作条件**

工作条件由工艺（Processs）、电压（Voltage）和温度（Temperature）PVT共同定义。单元延迟和互联延迟是在特定的工作条件下计算的。

# 三、标准单元库

电路单元的输入输出特性通常用Liberty格式文件来描述。其特征包括：1）引脚电容，通常单元引脚电容越大，延迟越大；在大多数情况下，延迟随着输入转换时间的增加而增加，在少数情况下，单元的延迟可能与输入转换时间呈现非单调性，更大的输入转换时间可能产生更小的延迟。

## **3.1、线性时序模型（Linear Delay Model）**

单元延迟D = D0 + D1\*S + D2\*C

式中，D0，D1，D2是常数，S是输入转换时间，C是输出负载电容。

## **3.2、非线性时序模型（Non-Linear Delay Model）**

NLDM模型主要有：Table Model、ECSM（电流源模型）

## **3.3、组合逻辑时序模型**

## **3.4、时序逻辑时序模型**

3.4.1、同步检查：建立时间和保持时间

建立时间是测量最晚的数据信号（Latest Data Signal）应先于有效时钟沿的时间间隔。保持时间是有效时钟沿之后，数据输入必须保持稳定的最小时间。

1. 建立时间和保持时间的测量和计算参考点是有效时钟沿。
2. 建立时间 + 保持时间 > 0。
3. 数据变化时刻，应在建立时间与保持时间窗口之外，否则电路会进入亚稳态。
4. 通常有效时钟沿时刻位于建立时间与保持时间窗口之间，此时建立时间和保持时间的计算值均为正值；如果有效时钟沿时刻位于建立时间与保持时间窗口之外，则计算出的建立时间或保持时间可能为负值。
5. 建立时间或保持时间出现负值的原因是，计算参考位置以及计算参考时刻的选择造成的。通常的计算参考位置是芯片的引脚，参考时刻也是有效时钟沿到达芯片时钟引脚的时刻。但是芯片内部数据路径与时钟路径存在延时，这两个路径的延时差异可能会导致建立时间或保持时间出现负值。如果参考点选在芯片内部锁存点（Latch Point）以及有效时钟沿到达芯片内部时钟输入点的时刻，那么建立时间和保持时间都为正值。
6. 芯片内部数据路径长度小于时钟路径长度时，建立时间可能为非正值。此时，（计算的参考点是芯片引脚以及有效时钟沿到达时钟引脚）数据引脚可以在有效时钟引脚后变化而不会造成建立时间违例。
7. 芯片内部数据路径长度大于时钟路径长度时，保持时间可能为非正值。此时，（计算的参考点是芯片引脚以及有效时钟沿到达时钟引脚）数据引脚可以先于时钟引脚变化而不会造成保持时间违例。
8. 在某些芯片手册中可以看见保持时间为0，即上面g情况。若芯片内部数据路径与时钟路径在设计时保持等长，则仅存在e情况。对于f、g可理解为江湖歪门邪道。

3.4.2、异步检查

a、恢复时间和移除时间检查（Recovery and Remove Check）

异步引脚（如：异步清除Clear，异步置位Set）可以覆盖单元的任何同步行为。当某个异步引脚有效（Active）时，输出由异步引脚控制而非同步时钟控制。当异步引脚无线（Inactive）时，有效时钟沿开始锁存数据输入端数据并经一定的延迟后出现在输出端。

恢复时间是异步引脚被设置为无效（De-asserted）后，在下一个有效时钟沿之前需要保持稳定的最短时间。

移除时间是异步引脚被设置为无效（De-asserted）前，在某一个有效时钟沿之后

需要保持稳定的最短时间。

理解异步恢复时间：有效时钟沿之前，异步引脚持续无效的最短时间。恢复的意思是电路由异步状态恢复到同步工作状态？

理解异步移除时间：？

b、脉冲宽度检查（Pulse Width Check）

确保单元输入引脚的脉冲宽度满足最小要求。比如时钟输入引脚脉冲宽度必须大于指定值，否则不能正确锁存数据。最小脉冲宽度检查可以指定高脉冲或低脉冲。

3.4.3、传播延迟

传播延迟是从时钟有效沿到输出上升或下降沿的时间差。tco

## **3.5、状态相关的时序模型**

状态相关时序模型（Stat-Dependent Model）：输入与输出之间的时序弧取决于该逻辑块的其它引脚。状态相关模型可以用来指定时序库中任何属性（Attribute）。如：timing()、power()、leakage\_power()等。

## **3.6、黑箱（Black Box）接口时序模型**

黑箱模型，描述任意模块的IO接口上的时序弧，可能有以下四个：

1. 对于组合逻辑路径的输入到输出时序弧。
2. 从同步输入到相应的时钟引脚的建立和保持时间时序弧。
3. 从异步输入到相应的时钟引脚的恢复和移除时间时序弧。
4. 从时钟引脚到输出引脚的传播时间延迟。

## **3.7、先进时序模型**

LDM、nLDM模型将单元时序弧延迟看作是输入转换时间和输出负载电容的函数。但是单元的输出负载并不是纯容性的，而是RC串联负载。因此LDM、nLDM模型需要改进。常见的改进方法有：CCSM（Composite Current Source Model）、ECSM（Effective Current Source Model）等。由于互连线RC和输入器件的Miller Effect，等效负载是非线性电容。

串扰噪声分析模型（CCSN）、直流余量模型、抗噪声模型等。

## **3.8、功耗建模**

包括动态功耗（Active Power）、~~待机功耗（Standby Power）、~~漏电功耗（Leakage Power）。

3.8.1、Active Power

单元的动态功耗由输出开关功耗和内部开关功耗组成。输出开关功耗与单元类型无关，只与输出负载、开关频率、单元供电电压相关。内部开关功耗与单元类型、输出负载、开关频率、单元供电电压相关，单元库提供相关描述。

3.8.2、Leakage Power

随着工艺尺寸的缩小，漏电功耗变得显著，和动态功耗相比较无法忽略。漏电功耗主要由亚阈值电流和栅极氧化物隧道电流引起。提高单元阈值电压可降低亚阈值电流，但单元工作频率降低。同时，阈值漏电流随着结温（Junction Temperature）的升高而非线性增大；常温下，栅极隧道电流在65nm或更先进工艺下，与阈值电流相当，并随工艺尺寸减小而增大。

## **3.9、单元库中的其它属性**

3.9.1、面积规范：提供单元或单元组的面积。

3.9.2、功能规范：指定引脚或引脚组的功能。

3.9.3、SDF条件

SDF条件属性支持标准延迟格式（Standard Delay Format）文件的生成，以及反标（Back annotation）时的条件匹配。单元中状态相关时序分析由关键字sdf\_cond指定。

**·**标称条件（标称工艺、电压、温度）指定了库在特征化时的环境条件。当实际工作条件与标称条件不一致时，通过计算得到的时序值就需要减免（Derate），这通过库文件中的K系数（K-factor）来实现。K系数主要分为：工艺、温度和电压三类减免系数。

# 四、互连寄生参数

## **4.1、互连线电阻、电容和电感（就是互连寄生参数）**

互联电阻可认为是单元输出引脚与扇出单元输入引脚间的电阻。

互联电容来自于走线，是由接地电容和邻近信号线间的电容。

电感的影响在芯片内可以忽略，仅在封装和板级分析时考虑。

互联走线L任一部分的电阻和电容被理想地表示为一个分布式RC树（Distributed RC Tree）。RC树上的总电阻和总电容分别为Rt和Ct，单位长度电阻和单位长度电容分别是Rp和 Cp，总长度是L。Rt = Rp \* L；Ct = Cp \* L。

T模型：总的电容Ct连接在电阻树（Resistive Tree）的中间。总电阻Rt被均分成两个电阻，Ct连接在两电阻之间，形成T字型。

Pi模型：总电容Ct被均分成两个电容，Rt连接在两电容之间，形成Π字型。

## **4.2、线负载模型**

在布局规划（Floorplan）或布局（Layout）之前，**线负载模型（Wire Load Model）可以用来估算电容、电阻和互连线的面积开销**。

4.2.1、互联树

一但由WLM模型估算出目标区域的电阻、电容和面积后，就可以认为预布局互连线的Rwire 和Cwire就确定了，下一步就是明确换连线的结构。在预布局阶段，互联RC树用三种方式来表示，三种方式的互联长度一致，因而预估互联电阻和电容也一致。

1）最佳情况树：假设终点（负载）引脚与驱动是紧邻的。所以驱动到终点引脚没有线电阻，仅有终点引脚电容与线电容之和作为驱动的负载电容。相当于负载引脚在RC树的近端。

2）平衡树：每个终点（负载）引脚分别与驱动星型等长连接。每条到终点的线路都等分WLM模型估算的Rwire 和Cwire。

3）最差情况树：在此情况下，假设所有终点引脚聚集在线的最远端。

4.2.2、指定线负载模型及模式

模型：set\_wire\_load\_model: light、conservative、typical、aggressive

不同的模型决定于电路块的面积大小，平均线长。

模式：set\_wire\_load\_mode: top、enclosed、segmented。当线穿过层次边界时，使用不同的线负载模式。模式指定模型的优先级。如top模式，指定顶层快的线负载模型具有优先级；segmented模式，指明底层快的线负载模型具有优先级。

## **4.3、提取的寄生参数的表示方法**

在预布局阶段，通过线性负载模型预估电路的寄生参数。在布局（Layout）后提取详细寄生参数，可以用以下3种格式来描述

1. 详细标注寄生参数格式（Detailed Standard Parasitic Format，DSPF）
2. 精简标准寄生参数格式（Reduced Standard Parasitic Format，RSPF）
3. 标准寄生参数交换格式（Standard Parasitic Exchange Format，SPEF）
4. 新思二进制寄生参数格式（Synopsis Binary Parasitic Format，SBPF）

## **4.4、耦合电容的表示方法**

## **4.5、层次化设计方法**

## **4.6、减少关键线的寄生参数**

4.6.1、减小互连电阻

4.6.2、增加走线间距

4.6.3、关联走线的寄生参数（相似功能线要有相同的寄生参数）

# **五、延迟计算**

## **5.1、概述**

5.1.1、延迟计算的基础

依据第二章中NLDM模型，单元延迟和输出转换时间是该单元输入转换时间和总输出电容的函数，因此单元延迟可以基于输入转换时间和输出电容用查表法得出。

5.1.2、带有互连线的延迟计算

1）预布局时序

在预布局阶段时序验证时，互连寄生参数是用线负载模型预估。

2）布局后时序

布局后，金属走线的寄生参数将映射为驱动和终点单元间的RC网络。

## **5.2、使用有效电容的（驱动）单元延迟**

NLDM不能适用于RC树的情况，需要使用“等效”电容来处理线电阻寄生参数的影响。依据PI模型，Ceff = C1 + k \* C2, 0<= k <=1。如果互连电阻忽略为0，则k=1，Ceff = C1 + C2；如果互连电阻为无穷大，则k=0，Ceff = C1。

有效电容的影响因素：

1. 驱动单元特性
2. 负载的特性

有效电容法能近似计算单元延迟，但不能近似单元输出波形，因为有效电容负载与真实RC负载的差别较大，导致有效电容作为负载时的单元输出波形与真实波形必然存在较大差异，输出转换时间差别也较大。为了得到近似的输出转换时间，需另找办法。但是对于目标单元来说，已知输入转换时间和计算得出的近似的有效电容负载，便可通过NLDM模型查表得出目标单元的延迟。

## **5.3、互连线延迟**

单元间的互连线电路模型是RC树。使用Elmore方法可以近似计算RC树的延迟。

使用Elmore方法计算RC树延迟，需要满足三个条件：

1）树有唯一的输入（源）节点；

2）没有电阻回路（Resistive Loop）

3）所有电容都位于节点和电源地之间

设有N级RC树，Elmore延迟模型表示如下：

Td1 = C1 \* R1

Td2 = Td1 + C2 \* (R1 + R2)

Td3 = Td1+ Td2 + C3 \* (R1 + R2 + R3) = C1 \* R1 + C2 \* (R1 + R2) + C3 \* (R1 + R2 + R3)

Tdn =

设一条具有Rwire和Cwire寄生参数线，下游扇出单元的输入电容为Cload，根据：

T模型：Td = Rwire\*Cwire + Cload\*(Rwire + Rwire)

PI模型：Td = 0\*Cwire + Rwire\*(Cwire + Cload)

上述两个计算结果一致。

算例：

对一条线使用线负载模型和平衡RC树以及最差情况RC树，扇出是N，分别计算Elmore延迟：

平衡树：Td = (Rwire/N) \* (Cwire + Cpin), Cpin为某一个分支的输入电容。

最差树：Td = Rwire \* (Cwire + Cpins), Cpins为所有分支的输入电容。

其它互连线计算模型：AWE（Asymptotic Waveform Evaluation，渐近波形估算）、Arnoldi算法等。

通过等效电容模型估算单元延迟，通过Elmore模型计算互连线延迟，如此重复可计算芯片任意路径延迟。但书中的说明还存在遗漏，等效电容法仅能近似估算单元延迟，不能估算单元输出转换时间，单元输出信号经过互连线其波形发送变化，对于下一级的来说，输入波形或转换时间就未能确定，因而下一级的单元延迟也就无法估算。书中74页末尾处提到“有很多种办法来计算延迟和互连线终点处的波形”，但没有阐述具体内容。

## **5.4、转换率融合**

对于多输入单元或多驱动线（单元），需要确定某个输入转换时间作为标准，这样的输入点称为转换率融合点。输入转换时间的选择取决于时序分析要求。

在进行**最大时序路径**分析时有两种选择可能性：

1. 最差转换率传播（Worst Slew Propagation）：选择融合点处**最慢**转换率的输入
2. 最差到达时间传播（Worst Arrival Propagation）：选择融合点处**最晚**到达的输入。

在进行最小时序路径分析时也有两种选择可能性：

1. 最佳转换率传播（Best Slew Propagation）：选择融合点处**最快**转换率的输入。
2. 最佳到达时间传播（Best Arrival Propagation）：选择融合点处**最早**到达的输入。

大部分静态时序分析工具会使用**最差和最佳转换率传播**，因为是保守的分析方式。但是，当分析一条路径时，可以使用准确的转换率传播方式，明白什么情况下是过于悲观的。

## **5.5、不同的转换率阈值**

用于确定前后级连接时，不同转换阈值的折算。相关前述的转换率减免系数（Slew Derate）。目的是确定准确的转换时间（率）。

## **5.6、不同的电压域**

延迟是通过电压50%的阈值点来计算。对于不同电压域的引脚，其阈值点电压也不同。

## **5.7、路径延迟计算**

5.7.1、组合逻辑路径计算

通常，通过互连线上升和下降的延迟不一样，因为驱动单元输出的戴维南模型（Thevenin Source Model）不一样。

Tdrise = ……

Tdfall = ……

5.7.2、到触发器路径

1）输入到触发器路径（输入数据和输入时钟）

输入数据到触发器数据输入端的路径：通常上升和下降的延迟不一样，原因同上。

输入时钟到触发器时钟输入端的路径：通常计算上升延迟。

1. 触发器到触发器路径（前级触发器到后级触发器）

数据通路：Tco + Tdata

发射时钟：Tlaunch

捕获时钟：Tcapture

5.7.3、多路径

在任意两个点之间，可能有多条路径。最长路径就是用时最长的路径，也称最差路径（Worst Path）、晚路径（Late Path）、最大路径（Max Path）；最短路径就是用时最短的路径，也称最佳路径（Best Path）、早路径（Early Path）、最小路径（Min Path）。

## **5.8、裕量计算**

裕量（Slack）是信号的需要到达时间（Require Time）减去实际到达时间（Arrival Time）之差。

# 串扰与噪声

## **6.1、概述**

串扰噪声是两个以上信号之间活动的非故意耦合。深亚微米工艺中容易产生串扰噪声的原因：

1. 越来越多的金属层：0.3至0.25um工艺有4或5层金属，而65nm至45nm有10层以上金属。
2. 金属连线高且细：相邻连线的侧壁间存在耦合电容。
3. 更小的尺寸导致更高的连线密度。
4. 器件密度大：相互影响大。
5. 更高的频率导致更大的信号边缘斜率。
6. 更低的供电电压。

## **6.2、串扰毛刺分析**

6.2.1、基础

产生影响的信号称为侵害者（Aggressor），受影响的信号称为受害者（Victim）。

串扰的本质是信号线间的电磁耦合，如果信号线间介质的介电常数为无穷大，磁感应强度为0，则不存在耦合串扰。

通常，串扰造成的噪声影响有两种：毛刺（Glitch），由侵害者电平翻导致受害者出现耦合噪声；串扰增量延迟，由串扰导致的信号时序变化，侵害者和受害者均能出现。

通常两条线之间的耦合电容被**描述**（不是近似计算用）为一个集总（Lumped）电容Cc，而不是分布（Distribute）电容。

产生毛刺信号（功率？）的大小取决于多种因素：

1. 连线间耦合电容：耦合电容越大，毛刺信号越强。
2. 侵害线的转换率：转换率越高，毛刺越大。
3. 受害线的接地电容：受害线接地电容越小，毛刺越大。
4. 受害线的驱动能力：驱动能力越弱，毛刺越大。

6.2.2、毛刺的类型

1）上升和下降毛刺：受害线低电平稳定时，其上正向（Positive）或上升（Rise）glitch；受害线高电平稳定时，其上负向（Negative）或下降（Fall）glitch。

2）过冲（Overshoot）和下冲（Undershoot）：受害线高电平稳定时，串扰继续推高其电平值，并短暂大于标准高电平；受害线低电平稳定时，串扰继续拉低其电平值，并短暂小于标准低电平。

详细的毛刺计算是基于库模型，所需相关参数属于标准单元库。

6.2.3、毛刺的阈值和传播

毛刺的高度和宽度是导致其在单元电路中传播的关键因素。基于直流或交流噪声分析可得出结论。

1. 直流阈值（DC Threshold）

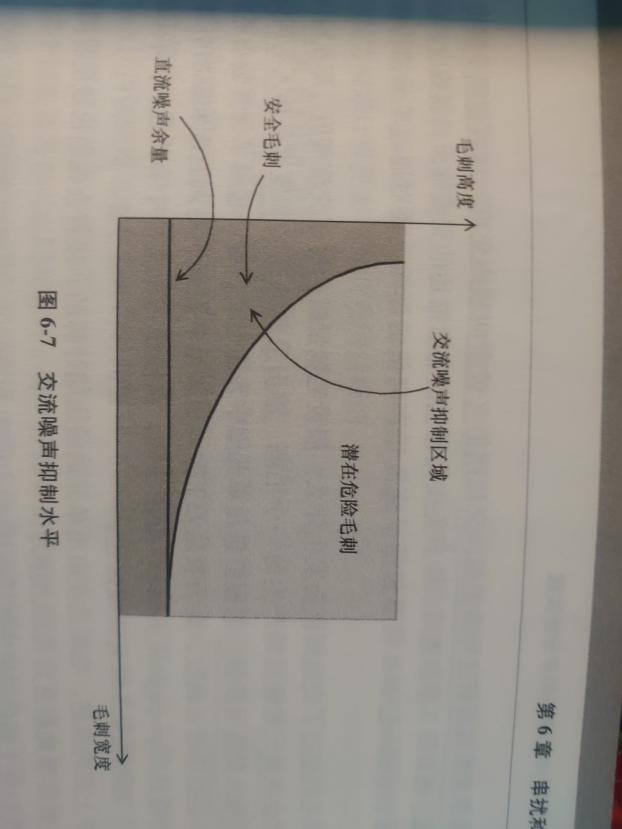
直流噪声余量（DC Noise Margin）是针对毛刺量级的检查，是检查单元输入上直流噪声大小的极限。只要噪声电平不超越单元VILmax、VIHmin界限，单元不受毛刺干扰。此外，**不是所有**超越VILmax、VIHmin界限的毛刺都能改变单元的状态，这还与毛刺的持续时间（宽度）有关。持续时间越长，改变单元状态的可能性越高。

1. 交流阈值（AC Threshold）

噪声分析的直流余量极限是保守的，通常验证毛刺的影响时还要考虑毛刺的宽度和单元的输出负载，这可称为交流阈值，可以用单元的惯性（Inertia）来解释。通常，单极单元会阻止任何比该单元延迟窄得多的输入毛刺。同时，增大单元的输出负载，也增大了单元延迟，可以进一步降低输入毛刺的影响。（可以理解为惯性与输出负载决定了单元变化所需的能量阈值，毛刺的能量较小时，无法达到能量阈值，单元的状态不会因毛刺的输入而改变）

如果输入毛刺高度和宽度超过了交流阈值，那么该毛刺将通过单元传播。输出毛刺的高度和宽度是输入毛刺宽度、高度、和单元负载的函数。

毛刺（Glitch）：上升毛刺（Rise Glitch，由propagated\_noise\_high或noise\_immunity\_high建模）、下降毛刺（Fall Glitch，由propagated\_noise\_low或moise\_immunity\_low建模）、过冲毛刺（Overshoot Glitch，由noise\_immunity\_above\_high建模）、下冲毛刺（Undershoot Glitch，由noise\_immunity\_below\_low建模）。



6.2.4、多侵害者的噪声累积

将各侵害者毛刺耦合累加，计算出受害者的累积效应，此方法可获得受害线上最差毛刺，但结果过于保守。可计算每个侵害者毛刺的方均根RMS（Root Mean Square）值来确定受害线上的毛刺量级。

6.2.5、侵害者的时序相关性

要计算多侵害者引起的串扰噪声，必须分析侵害线的时序相关性。由于所有侵害线不是同时发生电平翻转（同时发生只是概率极低的最差情况），因而在受害线上出现的串扰应尽可能的根据各侵害线电平翻转的时序分时段计算，这可以减少计算的悲观程度。

6.2.6、侵害者的功能相关性

在特定条件下，受害线只可能受到功能相关的侵害线的串扰，而不可能受到不相关的侵害线串扰，因此在计算累积效应时需排除不相关者，避免过于保守。

## **6.3、串扰延迟分析**

6.3.1、基础

连线寄生电容由接地电容Cg和线间耦合电容Cc构成。其等效电容会因线间电平稳定情况而变化。根据侵害线相对受害线电平翻转的情况，等效线间耦合电容通常非三种情况：

1. 侵害线电平稳定，等效于没有串扰，总寄生等效电容为：Cg+Cc ，受害线向总寄生电容的充电量Q = （Cg+Cc）\* Vdd，Vdd是电源电压。此时Cc 看作对地电容。

延迟是**基准值**。

1. 侵害者与受害者电平同相翻转，受害线驱动单元得到了侵害线的帮助，是负串扰。受害线驱动单元提供的电荷为（Cg + 0）\* Vdd甚至更小，连线延迟也将变小，减小部分通常被标记为**负串扰延迟**（Negative Crosstalk Delay，相对于基准值是负值）。通常在最小路径分析时考虑。
2. 侵害者与受害者电平反相翻转，受害线驱动单元负载被加重，驱动单元提供的电荷为（Cg + 2Cc）\* Vdd甚至更大，连线延迟也将变大，增大部分通常被标记为**正串扰延迟**（Positive Crosstalk Delay，相对于基准值是正值）。通常在最大路径分析时考虑。

6.3.2、正向串扰和负向串扰

侵害线与受害线电平同向翻转时，减小了受害线驱动单元所需的电荷，减少了驱动单元的延迟和受害线的互联延迟。

侵害线与受害线电平反向翻转时，增大了受害线驱动单元所需的电荷，增加了驱动单元的延迟和受害线的互联延迟。

6.3.3、多侵害者的累积

同毛刺分析类似，将每个侵害者串扰增量相加后作为受害者受到的累积影响，可能过于保守，但这可以指明受害者的最差串扰延迟。通常使用RMS值作为受害者的累积串扰值。

6.3.4、侵害者和受害者的时序相关性

6.3.5、侵害者和受害者的功能相关性

## **6.4、考虑串扰延迟的时序分析**

现需要为每个**单元**和互连线计算4种串扰延迟影响：

1. 正向上升延迟（Positive Rise Delay）：上升沿推迟到达；
2. 负向上升延迟（Negative Rise Delay）：上升沿提前到达；
3. 正向下降延迟（Positive Fall Delay）：下降沿推迟到达；
4. 负向下降延迟（Negative Fall Delay）：下降沿提前到达；

然后在时序分析验证最大和最小路径（对应于建立时间和保持时间检查，立马保命）时，分析串扰的影响。对于发射触发器和捕获触发器的时钟路径将按不同的方式处理。

带有串扰分析的STA通过验证数据路径和时钟路径的**最差**情况串扰延迟来保证设计的时序。

时钟信号是非常关键的，任何时钟树上的串扰都会直接转换为时钟抖动（Jitter）。

6.4.1、建立时间分析

建立时间（最大路径）检查的最坏情况是：发射时钟路径和数据路径都有正向串扰（使得发射数据延迟最大，到达捕获触发器的时间滞后），捕获时钟路径有负向串扰（捕获时钟沿到达捕获触发器的时间提前）。

建立时间检查时，捕获时钟沿**通常**比发射时钟沿晚一个时钟周期。

时钟路径一个分支连接发射触发器，另一个分支连接捕获触发器。共同的主干部分即共同时钟路径（Common Clock Path）可能在计算发射和捕获时钟沿时，有不同的延迟。因而在分析时，对于发射时钟路径和捕获时钟路径应分别完整的计算，这不同于下面的保持时间分析。

建立时间（最大路径）分析假设：

1. **完整的**发射时钟路径可能正向串扰延迟，所以数据发射时刻可能延迟；
2. 数据路径可能正向串扰延迟，所以数据传输可能延迟；
3. **完整的**捕获时钟路径可能负向串扰延迟，所以可能会提前捕获数据。

6.4.2、保持时间分析

保持时间（最小路径）检查的最坏情况时：发射时钟路径和数据路径都有负向串扰（使得发射**新数据（下一次数据）**延迟最小，到达捕获触发器的时间提前），捕获时钟路径有正向串扰（捕获时钟沿到达捕获触发器的时间滞后）。

保持时间检查时，捕获时钟沿与发射时钟沿通常一致。对于时钟路径上重叠共同的部分，串扰影响一致。所以，最差情况保持时间分析可删除共同时钟路径的串扰影响。

保持时间（最小路径）分析假设：

1. 发射时钟路径（不包括共同路径）可能负向串扰延迟，所以数据发射时刻可能提前；
2. 数据路径可能负向串扰延迟，所以数据传输可能提前；
3. 捕获时钟路径（不包括共同路径）可能正向串扰延迟，所以数据捕获时刻可能滞后。

STA保持时间分析报告中，对来自共同时钟路径的串扰影响会作为单独的项目被去除，并被标记为共同路径悲观去除，CPPR（Common Path Pessimism Removal）。

## **6.5、计算复杂度考虑**

## （目的是降低大型复杂系统分析工作量，并能保证一定的精度）

6.5.1、层次化设计与分析

6.5.2、耦合电容过滤

过滤基准：

1. 小电容值：通常小于1fF的耦合电容被视为接地电容（不产生耦合效应）
2. 耦合比（Coupling Ratio）：耦合比小于千分之一通常可忽略。
3. 合并小侵害者：多个小侵害者可以映射为一个大的虚拟侵害者。该方式可能过于悲观，但可以简化分析。

## **6.6、避免噪声的技术**

1）屏蔽：相邻金属层走线通常是正交的，所以窜扰通常来自于同层走线。可在关键信号线两边做屏蔽线。屏蔽线连接至地或电源。此外，低翻转率的信号线可以与关键信号线相邻，此法可以缓解过多屏蔽线造成的布线拥塞（Routing Congestion）。

2）线段间距：较大的间距，串扰较小。

3）快速转换率：较快速的转换率不易受串扰，但易干扰他人。

4）保持良好稳定的供电

5）保护环

6）深N阱

7）隔离块

# **配置STA环境**

STA准备工作包括，1设置时钟，2指定IO时序特征，3设定伪路径和多周期路径。

## 7.1、STA环境

大部分数字电路是同步的。待分析设计（Design Under Analysis）与上下游时序电路建立正确的时序约束关系的过程即建立了STA环境。

## 7.2、指定时钟

时钟要素：

1. 时钟源：设计的一个端口，引脚，是时钟生成逻辑的一部分。
2. 周期：时钟周期。
3. 占空比：高电平持续时间，低电平持续时间
4. 沿时刻：上升沿时刻，下降沿时刻。
5. 时钟源转换时间：set\_clock\_transition -rise m -fall n [get\_clocks CLKSRC]

定义时钟；

**create\_clock** \

**-name** clockname \ 时钟名，如未指定时钟名则与端口同名，同名也是惯例

**-period** 20 \ 周期，单位ns

**-waveform** {0 5} \ 单周期内的波形 {上升沿时刻 下降沿时刻 上升沿时刻 下降、沿时刻 ……}，上升沿与下降沿须成对出现，次序不能颠倒。如未指定-waveform选项，默认值是{0 period/2}。

[ **get\_ports** SCLK] \ 目标单元或引脚

例子：

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

时钟名sysclk，周期125ns，上升沿时刻100ns，下降沿150ns（但已超过时钟周期，因而推断出下降沿在25ns），时钟端口是ARMCLK

7.2.1、时钟不确定性

不确定性可以用来对减少有效时钟周期的各种因素建模。由于考虑时钟的悲观项。

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

/ 设置时钟CLK的下一个上升沿提前到达0.2ns（周期减少0.2ns），对应建立时

/ 间裕量减少0.2ns

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

/ 设置时钟CLK的下一个上升沿滞后到达0.05ns（周期增加0.05ns），对应保持

/ 时间裕量减少0.05ns

上一章中时钟路径出现的串扰现象，可能导致时钟信号的负向（提前）或正向（滞后）延迟，这可用set\_clock\_uncertainty约束。

指定跨过时钟边界路径的不确定性，叫做**时钟间不**确定性（Inter-clock Uncertainty）。跨过不同时钟域的路径通常是指某数据路径（即某个时钟域的数据向另一时钟域传递）。

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

对经SYS\_CLK时钟域到CFG\_CLK时钟沿的数据路径作建立时间检查时，时钟沿不确定性是0.1ns，时钟沿悲观提前0.1ns。

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

对经SYS\_CLK时钟域到CFG\_CLK时钟沿的数据路径作保持时间检查时，时钟沿不确定性是0.05ns，时钟沿悲观滞后0.05ns。

7.2.2、时钟延迟

两种时钟延迟：源延迟和网络延迟。

**源延迟（也叫插入延迟，Source Latency）：**

对于时钟发生源在芯片外部（比如外部输入的时钟）的情况，源延迟是外部时钟发生源到芯片时钟引脚（时钟定义点）之间的延迟；

对于时钟发生源在芯片内部（比如内部PLL或MMCM电路）的情况，源延迟是内部时钟发生源到内部时钟缓冲输出（时钟定义点）之间的延迟。

set\_clock\_latency 1.9 -source [get\_clocks CFG\_CLK]

CFG\_CLK的源延迟是1.9ns

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

CFG\_CLK的最小源延迟是0.851ns

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

CFG\_CLK的最大源延迟是1.322ns

**网络延迟（Network Latency)：**

指时钟定义点（create\_clock）到触发器时钟引脚的延迟。

set\_clock\_latency 0.8 [get\_clocks CFG\_CLK]

CFG\_CLK的网络延迟是0.8ns

set\_clock\_latency 1.8 -rise [get\_clocks CFG\_CLK]

CFG\_CLK的上升沿网络延迟是1.8ns，-rise指的是触发器时钟引脚上的沿

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

所有时钟的下降沿网络延迟是2.1ns，-fall指的是触发器时钟引脚上的沿

源延迟与网络延迟的重要区别是，一旦设计的时钟树构建完成，网络延迟是可以由set\_propagated\_clock命令忽略的，但是，源延迟任然存在。网络延迟用于时钟树综合之前延迟估算。当时钟树综合完成后，从时钟源到触发器时钟引脚的总延迟是：源延迟加上时钟定义点到触发器的时钟树真实延迟？

## 7.3、生成时钟

主时钟派生出生成时钟。create\_clock定义主时钟，create\_generated\_clock定义派生时钟。

create\_clock -name CLKP 10 [get\_pins UPLL0/CLKOUT] #在PLL引脚CLKOUT创建主时钟CLKP，周期为10ns，占空比50%。

create\_generated\_clock -name CLKPDIV2 -source UPLL0/CLKOUT -divide\_by 2 [get\_pins UFF0/Q]

#在触发器UFF0的Q脚，创建一个派生时钟CLKDIV2

#主时钟是UPLL0/CLKOUT引脚处的CLKP。CLKDIV2周期是主时钟的2倍。

**定义一个新（主）时钟会对于产生一个时钟域，而定义派生时钟不会产生新的时钟域。主时钟及其派生时钟同属于一个时钟域，约束主时钟后不需要再额外约束派生时钟。**

**主时钟的起始点在主时钟的定义点；生成时钟的源点在主时钟的定义点而不是生成时钟的定义点；生成时钟的起始点是其定义点。**

多路时钟经多路复用器选择某一路作为有效时钟的情况下，如果复用器选择信号在电路工作时保持不变（通常情况），那么可按照STA并配合将其它无效时钟设置为伪路径。如果复用器选择信号在电路工作时动态变化，这需要对复用器输入进行时钟门控检查，确保时钟信号安全切换。

如果时钟输出被门控（可以被禁止输出），则通常在门单元的输出端定义一个生成时钟，该时钟与输入门单元的时钟相同。

create\_clock -name SYS\_CLK -period 0.1 [get\_ports SYS\_CLK] #创建主时钟SYS\_CLK

create\_generated\_clock -name CORE\_CLK -source SYS\_CLK -divided\_by 1 [get\_pins UAND1/Z] # 创建经门单元UAND1控制输出的派生时钟CORE\_CLK，定义点在

#UAND1的Z引脚

生成时钟的频率比源时钟高。

create\_generated\_clock -name PCLKx2 -source [get\_ports PCLK] -multiply\_by 2 [get\_pins UCLKMULTIREG/Q] #主时钟PCLK的两倍频派生时钟PCLKx2

多路时钟输入经门单元输出运算后的时钟信号，可分别对输入、输出独立建立主时钟，形成不同的时钟域。但每个主时钟需要手动确定源延迟。

使用-edges选项，即使用源主时钟的沿{上升沿1 下降沿2 上升沿3 ……}来定义生成时钟的{上升沿 下降沿，上升沿，下降沿……}。

create\_clock 2 [get\_ports DCLK] #主时钟{沿1 沿2 沿3 ……} 对应 {0ns 1ns 2ns ……}

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

#创建生成时钟DCLKDIV2，定义在UBUF2/Z，其上升沿在主时钟第2沿，下降沿在主时钟第4沿，下一个上升沿在主时钟第6沿，周期是源时钟DCLK的两倍。

create\_generated\_clock -name PH1CLK -source DCLK -edges {1 2 5 6} [get\_pins UAND1/Z}

#创建生成时钟PH1CLK，定义在UAND1/Z。其上升沿在主时钟的沿1时刻，下降沿在主时钟的沿2时刻, 下一个上升沿在主时钟的沿5刻时，下一个下降沿在主时钟的沿6时刻。

-edge\_shift与-edges配合使用，使相应的沿偏移，形成新波形。它指定了沿列表中的每个沿的偏移量（以时间为单位）。

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]

#创建二分频生成时钟

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

#创建生成时钟MIIDIV2，其上升沿在主时钟的沿1时刻，下降沿在主时钟的沿1时刻，下一个上升沿在主时钟的沿5时刻。-edge\_shift {0 5 0}选项指明，生成时钟的上升沿增加0ns，下降沿增加5ns，下一上升沿增加0ns，最终波形是-waveform {0 5 20}。

7.3.2、使用-invert选项生成反向时钟。该选项在其它时钟生成选项后生效，使生成时钟反向或相移180度。

7.3.3、生成时钟的时钟延迟

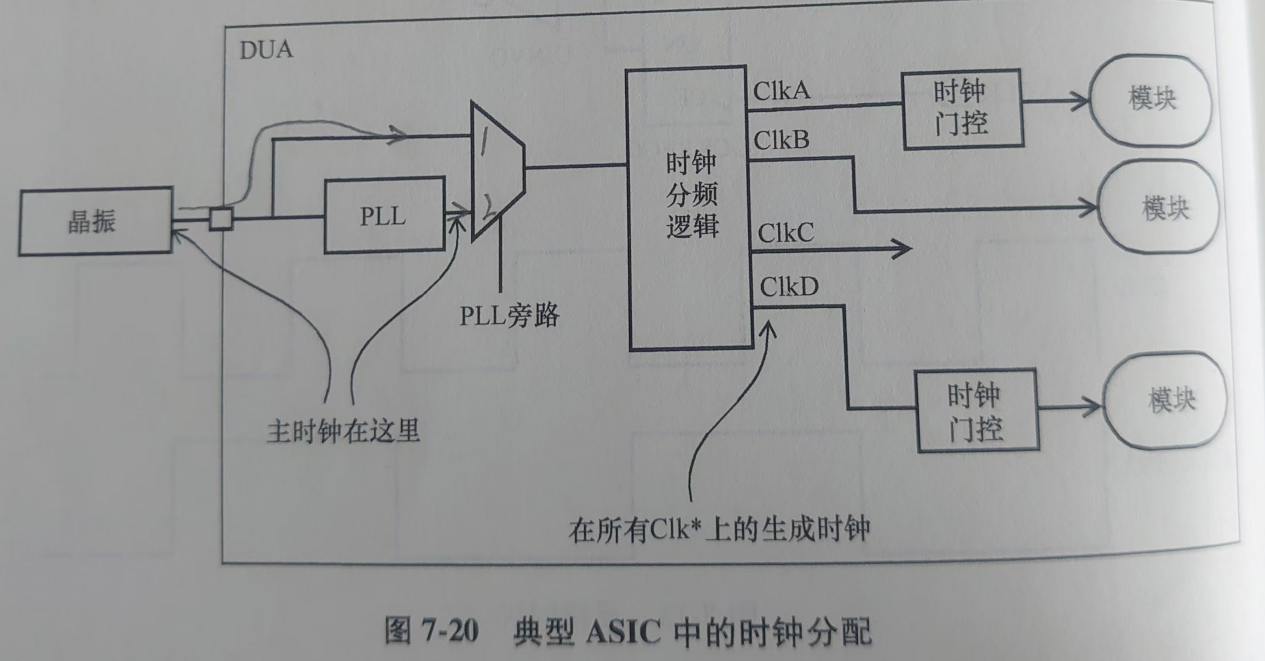
生成时钟源延迟：主时钟定义点到生成时钟定义点的延迟。

生成时钟网络延迟：生成时钟定义点到触发器时钟引脚的延迟。

被生成时钟驱动的触发器时钟引脚上的总延迟等于主时钟源延迟加上生成时钟源延迟再加上生成时钟网络延迟。

生成时钟可以继续派生时钟，每一个生成时钟只能有唯一的主时钟（此主时钟可以是一个生成时钟）。

7.3.4、典型的时钟生成场景



可定义两路主时钟：

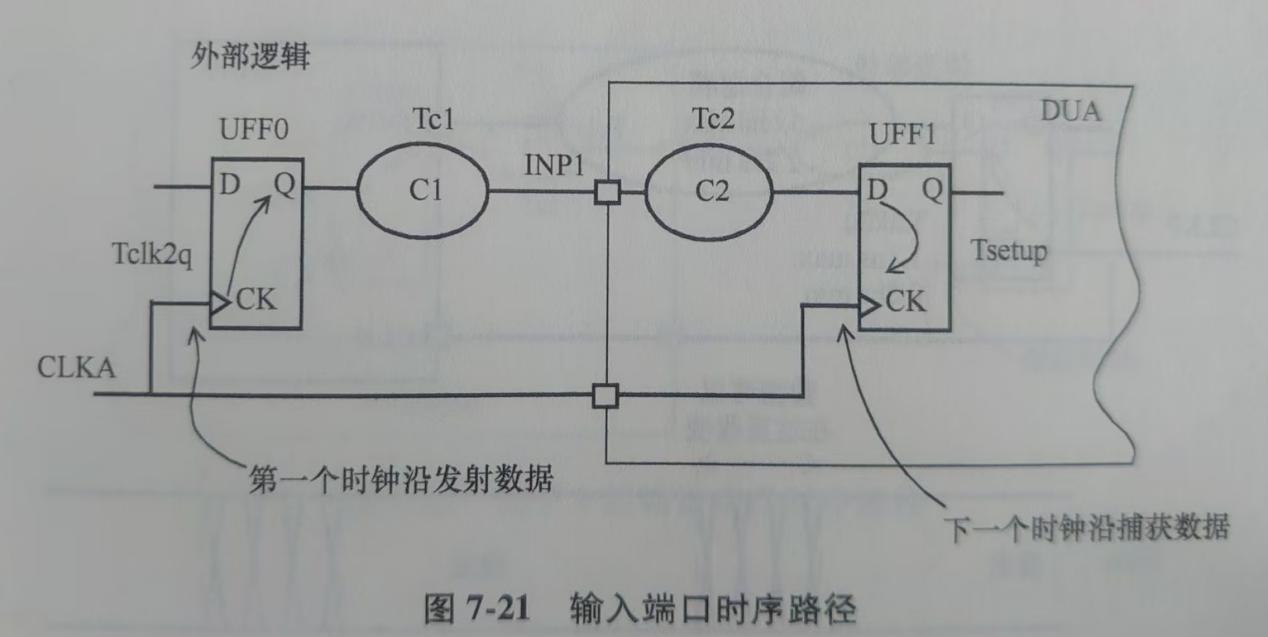
主时钟CLK1，晶振经芯片引脚进入到PLL旁路器，定义点在芯片晶振引脚。

主时钟CLK2，芯片内PLL输出到PLL旁路器，定义点在PLL输出。

假设时钟分频逻辑输出ClkA，ClkB与CLK1相关，则可将他们分别定义为CLK1的生成时钟。

假设时钟分频逻辑输出ClkC，ClkD与CLK2相关，则可将他们分别定义为CLK2的生成时钟。

## 7.4、约束输入路径



Tclk2q, UFF0的CK to Q的延迟；Tc1，组合电路C1的延迟；Tc2，组合电路C2的延迟；UFF1的建立时间 Tsetup；UFF1的保持时间Thold；T为时钟周期。有公式：

1. 0 + Tclk2q + Tc1 + Tc2 + Tsetup < 0 + T。公式1
2. 0 + Thold < 0 + Tclk2q + Tc1 + Tc2 。公式2

如果Tclk2q + Tc1 + Tc2 < Thold，那么UFF1的**老**数据还没有发射出去，就会被**新**到达的数据冲掉。

set\_input\_delay VALUE **-clock CLKA** -max 指定相对于**发射触发器有效时钟沿（DUA在下游）**，数据从发射触发器有效时钟沿开始，最晚到达芯片引脚时间（路径）。用于建立时间检查。参照公式1）

set\_input\_delay VALUE **-clock CLKA** -min 可以指定相对于**发射触发器有效时钟沿（DUA在下游）**，数据从发射触发器有效时钟沿开始，最早到达芯片引脚的时间（路径）。用于保持时间检查。参照公式2）

综上理解为，无论作何种检查（建立还是保持）如果外部输入的数据通路的延迟已知，则可以使用set\_input\_delay指定，在芯片内部设计时需要约束内部的数据路径延迟。

set Tclk2q 0.9

set Tc1 0.6

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

设置相对于**时钟CLKA发射沿**的数据通路最大延迟为0.9+0.6=1.5ns。经过1.5ns最大延迟后，新数据才能到达芯片的输入引脚INP1，那么芯片内部的保守延迟仅有T - 1.5ns，这制约着 Tc2 + Tsetup的设计目标。此时，Tc2和UFF1的Tsetup是未知而需要约束的，而Tclk2q + Tc1是已知的。

最大或最小数据输入延迟决定于上游电路的设计、制造工艺和工作条件。一方面在设计时，上游电路的数据路径与时序路径相对长短不一致，差额可大可小，相对延迟也可大可小。另一方面，快工艺可可用较小延迟，慢工艺可可用较大延迟。但是上游电路一旦固定连接，其input\_delay应该是仅受工作条件影响而变化不大的常数。

又例如：

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]

设置输入最大和最小，最长和最短，最差和最快，最大时序工艺角和最小时序工业角。相对于时钟CLKP上升沿，数据到芯片INPA引脚的最大延迟是6.7ns，最小延迟是3ns。

对于建立时间检查，使用立马保命口诀，应使用最大数据延迟6.7ns，因此芯片内部数据延迟不能超过：15ns - 6.7- Tsetup = 8.3 - Tsetup。

对于保持时间检查，使用立马保命口诀，应使用最小数据延迟3ns，因此芯片内部数据延迟不能低于：Thold - 3。（意思是如果Thold大于（set\_input\_delay -min），那么就需要增加Tc2的延迟，使得（set\_input\_delay -min）+ Tc2 > Thold；如果Thold小于（set\_input\_delay -min），那么只要Tc2的延迟大于等于0即能满足保持时间检查，因为即使Tc2无延迟，新数据经过（set\_input\_delay -min）时间后，也不会冲掉老数据）

如果芯片内部的UFF的Tsetup以及Thold是常数，那么可约束项只有Tc2。则：

(Thold - 3) < Tc2 < (8.3 - Tsetup)

进一步考虑：对于某UFF其Tsetup + Thold = Constant。结合公式1和公式2，有

Thold < Tclk2q + Tc1 + Tc2 < T - Tsetup 公式3

或

0 < Tclk2q + Tc1 + Tc2 - Thold < T - (Tsetup + Thold) = T - Constant 公式4

公式4又可分解为：

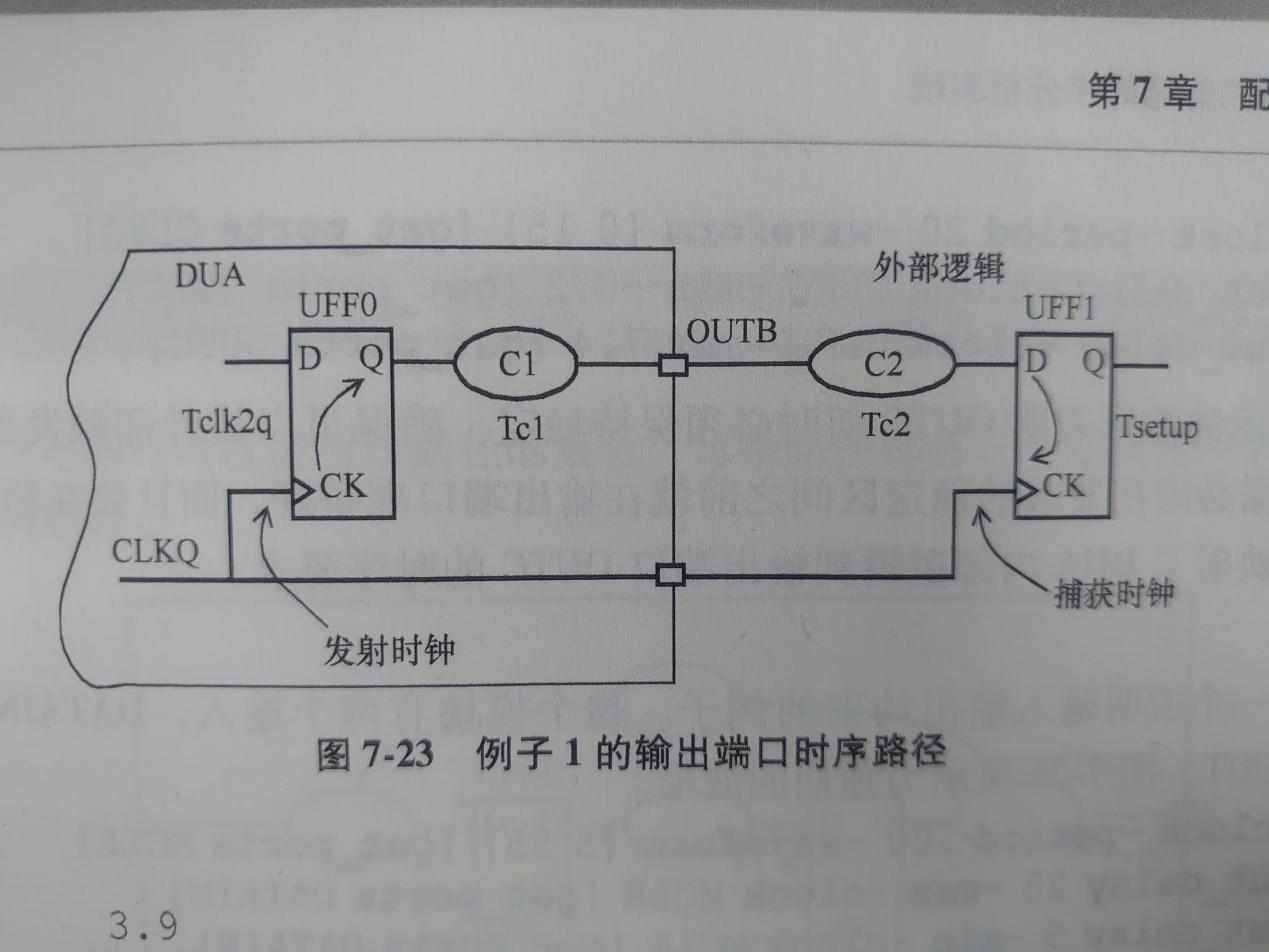
（Tclk2q + Tc1 + Tc2）min - Thold > 0，公式5

（Tclk2q + Tc1 + Tc2）max + Tsetup < T，公式6

（Tclk2q + Tc1 + Tc2）max - Thold < T - Constant，公式7

如果参考时钟沿是下降沿，那么使用-clock\_fall选项。

## 7.5、约束输出路径



由上图：Tclk2q, UFF0的CK to Q的延迟；Tc1，组合电路C1的延迟；Tc2，组合电路C2的延迟；Tsetup，UFF1的建立时间; T，CLKQ的周期。

0 - Tsetup - Tc2 - Tc1 - Tclk2q > 0 - T ;

Tc2 + Tsetup定义为外部输出延迟，DUA在上游。**输出延迟是对应捕获触发器/CK指定的，设捕获时刻为0。延迟是DUA的OUTB到UFF1/D再到UFF1/CK。**

create\_clock -period 15 -waveform {0 7.5} [get\_port CLKQ]

set Tc2 3.9

set Tsetup 1.1

set\_output\_delay -clock CLKQ -max [expr Tc2max + Tsetup] [get\_ports OUTB]

#设置OUTB到UFF1/CK的**最大**延时（不是路径的延迟）： Tc2max是外部逻辑C2的最大延迟，即OUTB到UFF1/D的**最大**延时。但是，为了让UFF1能正确捕获本次数据，本次数据经过Tc2max延迟后还必须相对于CLKQ有效沿提前Tsetup时间到达UFF1/D，所以总的OUTB到UFF1/CK的**最大延迟是**Tc2max + Tsetup（Tc2max时间再向时间轴正方向增加Tsetup）。此和值被set\_output\_delay -max -clock命令设定，出现在8.1.3节时序报告中的output external delay项。

此例，输出最大延时为5ns。理解为芯片内部UFF0的Tclk2q + Tc1是未知而需要约束的，是芯片设计时需要实现的，芯片外部的Tc2max + Tsetup是已知的（由set\_output\_delay -max指明这一个外部条件）。

set\_output\_delay -clock CLKQ -min [expr Tc2min - Thold] [get\_ports OUTB]

#设置OUTB到UFF1/CK的**最小**延时（不是路径的延迟）：Tc2min是外部逻辑C2的最小延迟，即OUTB到UFF1/D的**最小**延时。为了保证本次数据经过最小延迟到达UFF1/D时，不会冲掉上一个数据，因此要求UFF1在发射上一个数据并经过Thold后，本次数据才能到达UFF1/D。所以，总的OUTB到UFF1/CK的**最小延迟是**Tc2min - Thold（Tc2min延迟时间再向时间轴负方向倒退Thold，发射沿比Tc2min还要提前Thold）。此差值被set\_output\_delay -min-clock命令设定，出现在8.2.3节,图8-13时序报告中的output external delay项。

**如何理解set\_input\_delay与set\_output\_delay？**

**依据后面7.6节，时序路径的定义：**

1. **输入端口到输出端口；**
2. **输入端口到触发器或存储器的输入端口（引脚）；**
3. **触发器或存储器的时钟引脚到触发器或存储器的输入端口（引脚）；**

**4）触发器或存储器的时钟引脚到输出端口；**

**首先，set\_input\_delay约束的起始点是上游触发器的时钟引脚，终点是dua的输入端口，满足时序路径的定义；而set\_output\_delay约束的起始点是dua的输出端口，终点是下游触发器的时钟引脚，不满足时序路径的定义（终点不满足）。所以严格来讲set\_output\_delay不是约束时序路径的，而set\_input\_delay碰巧约束了一条时序路径。**

**然而，将set\_input\_delay加上DUA加上set\_output\_delay连贯起来看，时序的起点和终点均是时钟有效沿，即这一连贯的时序对齐的是时钟有效沿。由于时钟有效沿具有唯一性（参考点），因此可用set\_input\_delay与set\_output\_delay所设置的参数来作DUA的建立/保持时间检查。**

**最后，set\_input\_delay与set\_output\_delay均需要指明 -clock参数，这正说明了时序是参考时钟沿的。**

建立时间检查：

-T + Tclk2q + Tc1 + Tc2 + Tsetup < 0时刻

或

0 - Tsetup - Tc2 - Tc1 - Tclk2q > -T时刻

-(Tclk2q + Tc1) > -15+5 = -10ns

Tclk2q + Tc1 < 10ns

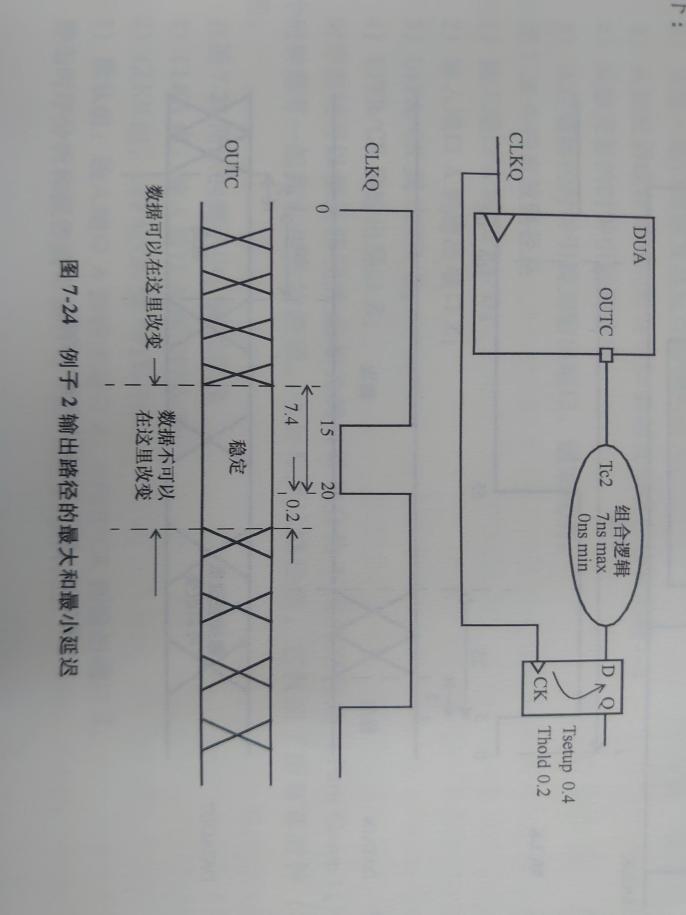
Tc1 < 10 - Tclk2q

保持时间检查：

-T + Tclk2q + Tc1 + Tc2 > -T + Thold

Tclk2q + Tc1 > Thold - Tc2 = 0.2 - 0

又例：



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

set\_output\_delay -clock CLKQ -max [expr 7 + 0.4] [get\_ports OUTC]

# 7ns延时加上 0.4ns延时，才到达参考时钟沿时刻。

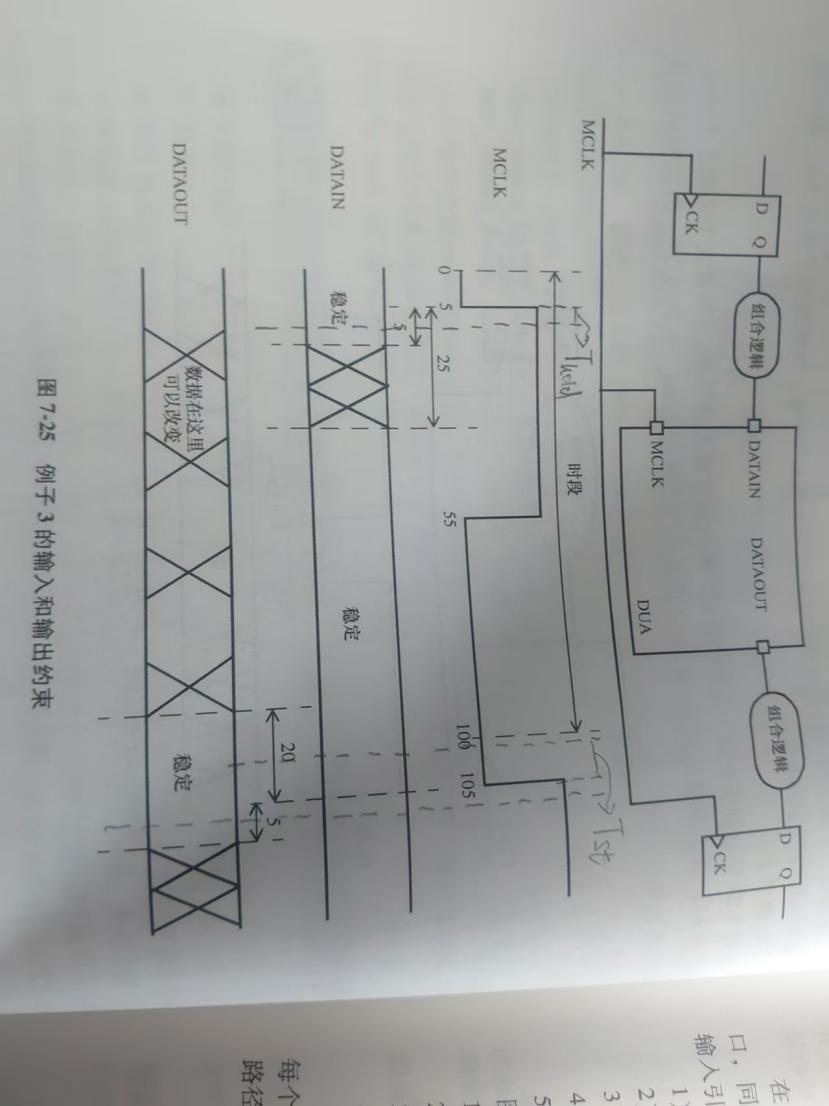
set\_output\_delay -clock CLKQ -min [expr 0 - 0.2] [get\_ports OUTC]

# 0ns延时加上-0.2ns延时（相当于提前），才到达参考时钟沿时刻。

上例中Tclk2q + Tc1出现0值，可考虑为芯片外部时钟路径与（Tclk2q + Tc1）数据路径相当。此例子的数据路径延迟是相对于时钟路径的，但是时钟路径也有长度，也会产生延迟。如果采用绝对时间观念，那么还需要设置时钟路径的延迟值；如果采用相对时间观念，可认为时钟与数据路径的共同延迟部分相互抵消，只考虑两者的差额部分，**是这样吗？**好处就是不需要设置时钟路径的绝对延迟值。相对部分就有正有负。

又例：

DUA有输入DATAIN和MCLK；有输出DATAOUT。



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]

# DATAIN端口输入的数据，相对于时钟有效沿之后的5ns至25ns之间可变化

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

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

# DATAOUT端口输出的数据，相对于时钟有效沿的-20ns至5ns之间稳定

另见后面8.1节、8.2节,建立和保持时间报告。

7.5.1 **输入/输出路径约束对比分析及汇总表**

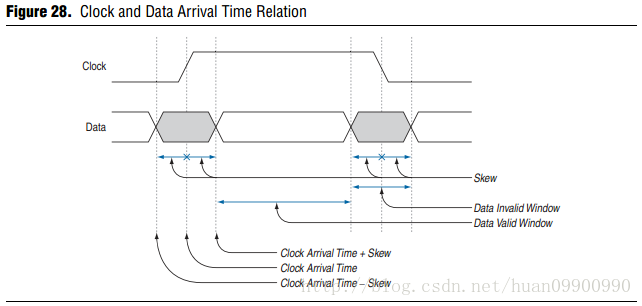
|  |  |  |
| --- | --- | --- |
|  | 建立时间检查(立马) | 保持时间检查（保命） |
| set\_input\_delay  检查点在DUA的输入端口 | 外围电路在上游，DUA在下游。时间关系：  0 + Tclk2q + Tc1-max + Tc2-max + Tsetup < T （T时钟周期）  (Tclk2q + Tc1-max) + Tc2-max + Tsetup < T（capture edge）  设（set\_input\_delay -max）= Tclk2q + Tc1-max  时序报告中参数：input external delay | 外围电路在上游，DUA在下游。时间关系：  0 + Tclk2q + Tc1-min + Tc2-min > 0 + Thold  (Tclk2q + Tc1-min) + Tc2-min - Thold > 0（launch edge）  设（set\_input\_delay -min）= Tclk2q + Tc1-min  时序报告中参数：input external delay |
| set\_output\_delay检查点在DUA的输出端口 | 外围电路在下游，DUA在上游。时间关系：  Tclk2q + Tc1-max + (Tc2-max + Tsetup) < T  TDATAOUT\_max + (Tc2-max + Tsetup) < T  设（set\_output\_delay -max）= Tc2-max + Tsetup  时序报告中参数：output external delay | 外围电路在下游，DUA在上游。时间关系：  Tclk2q + Tc1-min + (Tc2-min - Thold) > 0  TDATAOUT\_min + (Tc2-min - Thold) > 0  设（set\_output\_delay -min）= Tc2-min - Thold  时序报告中参数：output external delay |
| 对于输出路径  DUA输出端口作为检查点 | Data Arrival Time = TDATAOUT\_max  Data Require Time = Tcapture - output external delay  = T - output external delay  Slack = Data Require Time - Data Arrival Time  时间检查地点是DUA的输出端口DATAOUT | Data Arrival Time = TDATAOUT\_min  Data Require Time = Tlaunch - output external delay  = 0 - output external delay  Slack = Data Arrival Time - Data Require Time  时间检查地点是DUA的输出端口DATAOUT |
| 注意！！！ | 建立时间检查必须指明相对于哪一个时钟，参考哪些有效沿：（时间、空间相关）  set\_input\_delay -max -clock 指明时钟，参考上游发射沿及DUA捕获沿；  set\_output\_delay -max -clock 指明时钟，参考DUA发射沿，下游捕获沿； | 保持时间检查必须指明相对于哪一个时钟，参考哪些有效沿：（时间、空间相关）  set\_input\_delay -min -clock 指明时钟，参考上游发射沿及DUA发射沿；  set\_output\_delay -min -clock 指明时钟，参考DUA发射沿及下游发射沿； |

**建立时间检查的记忆表：**

|  |  |  |
| --- | --- | --- |
|  | 建立时间检查(立马) | |
| 基本原则 | tlaunch(0ns，上游) + set\_input\_delay -max < tcapture(Tns，下游) -（Tc2-max + Tsetup）（检查点时DUA输入）  tlaunch(0ns，上游) + (Tclk2q + Tc1-max) < tcapture(Tns，下游) - set\_output\_delay -max（检查点是DUA输出）  0 + Tclk2q + Tc1-max + Tc2-max + Tsetup < T（检查点是下游触发器的时钟端，下游触发器捕获沿）  （Tclk2q + Tc1-max）+（Tc2-max + Tsetup）< T  （set\_input\_delay -max）+（set\_output\_delay -max）< T （记忆公式形式上同保持时间检查，不是实际公式） | |
| 按IO接口分类 | set\_input\_delay -max | set\_output\_delay -max |
| 意义 | 上游输出触发器到DUA输入端之间，本次数据可变区间的右边界的最大值；  上游发射沿后，本次数据最晚（完全）到达DUA数据输入端的时刻。此时刻相对于上游发射沿滞后的时间就是set\_input\_delay -max，其大小，本质上由上游电路决定； | DUA输出端到下游输入触发器之间本次数据稳定区间的左边界的最小值（要求值）；  依据下游电路参数要求，相对于下游捕获沿前，本次数据数据必须完全到达DUA输出端的最早提前量（最大值）就是set\_output\_delay -max，其大小，本质上由下游电路决定；  本次数据从DUA内部发出，如果最晚（完全）出现在DUA数据输出端的时刻小（更早）于上述要求时刻，或者说相对于下游捕获沿有更大的提前量，则本次数据能被后续电路捕获，否则失败。 |
| 细细品 | 假设上游触发器组在同一参考发射沿向DUA发射一组数据。各比特数据分别由DUA数据端接收，并被DUA内部触发器组捕获。考虑某些bit位可能不会影响DUA捕获结果，那么这些bit位到达DUA输入端时，就可以不按时序要求，也就是说，即使这些bit位晚于要求时间，也不会影响DUA对本次数据捕获后的结果。  而set\_input\_delay -max统一要求了所有数据到达DUA输入端的最晚时限，显然是一个保守的要求。 | 同样，set\_output\_delay -max依据下游电路的延迟及保持参数，统一要求本次数据到达DUA输出端的时限，即距离下游捕获沿数据保持稳定的最早时刻。这也是一个保守的要求。  此外可以理解，数据每经过一级电路，就会产生延迟，同时数据也应该发生了变化。数据的变化是在时间和空间上同时发生的。 |
| 依据DUA有效时钟沿前后数据稳定的图像设置 | （set\_input\_delay -max）  = 上游发射沿后本次数据最晚到达时刻 - 0ns  （优选，因为这是依据上游参数决定的时刻）  = DUA捕获沿前数据稳定的最早时刻 - 0ns  （劣选，本次数据可能在此最早时刻前稳定）  = T -（T - DUA捕获沿前数据稳定的最早时刻）  （劣选，原因同上，意义不如优选准确） | （set\_output\_delay -max）  = T - 下游捕获沿前DUA输出端数据稳定的最早时刻  （优选，这是依据下游参数决定的时刻，是原定义）  = T - DUA发射沿后数据输出端变化的最晚时刻  （劣选，DUA输出端数据还可能允许更晚到达）  = T -（DUA数据输出端变化的最晚时刻 - 0）  （劣选，原因同上，意义不如优选准确） |
| 依据DUA有效时钟沿前后数据可变的图像设置 | （set\_input\_delay -max） = +skew  上游发射沿后数据可以变化的正偏斜，即+skew | （set\_output\_delay -max） = T - (+skew)  T - （发射沿后数据可变的正偏斜） |

**保持时间检查的记忆表：**

|  |  |  |
| --- | --- | --- |
|  | 保持时间检查(保命) | |
| 基本原则 | 0 + Tclk2q + Tc1-min + Tc2-min > 0 + Thold （检查点是下游触发器D端）  0 + Tclk2q + Tc1-min > 0 + Thold - Tc2-min （检查点是数据IO端，左侧延时晚于右侧0ns之后的Thold - Tc2-min）  **0 + Tclk2q + Tc1-min > 0 -（Tc2-min - Thold）（检查点是数据IO端，左侧延时晚于右侧0ns之前的Tc2-min - Thold，STA使用此公式）**  tlaunch(0ns，上游) + set\_input\_delay -min > tlaunch(0ns，下游) -（Tc2-min - Thold）（检查点时DUA输入）  tlaunch(0ns，上游) + Tclk2q + Tc1-min > tlaunch(0ns，下游) - set\_output\_delay -min（检查点是DUA输出）  Tclk2q + Tc1-min + Tc2-min - Thold > 0（检查点是下游触发器的时钟端，下游触发器发射沿）  （Tclk2q + Tc1-min）+（Tc2-min - Thold）> 0 （记忆公式）  （set\_input\_delay -min）+（set\_output\_delay -min）> 0 （记忆公式形式上与建立时间检查相同，但不是实际公式） | |
| 按IO接口分类 | set\_input\_delay -min | set\_output\_delay -min |
| 意义 | 上游输出触发器到DUA输入端之间，本次数据可变区间的左边界的最小值；  上游发射沿后，本次数据最早（首次）出现在DUA数据输入端的时刻。此时刻相对于上游发射沿滞后的时间就是set\_input\_delay -min，其大小，本质上由上游电路决定； | DUA输出端到下游输入触发器之间上次数据稳定区间的右边界的最大值（要求值）；  依据下游电路参数要求，向对于下游发射沿前，本次数据必须首次到达DUA输出端的最晚提前量（最小值）就是set\_output\_delay -min，其大小，本质上由下游电路决定；  本次数据从DUA内部发出，如果最早（首次）出现在DUA数据输出端的时刻大（更晚）于上述要求时刻，或者说相对于下游发射沿有更小的提前量，则本次数据不会影响到下游触发器发射上次数据，否则将导致下游触发器数据发射混乱；  同时，本次数据首次到达DUA输出端之前，DUA输出端必须保持上次数据稳定直至上述要求时刻，届时上次数据才能被下游触发器成功发射； |
| 细细品 | 与建立时间检查的约束情况同样保守吗？ | （set\_output\_delay -min）与Thold具有相似的意义：（set\_output\_delay -min）超前于下游发射沿，而Thold则滞后于下游发射沿；（set\_output\_delay -min）的检查点是DUA输出端，而Thold的检查点是下游触发器的D输入端；无论检查点在何处，均要求首次出现本次数据的时刻要晚于（set\_output\_delay -min）或Thold要求的时刻。  因将（Tc2-min - Thold）定义为（set\_output\_delay -min），则：  （Tc2-min - Thold）> 0：正值，数据稳定区间右边界提前于下游发射沿时刻；  （Tc2-min - Thold）= 0：临界条件，右边界对齐于下游发射沿时刻；  （Tc2-min - Thold）< 0：负值，右边界提前于下游发射沿时刻，具有Thold意义 |
| 依据DUA有效时钟沿前后数据稳定的图像设置 | （set\_input\_delay -min）  = 上游发射沿后，DUA数据输入端出现本次数据的最早时刻 - 0ns  上游发射沿后：  input\_delay\_min < DUA输入端数据可变时间 < input\_delay\_max | （set\_output\_delay -max）=  在DUA数据输出端，下游发射沿时刻 - 上次数据稳定区间的右边界（要求值）。  如果差值为负值，表示稳定区间右边界滞后于下游发射沿时刻；  如果差值为0，稳定区间右边界对齐于下游发射沿；如果差值为正值，稳定区间右边界提前于发射沿时刻。  下游发射沿前：（因为是0 - set\_output\_delay -min）  (0 - output\_delay\_max) < DUA输出端数据稳定要求< (0 - output\_delay\_min) |
| 悟了 | 从上游电路到DUA，时间轴正方向为参考，有input流程：  上游本次数据发射沿（与此同时，DUA发射上次数据） -> input\_delay\_min（本次数据首次到来） -> 数据陆续到来 -> input\_delay\_max （本次数据完全到来并稳定） -> DUA本次捕获沿（也是DUA本次发射沿）；  input\_delay\_min 对标 tco\_min，两者检查点不同；  Input\_delay\_max 对标tco\_max，两者检查点不同； | 从DUA到下游电路，时间轴负方向为参考，有output流程：  上次数据要求最晚稳定时刻（下游thold） -> 下游上次数据发射沿（也是上次数据捕获沿）-> 上次数据要求最早稳定时刻（0 - output\_delay\_max） -> 上次数据实际稳定 -> 上次数据到来 -> 上次首数据实际到来 -> 上游thold -> 上游上次数据发射沿  为了将上述output流程与input流程相统一，引入output\_delay\_min，有：  下游上次发射沿 -> 上次数据要求最晚稳定时刻（0 - output\_delay\_min）上次数据已经稳定-> 上次数据要求最早稳定时刻 (0 - output\_delay\_max) -> 上次数据实际稳定时刻 -> 上次数据陆续到来 -> 上次首数据实际到来时刻，上游上次发射沿时刻（两时刻不分先后）-> **上上次**数据要求最晚稳定时刻；  output\_delay\_max 对标 tsetup，两者的检查点不同，圣经中提到了这一点；  output\_delay\_min 对标 thold，两者的检查点不同，圣经中未明确提出； |
| 依据有效时钟沿前后数据可变的图像设置 | （set\_input\_delay -min） = -skew  发射沿前数据可以变化的最早时刻，即负偏斜(-skew) | （set\_output\_delay -min） = (0 - (-skew))  发射沿前数据可以变化的区间，|-skew| 负偏斜的绝对值 |



7.5.2依据有效时钟沿前后数据可变的图像设置

set unit\_interval [expr T/2]

create\_clock -name Clock -period p -waveform {0, p/2} [get\_ports Clock]

set\_input\_delay -min [expr -skew - 0] -clock Clock [get\_ports Data\_in\*]

set\_input\_delay -min [expr -skew - 0] -clock Clock [get\_ports Data\_in\*] -clock\_fall

set\_input\_delay -max [expr +skew - 0] -clock Clock [get\_ports Data\_in\*]

set\_input\_delay -max [expr +skew - 0] -clock Clock [get\_ports Data\_in\*] -clock\_fall

set\_output\_delay -min [expr 0 - (-skew)] -clock Clock [get\_ports Data\_out\*]

set\_output\_delay -min [expr 0 - (-skew)] -clock Clock [get\_ports Data\_out\*] \ -clock\_fall

set\_output\_delay -max [expr unit\_interval - (+skew)] -clock Clock [get\_ports \ Data\_out\*]

set\_output\_delay -max [expr unit\_interval - (+skew)] -clock Clock [get\_ports \ Data\_out\*] -clock\_fall

## 7.6、时序路径组

每条路径都有一个起点和一个终点，STA用有效起点和终点作为路径记录。

起点：输入端口（引脚）、同步器件时钟引脚；

终点：输出端口（引脚）、同步器件数据输入引脚。

路径可以是：

1. 输入端口到输出端口；
2. 输入端口到触发器或存储器的输入端口（引脚）；
3. 触发器或存储器的时钟引脚到触发器或存储器的输入端口（引脚）；
4. 触发器或存储器的时钟引脚到输出端口；

时许路径可依据路径终点相关的时钟被分为不同的路径组（Path Group）。每一个时钟都会有一组与之相关的路径。默认路径组包括了所以非时钟（异步）路径。

路径是空间概念，时序是时间概念。最大/最小时序路径指从时间的角度来衡量，空间中存在的延迟最大或最小的路径，而不是路径上含有逻辑单元的多少，这在第二章第8节有说明。显然也应该有类似最大逻辑单元路径或最小逻辑单元路径纯空间概念定义。

## 7.7、外部属性建模

为了得到DUA的IO的精确时序，除了create\_clock、set\_input\_delay、set\_output\_delay外，还需要对DUA上下游电路的驱动能力及负载情况建模。

7.7.1、set\_input\_transition

用于上游源电路建模，直接指定DUA输入端口的转换时间/率。input\_transition是计算单元延迟的参量，见前述。

set\_input\_transition 0.85 [get\_ports INPC] # 端口INPC的输入转换时间是850ps

set\_input\_transition 0.6 [all\_ports]

选项-min、-max可用来指定最小、最大转换时间

7.7.2、set\_load

约束DUA输出端口的负载。

set\_load 5 [get\_ports OUTX] # 输出端口OUTX的负载为5pf

Set\_load 25 [all\_outputs]

set\_load -pin\_load 0.007 [get\_ports OUT]

# 指定OUT端口的引脚的负载为0.007pf，-pin\_load是默认选项

# -wire\_load指定端口的线负载，线负载包括该输出端口的pin\_load、网络连线电容和下游电路的输入电容。

## 7.8、设计规则检查

用于全局STA检查的设置

1. set\_max\_transition

set\_max\_transition 0.6 IOBANK

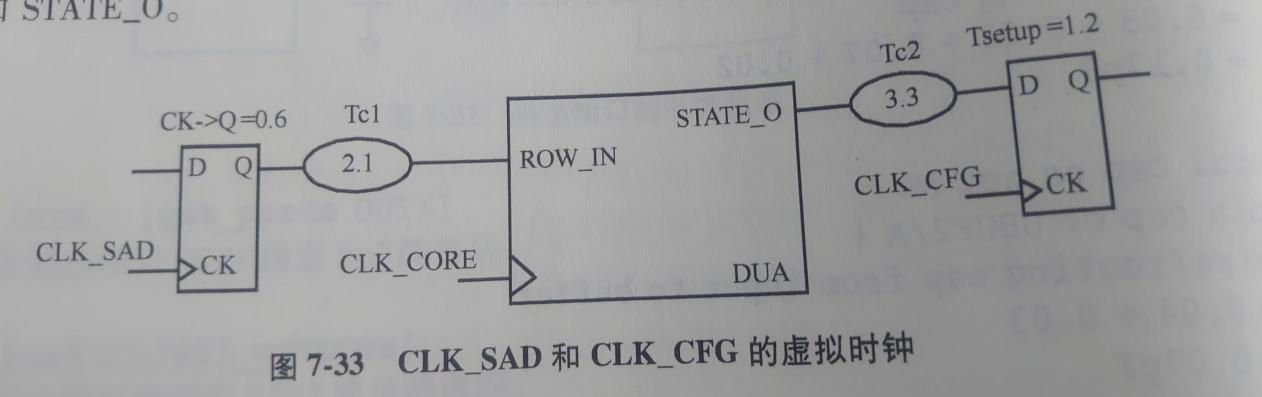
# IOBANK的input transition time 是0.6ps

1. set\_max\_capacitance

set\_max\_capacitance 0.5 [current\_design] # 当前设计的wire\_load为0.5pf

## 7.9、虚拟时钟

虚拟时钟（Virtual Clocks）是一个存在的时钟，但是于设计的任何引脚或者端口都不相关。常被用于参考时钟，指定相对于时钟的输入输出延迟。

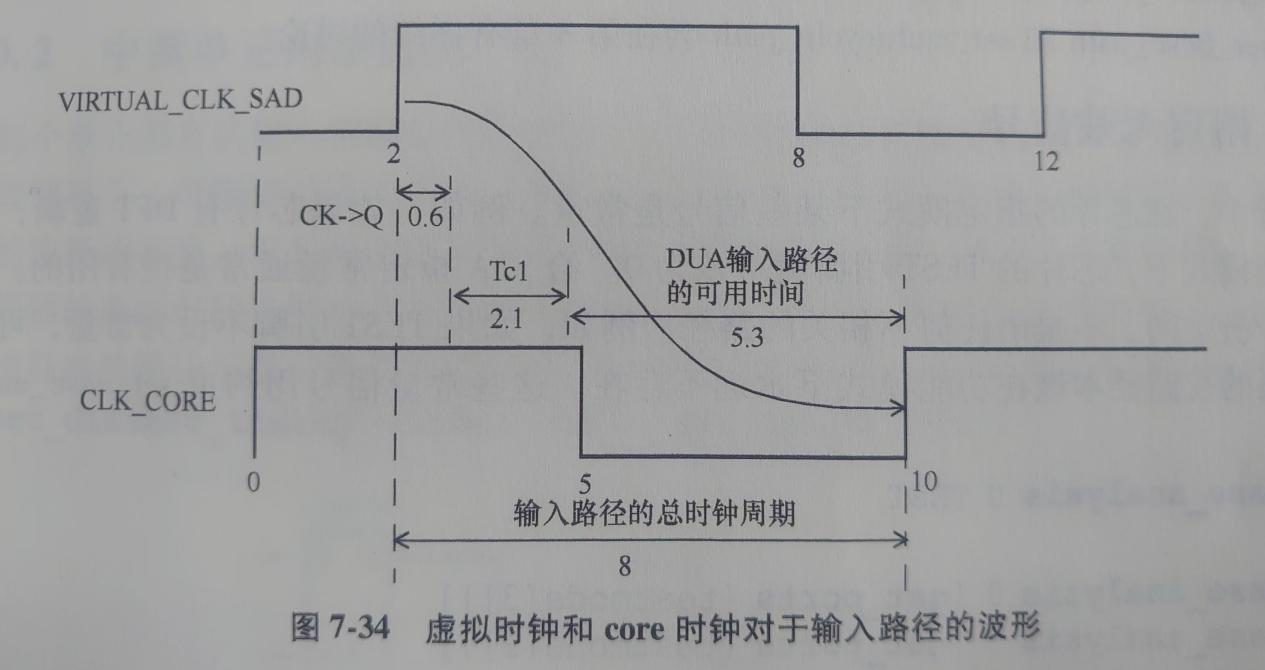


create\_clock -name VIRTUAL\_CLK\_SAD -period 10 -waveform {2 8}

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

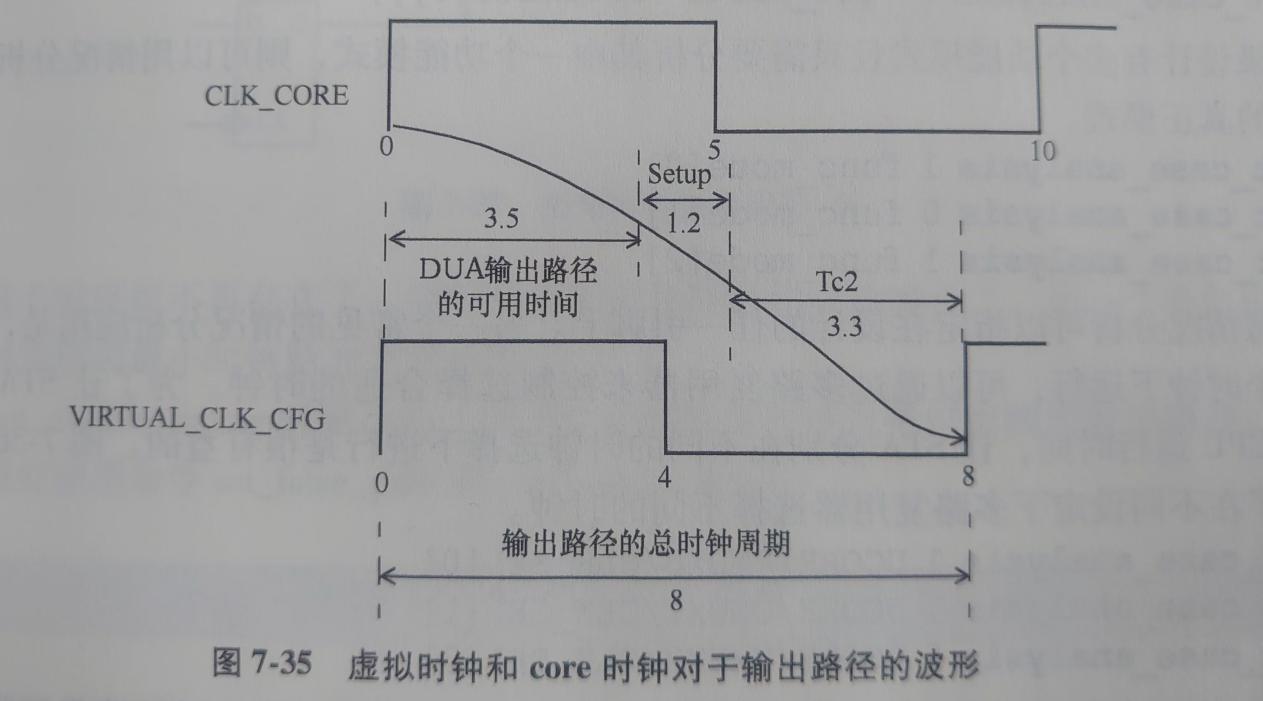
create\_clock -period 10 [get\_ports CLK\_CORE]

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



由上图，DUA的输入路径被约束为5.3ns或更少。

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



由上图，DUA的输出路径被约束为3.5ns或更少。

## 7.10、完善时序分析

4个常用命令：

1. set\_case\_analysis 在单元输入引脚或输入端口指定常量
2. set\_disable\_timing 中断单元时序弧
3. set\_false\_path 设置伪路径
4. set\_multicycle\_path 设置多时钟周期路径

set\_case\_analysis 0 TEST # 设置TEST信号为0

set\_disable\_timing -from s -to z [get\_cells UMX0]

# 设置单元UMX0 从s端口到z端口，不进行实现分析

## 7.11、点对点约束

set\_min\_delay

set\_max\_delay

约束点到点路径的延迟。该约束将覆盖路径上的任何默认单或多周期时序约束。

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

# UFF2/Q -to UFF3/D的最大延迟为0.6ns

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

#时钟域SYS\_CLK到时钟域CFG\_CLK的最小延迟为400ps

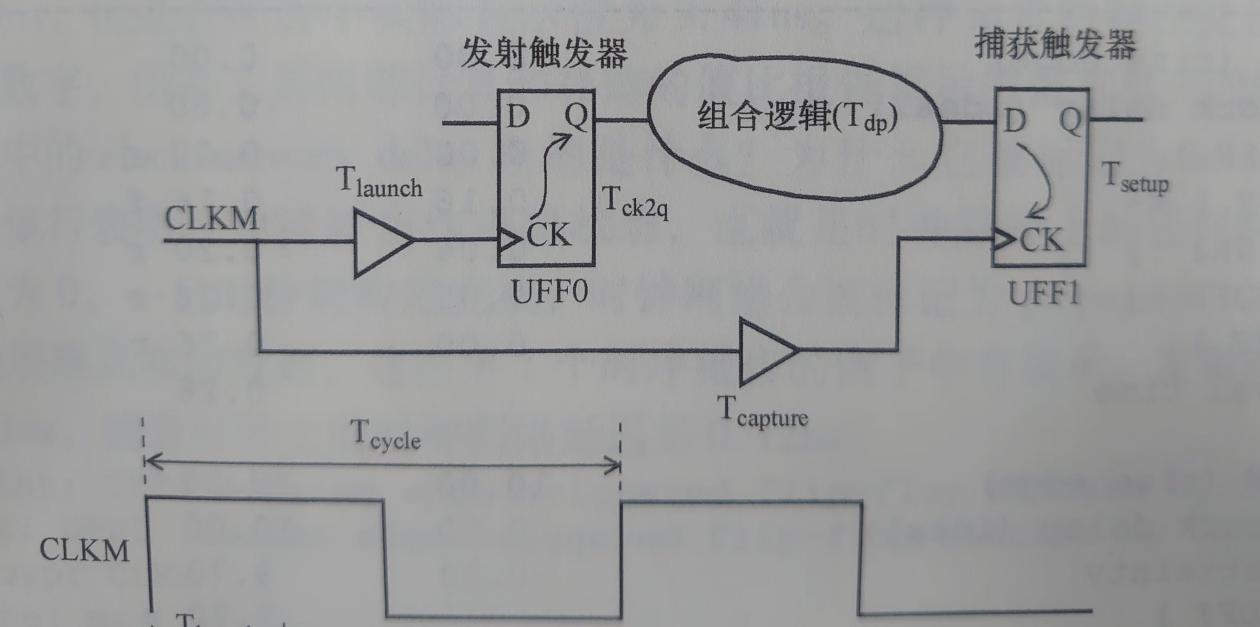
## 7.12、路径分割

将一条时序路径切割为更小的路径段后，进行分段时序路径约束。分段路径也是时序路径。

# **时序验证**

时序验证的目的是穷尽验证DUA的时序。建立时间检查和保持时间检查是两个重要的检查。

## 8.1、建立时间检查

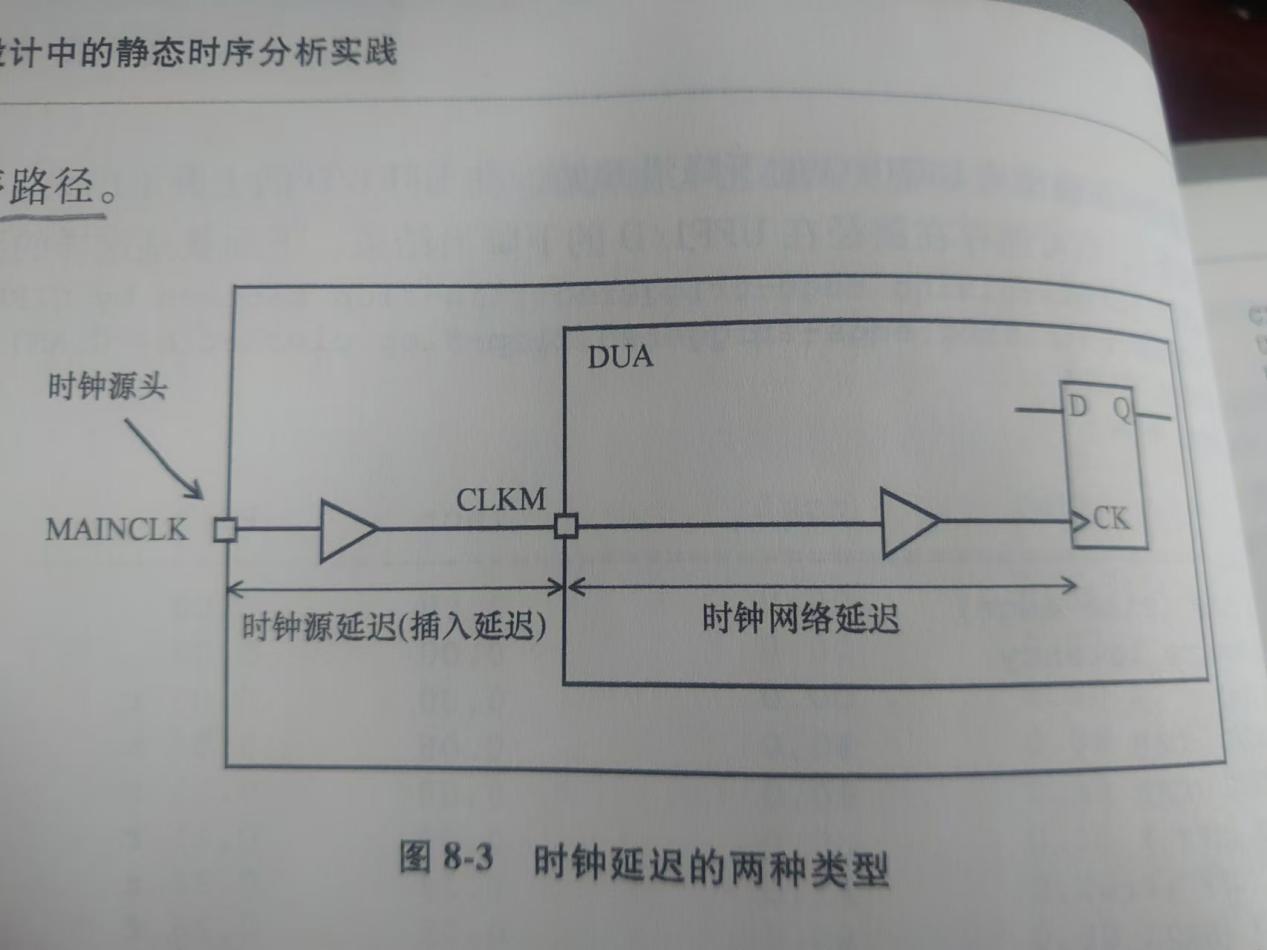


Tlaunch + Tck2q + Tdp < Tcapture + Tcycle - Tsetup

Tlaunch是发射触发器UFF0的时钟树延迟；Tcapture是捕获触发器的时钟树延迟

8.1.1、触发器到触发器路径

源时钟输入的延迟设置（原理见第七章）



1. 时钟从它的源头传播到DUA中时钟定义点所需时间，也叫插入延迟。

set\_clock\_latency -source -rise 0.7 [get\_clocks CLKM]

set\_clock\_latency -source -rise 0.65 [get\_clocks CLKM]

源延迟不影响内部的具有相同发射时钟和捕获时钟的路径，但是源延迟影响穿过DUA输入端和输出端的时序路径。

1. 如果没有-source选项，set\_clock\_latency定义时钟的网络延迟，也就是从DUA的时钟定义点到触发器的时钟引脚间的延迟。

3）时钟网络延迟是用来在时钟树构建完成前，给时钟路径延迟建模，也就是在时钟树综合前。一旦时钟树构建完成且被标记为传播状态（Propagated），这个时钟网络延迟就无效了。

时钟缓存单元的延迟计算（详见3、4章）

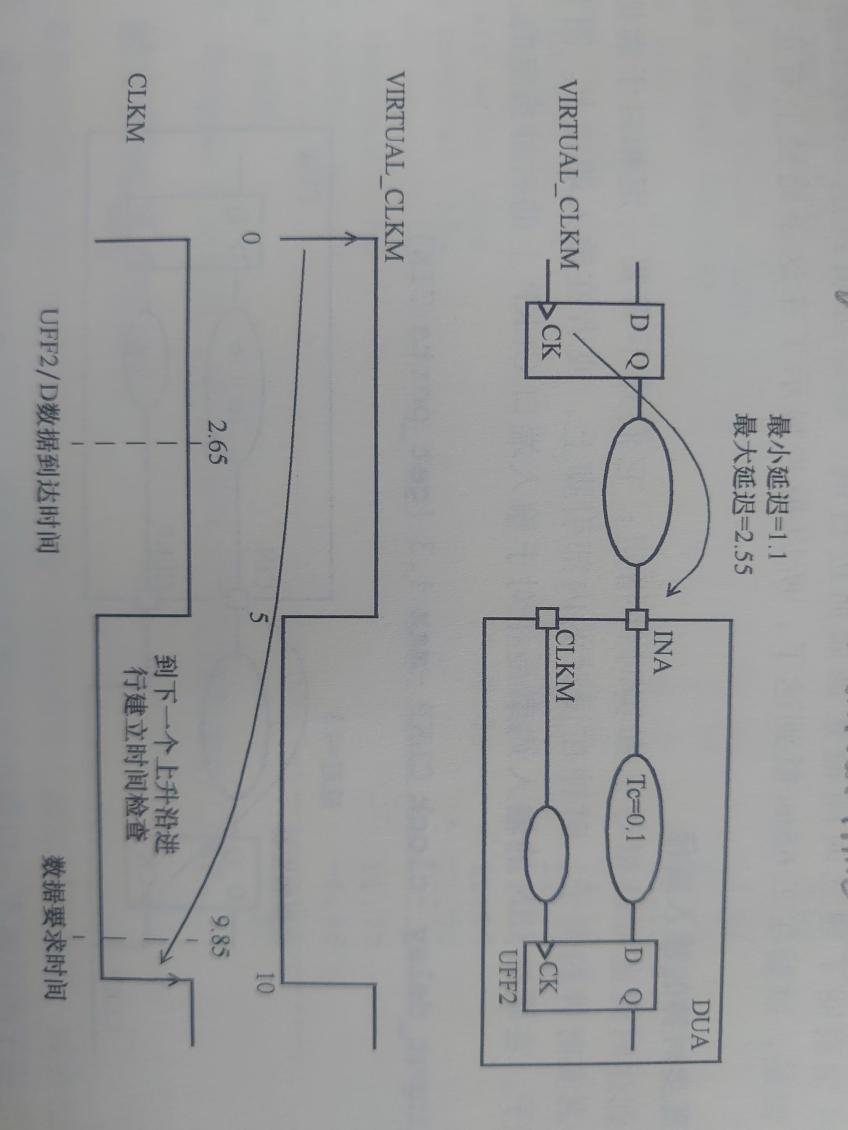
时钟缓存单元的延迟由输入转换时间和输出负载计算。

set\_input\_transition -rise 0.3 [get\_ports CLKM]

set\_input\_transition -fall 0.45 [get\_ports CLKM]

Set\_load xxx [get\_ports C\_buffer]

8.1.2、输入到触发器的路径



create\_clock -name VIRTUAL\_CLKM -period 10 -waveform {0 5}

VIRTUAL\_CLKM是虚拟参考时钟，无关联端口。用于输入INA引脚的参考。

另有外部虚拟触发器的时钟输入也是VIRTUAL\_CLKM。

虚拟触发器时钟引脚到INA的最大数据延迟设为2.55ns。

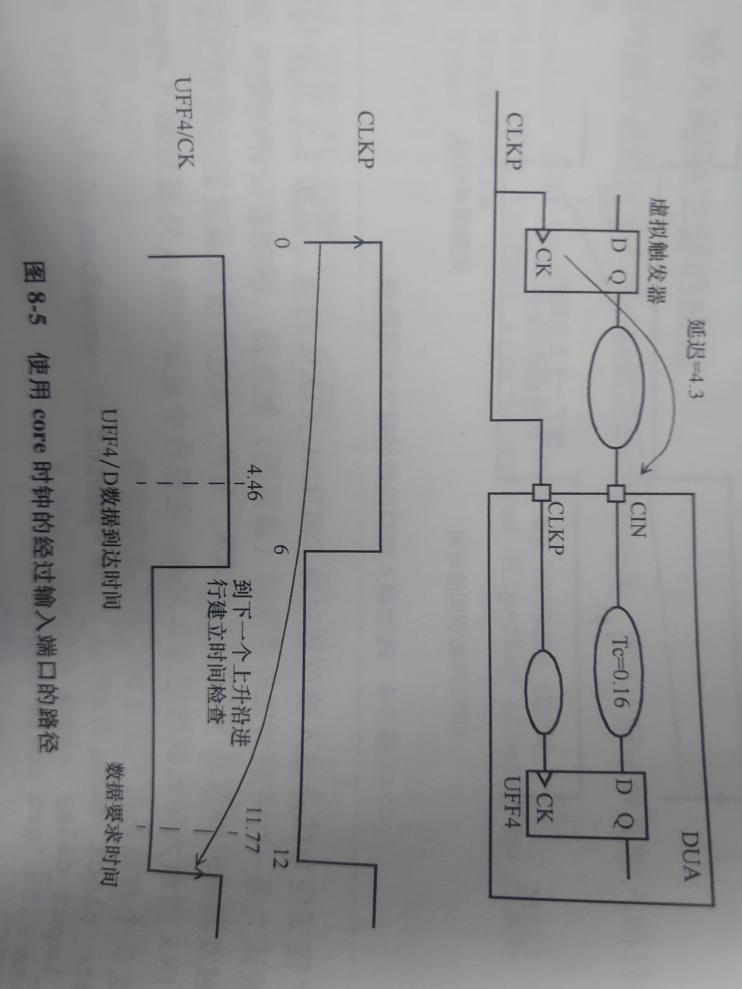
set\_input\_delay 2.55 -clock VIRTUAL\_CLKM -max [get\_ports INA]

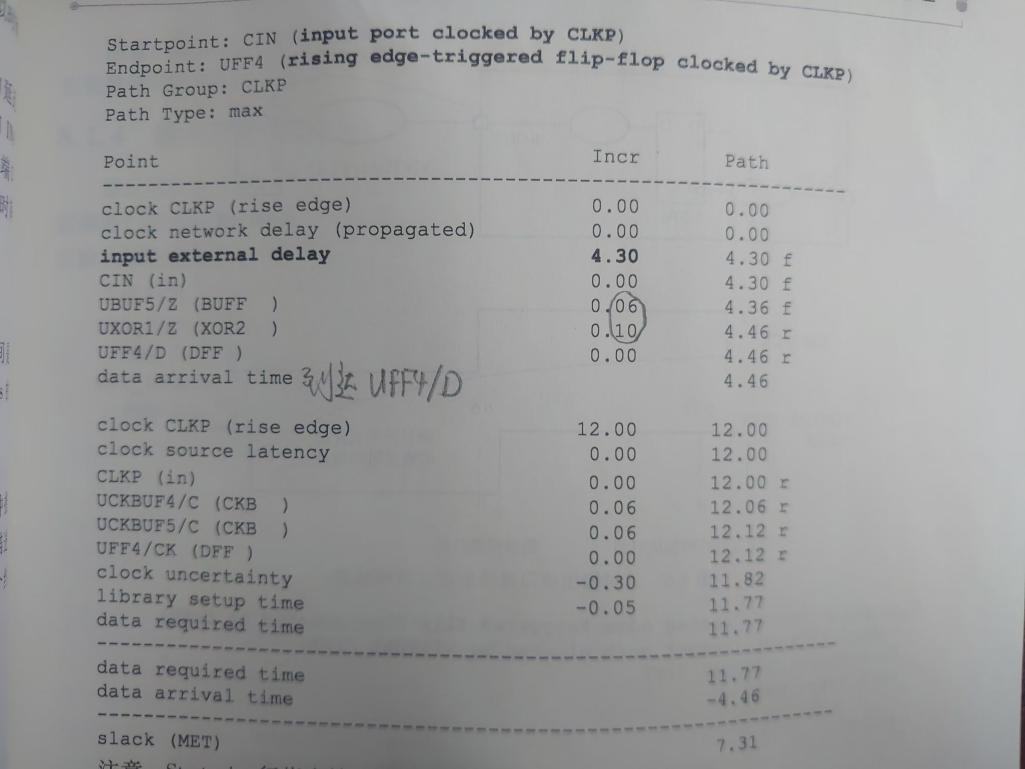
设输入端口（INA）的后接单元是BUFF，则通过单元参数，计算单元延迟。

set\_driving\_cell -lib\_cell BUFF -library lib013wc [get\_ports INA]

又例：

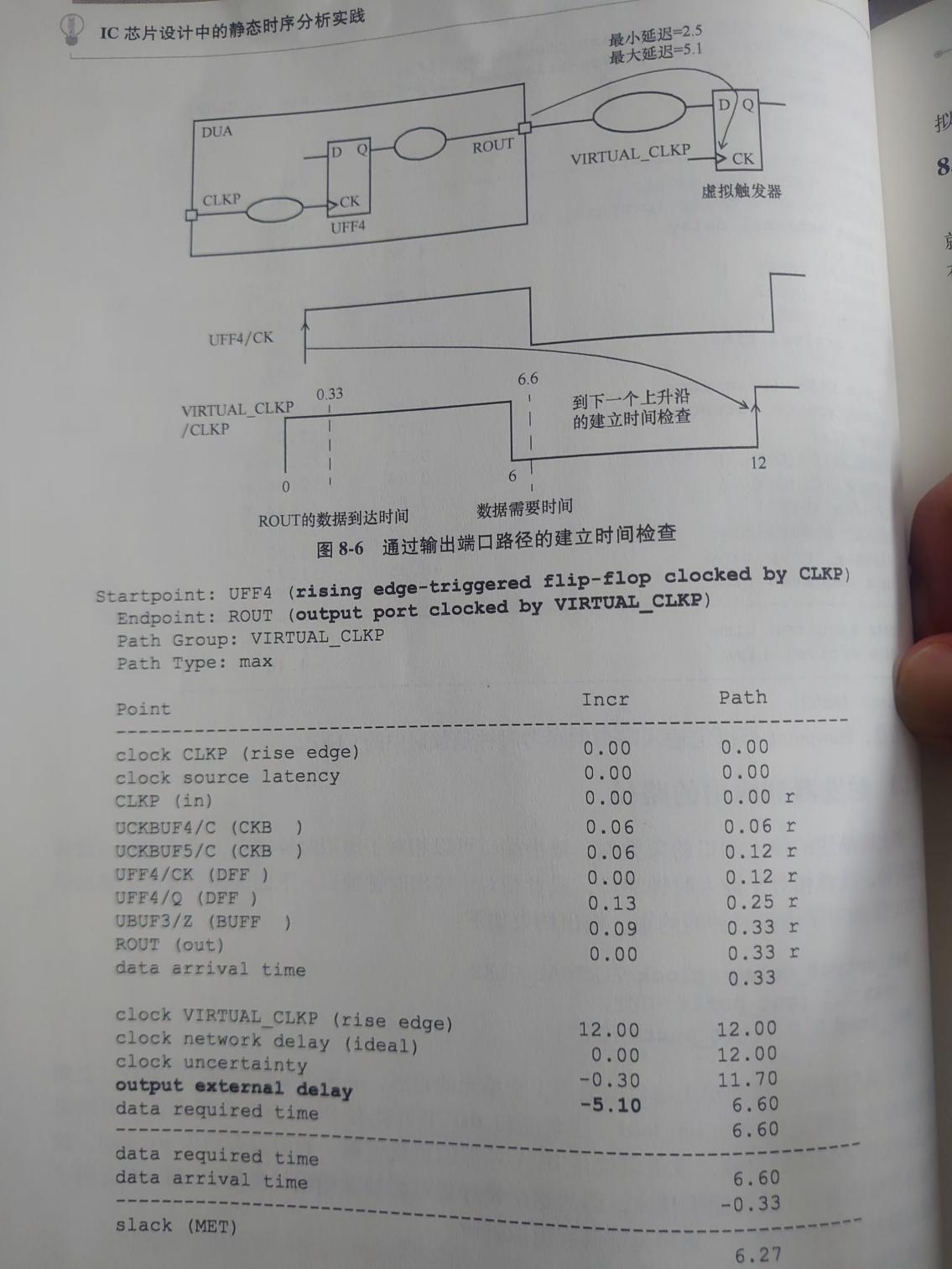
set\_input\_delay -clock CLKP - max 4.3 [get\_ports CIN]





8.1.3、触发器到输出的路径

输出端口可以相对于虚拟时钟、内部设计时钟、输入时钟端口或输出时钟端口进行约束。



set\_output\_delay 5.1 -clock VIRTUAL\_CLKP -max [get\_ports ROUT]

set\_load 0.02 [get\_ports ROUT]

输出延迟显示为output external delay，它的行为类似于虚拟触发器所需的建立时间。

8.1.4、输入到输出的路径

设计可以有输入端口到输出端口的组合逻辑路径，该路径可以被约束和时序分析。

相对于虚拟时钟约束：

create\_clock -name VIRTUAL\_CLKM -period 10 -waveform {0 5}

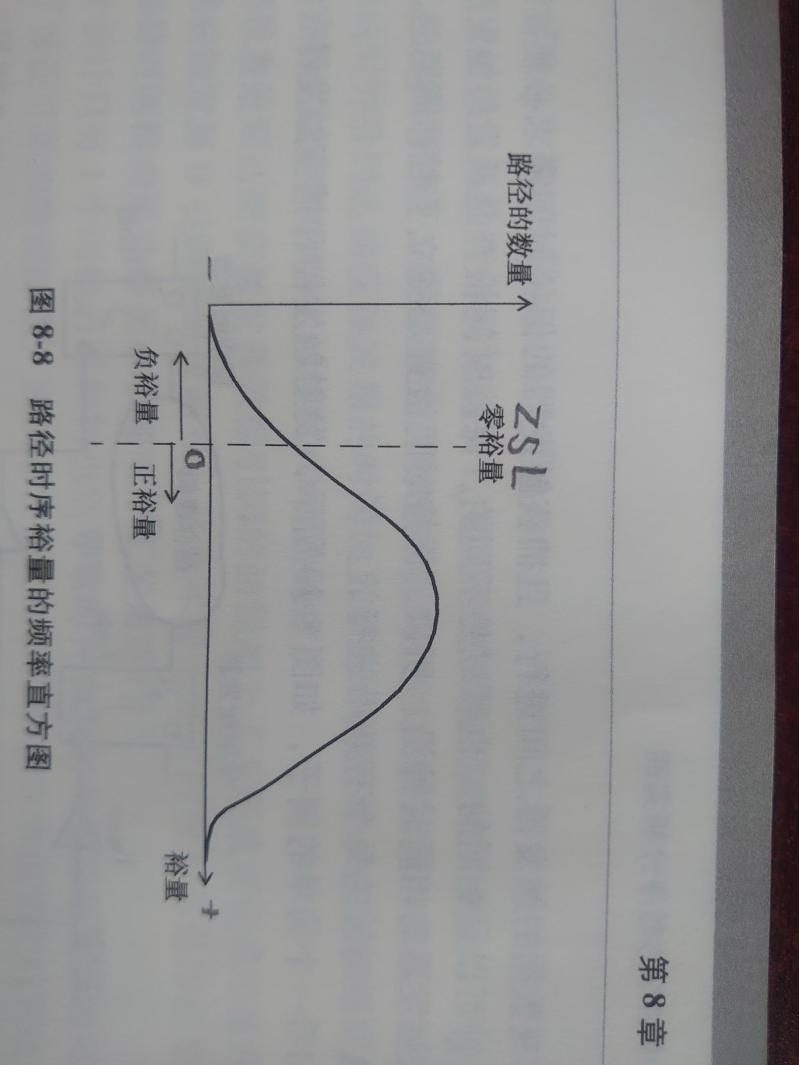
set\_input\_delay 3.6 -clock VIRTUAL\_CLKM -max [get\_ports INB]

set\_input\_delay 1.8 -clock VIRTUAL\_CLKM -min [get\_ports INB]

set\_output\_delay 5.8 -clock VIRTUAL\_CLKM -max [get\_ports POUT]

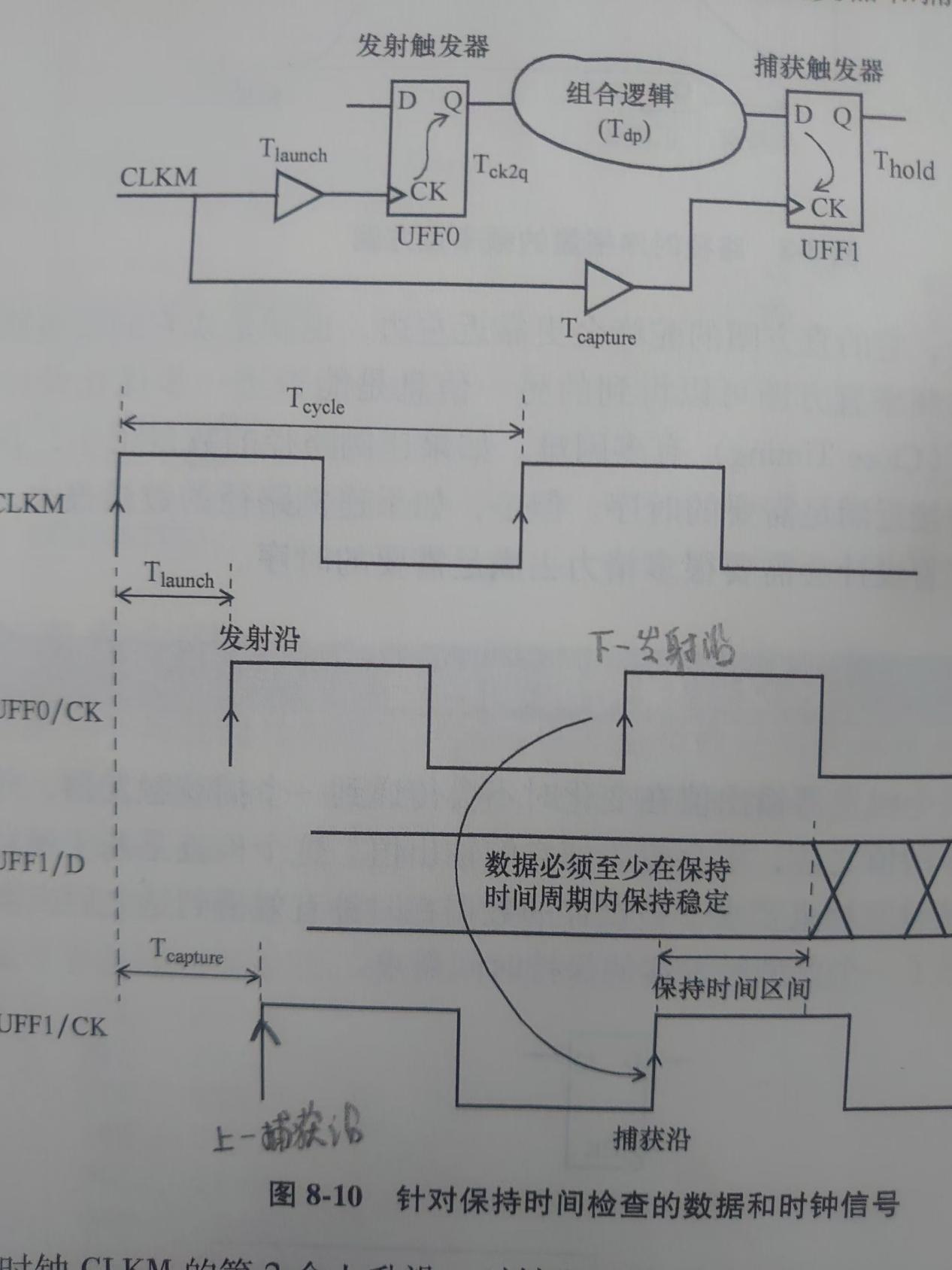
set\_output\_delay 3.2 -clock VIRTUAL\_CLKM -min [get\_ports POUT]

8.1.5 频率直方图



上图为典型设计的建立时间裕量和路径数量的频数直方图。如果设计没有被优化，零裕量线（Zero Slack Line）会更靠右，如果设计被优化了，那么ZSL会更靠左。

## 8.2、保持时间检查



Tlaunch + Tck2q + Tdp > Tcapture + Thold 或

Tlaunch + T + Tck2q + Tdp > Tcapture + T + Thold

8.2.1、触发器到触发器路径

时序报告中，建立时间裕量 = 建立数据需要时间 - 最大数据达到时间

保持时间裕量 = 最小数据到达时间 - 保持数据需要时间

上述四个时间按时间轴排列：先**本次**数据最大到达时间，然后**本次**数据建立需要时间，然后（时钟周期+**本次**数据保持需要时间），最后（时钟周期+**下次**数据最小到达时间）。

也可以表述为：先**上次**数据保持需要时间，然后**本次**数据最小到达时间，然后**本次**数据最大到达时间，最后**本次**数据建立需要时间。

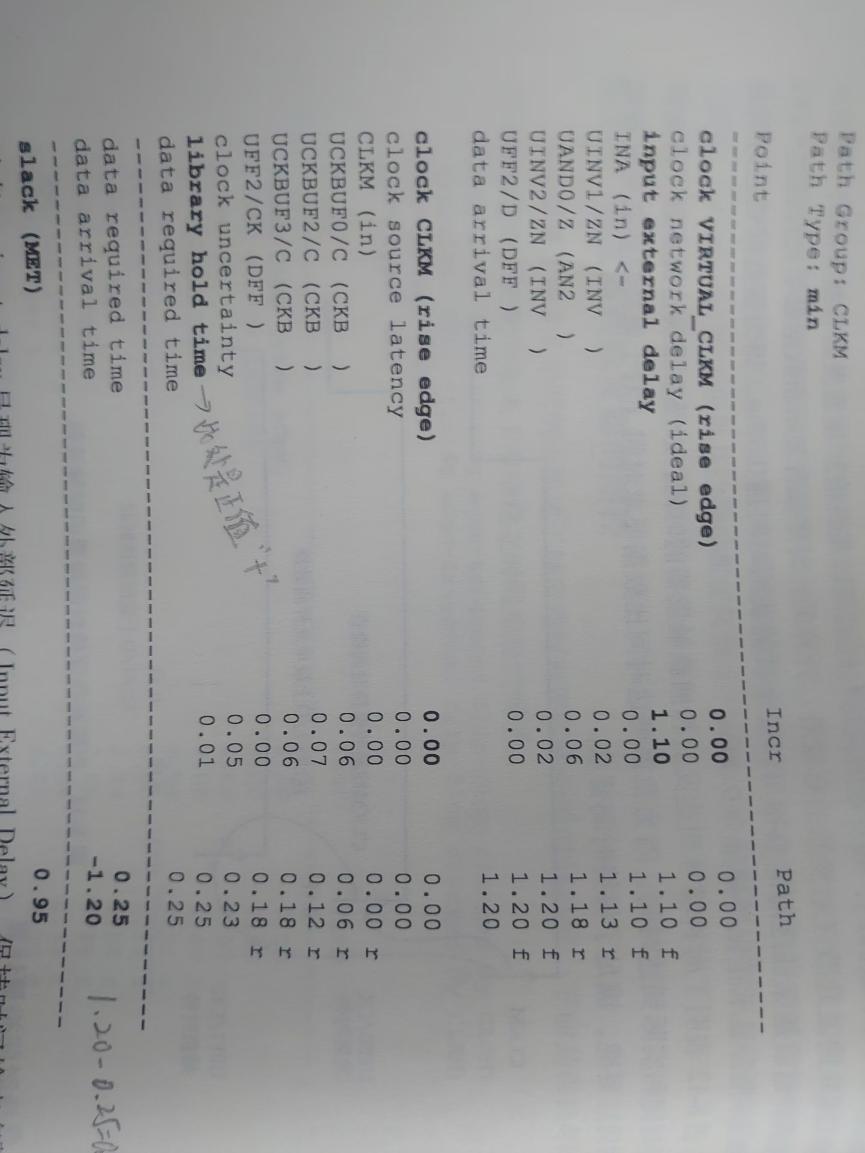
对于（触发器到触发器路径）以及（输入到触发器路径）这两种情况，数据从发射触发器出发，经延迟后到达捕获触发器。时序报告中data arrival time与data require time的检查点通常在捕获触发器的数据输入引脚。

对于触发器到输出的情况，时序报告中data arrival time与data require time的检查点通常在DUA的输出引脚。

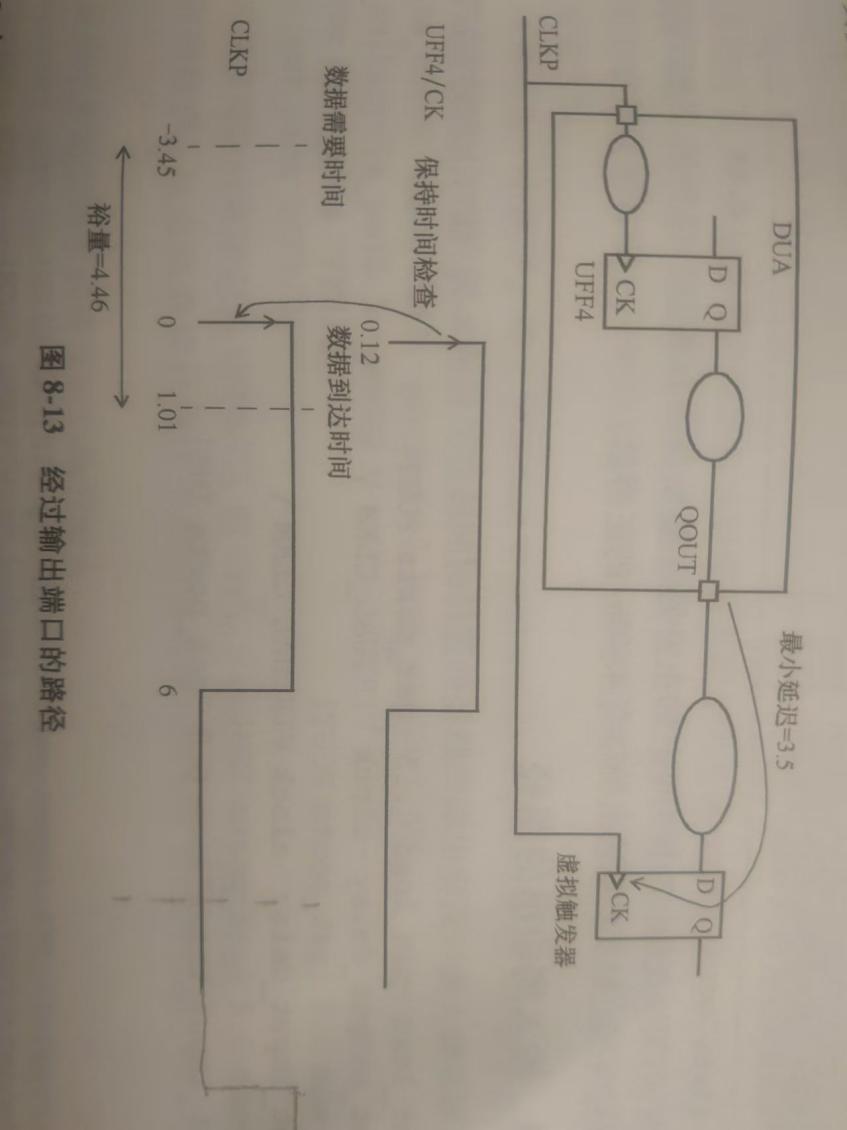
8.2.2、输入到触发器路径

使用虚拟时钟在输入端口上指定最小延迟：

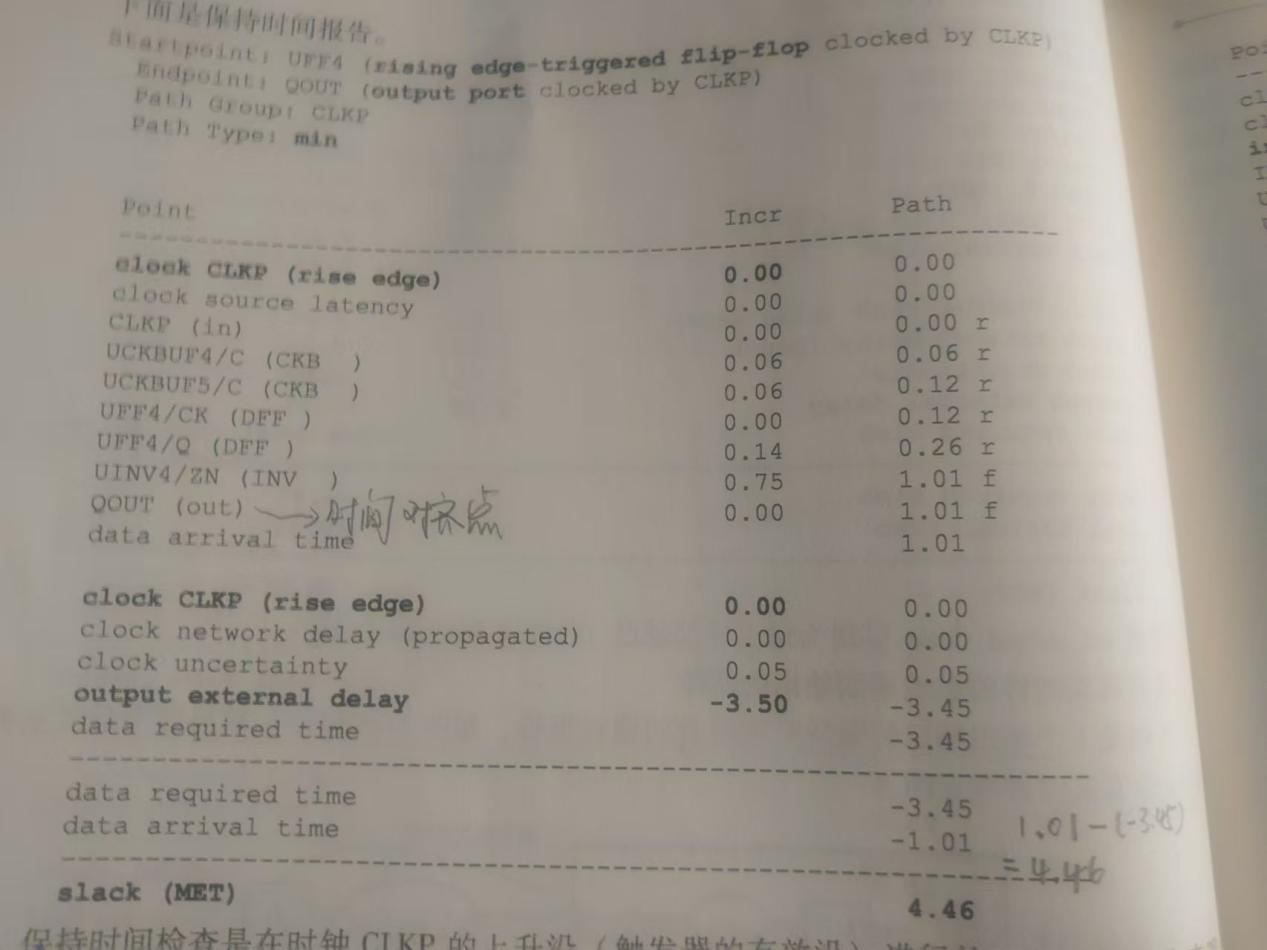
set\_input\_delay -clock VIRTUAL\_CLKM -min 1.1 [get\_ports INA]



8.2.3、触发器到输出路径



CLKP有效沿到达UFF4/CK的时间是0.12ns；CLKP有效沿到DUA/QOUT的路径延迟是1.01ns。见下面的报告：



set\_output\_delay -clock CLKP -min 3.5 [get\_ports QOUT]

对于下游虚拟触发器捕获沿(时刻0)来说，clock uncertainty延迟0.05，output external delay倒退3.5。Hole slack = 1.01 - (0.05 - 3.5) = 4.46

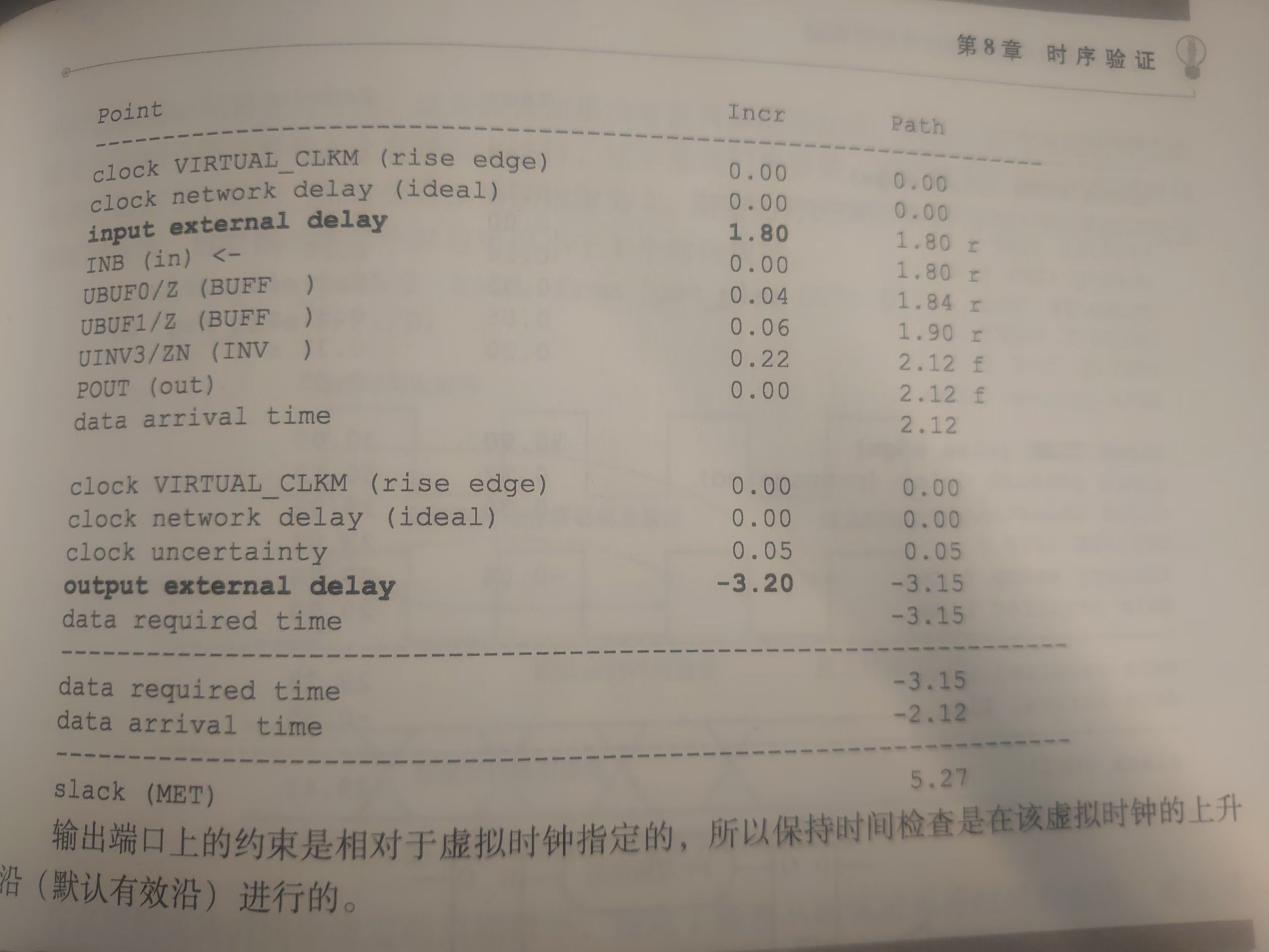
8.2.4、输入到输出的路径

set\_output\_delay -clock VIRTUAL\_CLKM -min 3.2 [get\_ports POUT]

set\_input\_delay -clock VIRTUAL\_CLKM -min 1.8 [get\_ports PIN]

set\_load -pin\_load 0.15 [get\_ports POUT]

set\_input\_transition 0.8 [get\_ports PIN]



Hole slack = 2.12 - (-3.15) = 5.27

8.2.5、对8.1节和8.2节的总结

1）对于芯片内部触发器到触发器的建立时间检查和保持时间检查，参见8.1.1和8.1.2

2）已知外围电路在上游，DUA在下游的情况。为了保证DUA的输入端在有效时钟沿能正确捕获上游数据并向DUA内部或下游发射数据。需要使用set\_input\_delay命令来设置从上游电路时钟引脚到DUA数据输入引脚路径的延迟值。从而约束DUA数据输入端口到DUA内部第一级捕获触发器的时序。

**(set\_input\_delay -min ) < （时序报告中input external delay） < (set\_input\_delay -max)**

用(set\_input\_delay -min)加上DUA数据输入端口到DUA内部第一级捕获触发器的最小延迟作保持时间检查。要求Tlaunch + (set\_input\_delay -min) + Tdua\_min > Tlaunch + Thold

用(set\_input\_delay -max)加上DUA数据输入端口到DUA内部第一级捕获触发器的最大延迟作建立时间检查。要求(set\_input\_delay -max)+Tdua\_max < Tcapture - Tsetup

3）已知外围电路在下游，DUA在上游的情况。为了保证DUA的输出端在有效时钟沿能正确向下游发射数据并被下游电路捕获。需要使用set\_output\_delay命令来设置从DUA数据输出端口到下游电路捕获触发器数据输入路径**再减去Thold**的延迟值。从而约束DUA末级触发器时钟输入引脚到DUA数据输出端口的时序。

**(set\_output\_delay -min ) < （时序报告中output external delay） < (set\_output\_delay -max)**

用DUA内部末级发射触发器到DUA数据输出端口的最小延迟加上(set\_output\_delay -min)作保持时间检查。要求Tlaunch + Tdua\_min > Tlaunch - (set\_output\_delay -min)

用DUA内部末级发射触发器到DUA数据输出端口的最大延迟加上(set\_output\_delay -max)作建立时间检查。要求Tdua\_max < Tcapture - (set\_output\_delay -max)

## 8.3 多周期（时序）路径

某些情况下，两触发器之间的组合逻辑数据时序路径可能需要超过1个时钟周期

才能传播，此种情况下，组合逻辑（时序）路径被声明为多周期路径（Multicycle Paths）。即使数据可被捕获触发器的每个时钟沿捕获，也可令STA相关的捕获沿实在指定数量时钟周期之后发生。

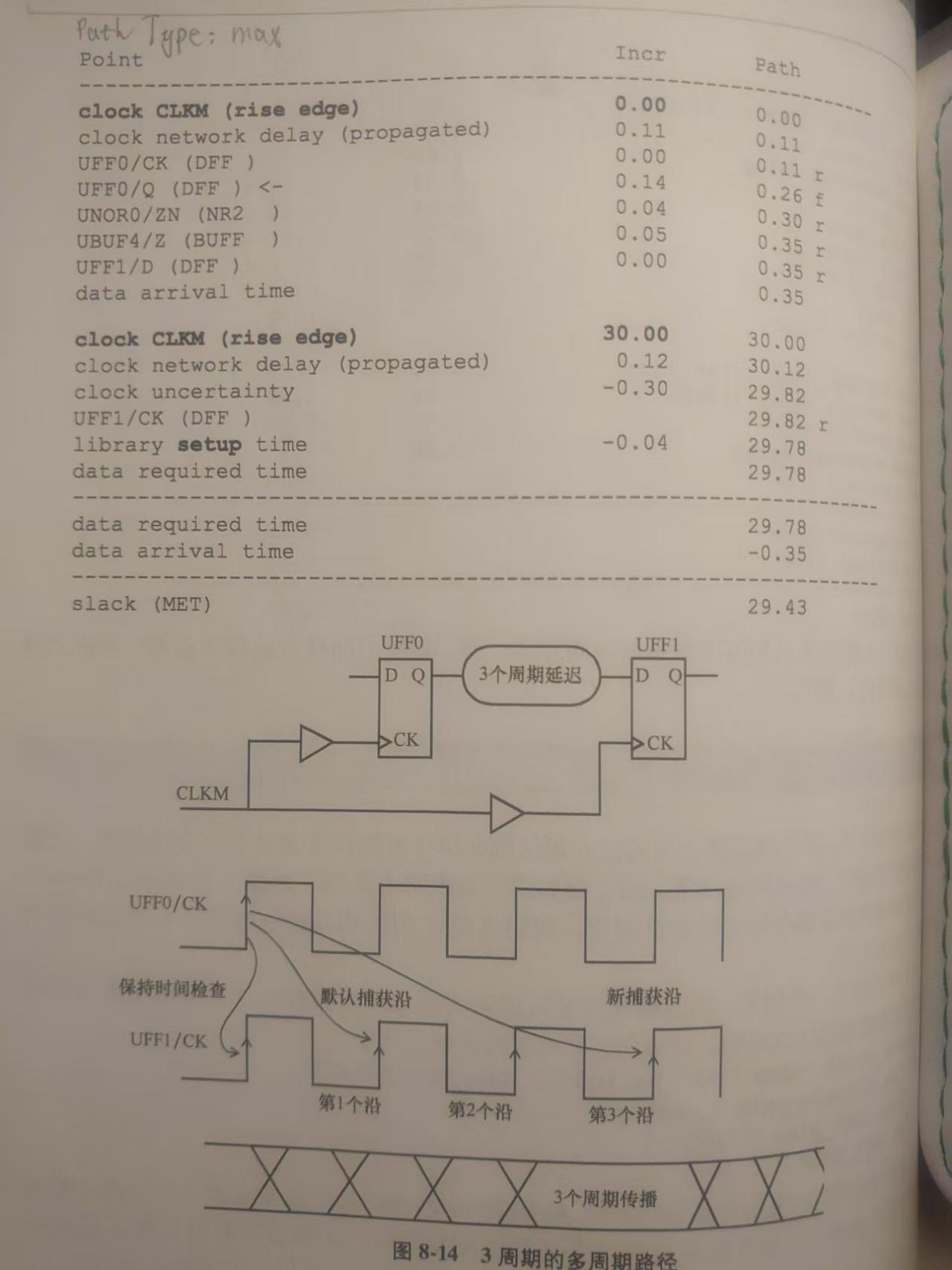
create\_clock -name CLKM -period 10 [get\_ports CLKM]

set\_multicycle\_path 3 -setup -from [get\_pins UFF0/Q] -to [get\_pins UFF1/D]

从UFF0/Q到UFF1/D因数据路径可能需要3个时钟周期来传递，所以需要需要指定建立时间检查推迟3个周期。时序报告如下：

注意报告中，数据到达UFF1/D的时间为0.35；建立检查的检查点为（UFF1/D，29.78ns）。

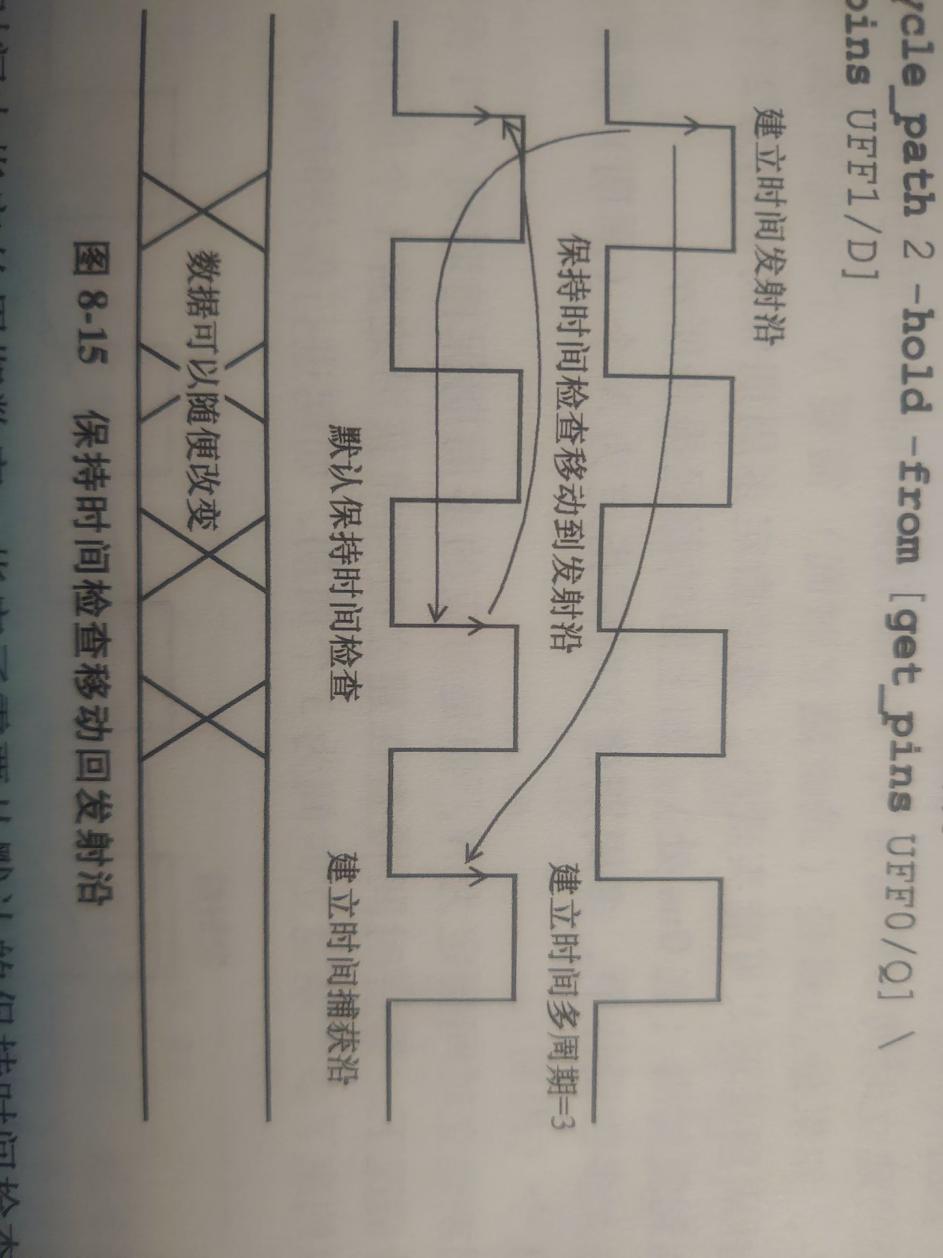
注意上面两个触发之间的数据时序路径需要多时钟周期，假设时钟数是N>1，那么即是说：0个时钟周期 < 最大/最小数据延迟 < N个时钟周期；最大与最小数据延迟的差值是不确定的。



对于多周期时序路径的保持时间检查，在常见情况下，想让保持时间检查时刻与单周期情况一致，也就是说在UFF1/D处的上一次数据应该尽早发射出去，如上图中默认捕获沿的前一个时钟沿时刻，此时刻应早于数据在两个触发器之间的最小传输延迟（显然是容易实现的）。

但是STA软件默认的保持时间检查时刻是有效捕获沿（第3个沿）的前一个沿。因此还需要用指令指明，将保持检查时刻提前2个时钟周期，如下：

set\_multicycle\_path 2 -hold -from [get\_pins UFF0/Q] -to [get\_pins UFF1/D]

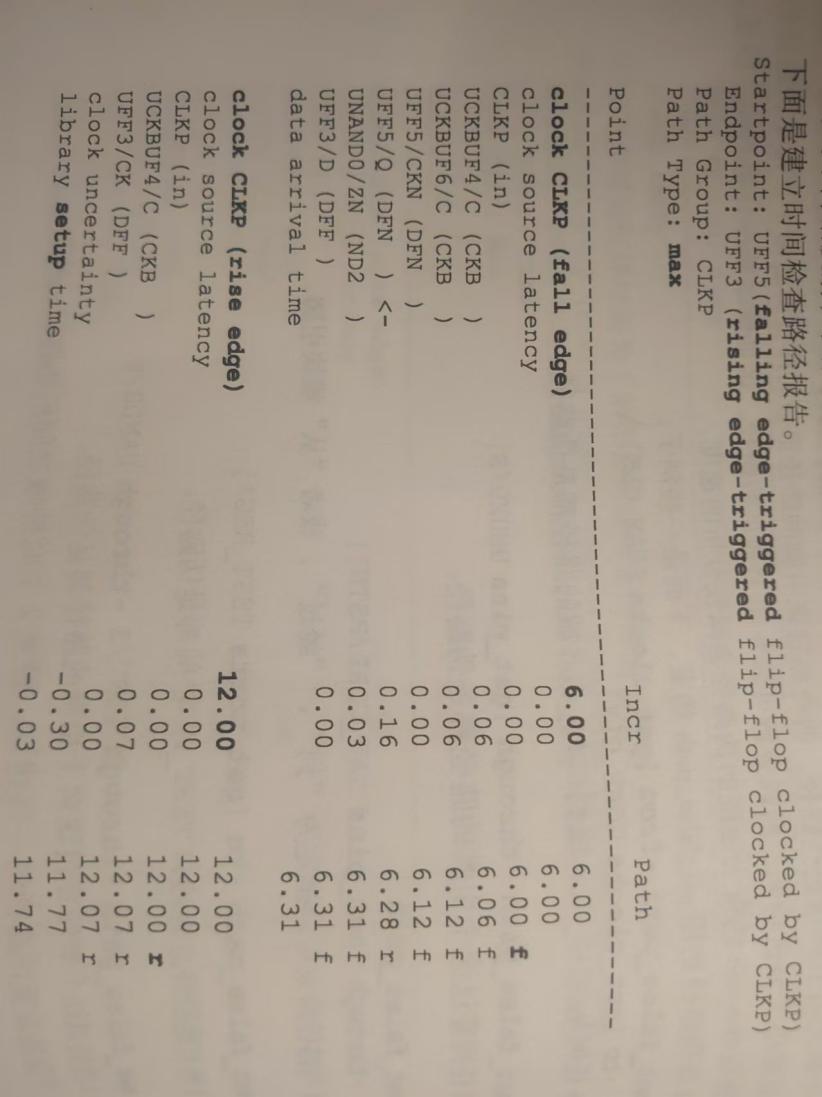


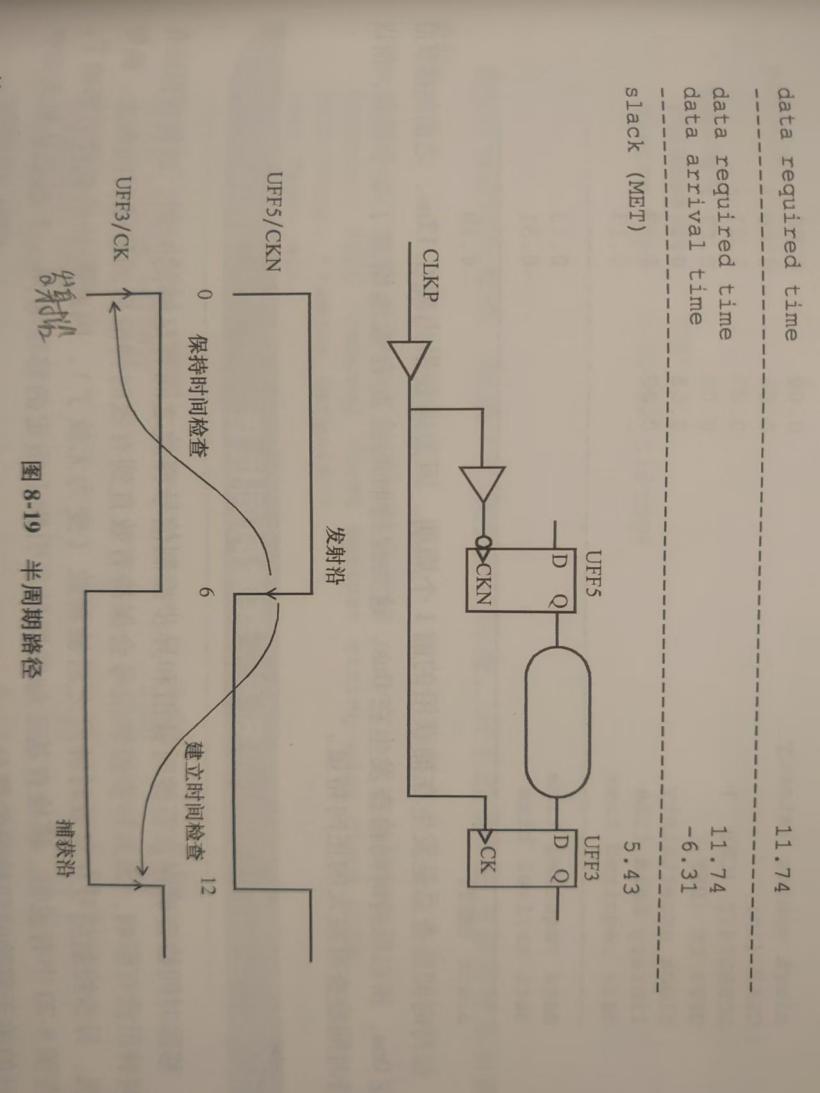
**在大多数设计中，多周期建立时间检查指定为N个周期（滞后N），需要配合多周期保持时间检查指定为N-1个周期（提前N-1）。**

另，通常情况下，建立时间和保持时间检查报告是在同一时序工艺角条件下生成的。但是，建立时间检查通常在最差情况慢速工艺角最难满足要求（有最小的裕量），而保持时间检查通常在最佳情况快速工艺角最难满足（有最小裕量）。

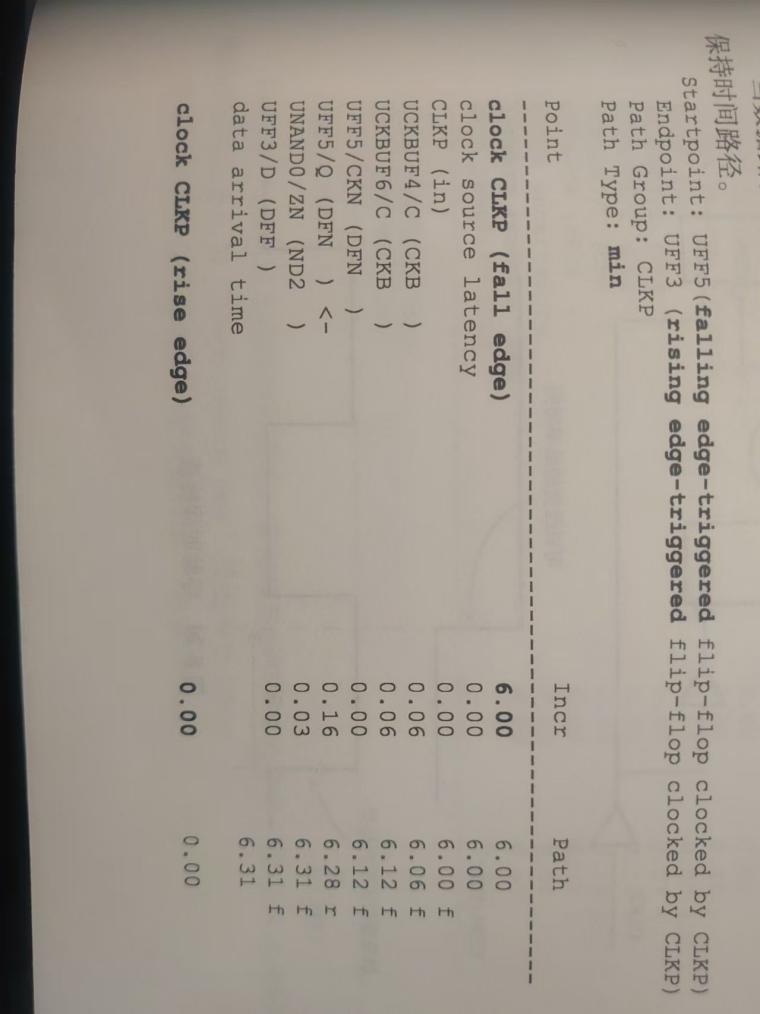
## 8.4 半周期（时序）路径

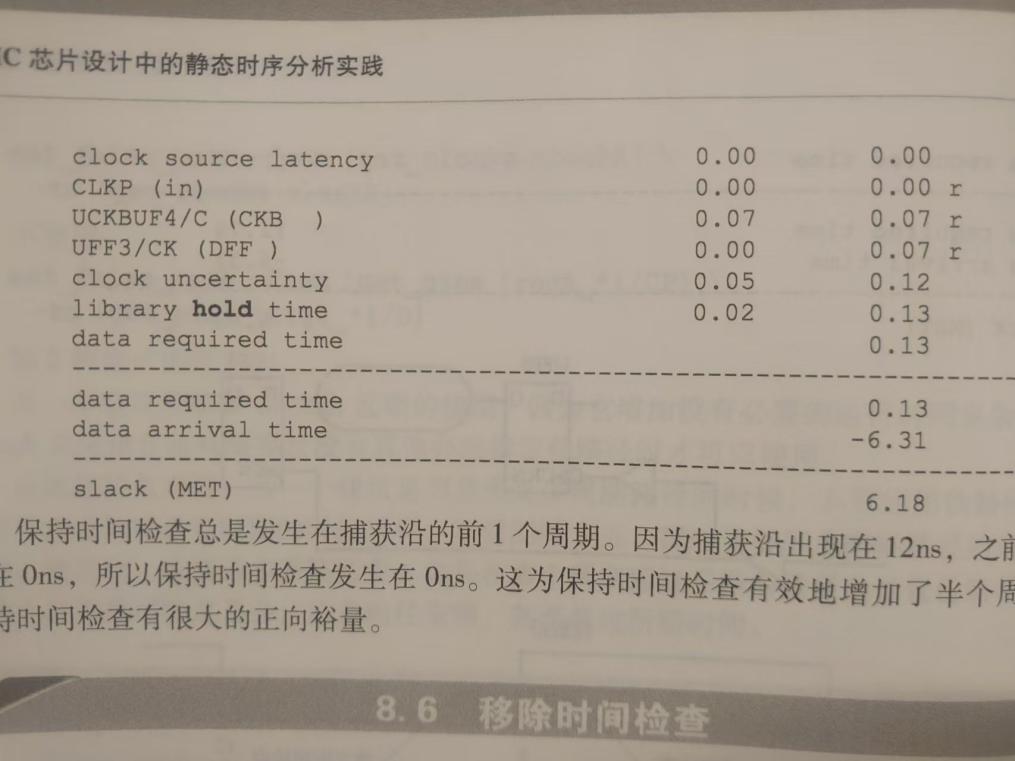
如果设计中既有负沿触发，又有正沿触发，则在此设计中可能存在半周期时序路径（Half-cycle Path）。





由图8-19，data arrival time 从CLKP后半周期的发射沿起，经UFF5/Q到UFF3/D，时间是6.31；data require time 从一个周期后的CLKP捕获沿起，计算得得时间是11.74，setup slack = 5.43





保持检查的data arrival time 从CLKP后半周期发射起，经UFF5/Q到UFF3/D，时间是6.31；data require time 从UFF3/CK的发射沿起，计算得得时间是0.13，hold slack = 6.18。

相比单周期传输，半周期传输的建立裕量减少半周期，保持裕量增加半周期。

## 8.5 跨时钟域的时序

8.5.1 低频时钟到高频时钟域

create\_clock -name CLKM -period 20 -waveform {0 10} [get\_ports CLKM]

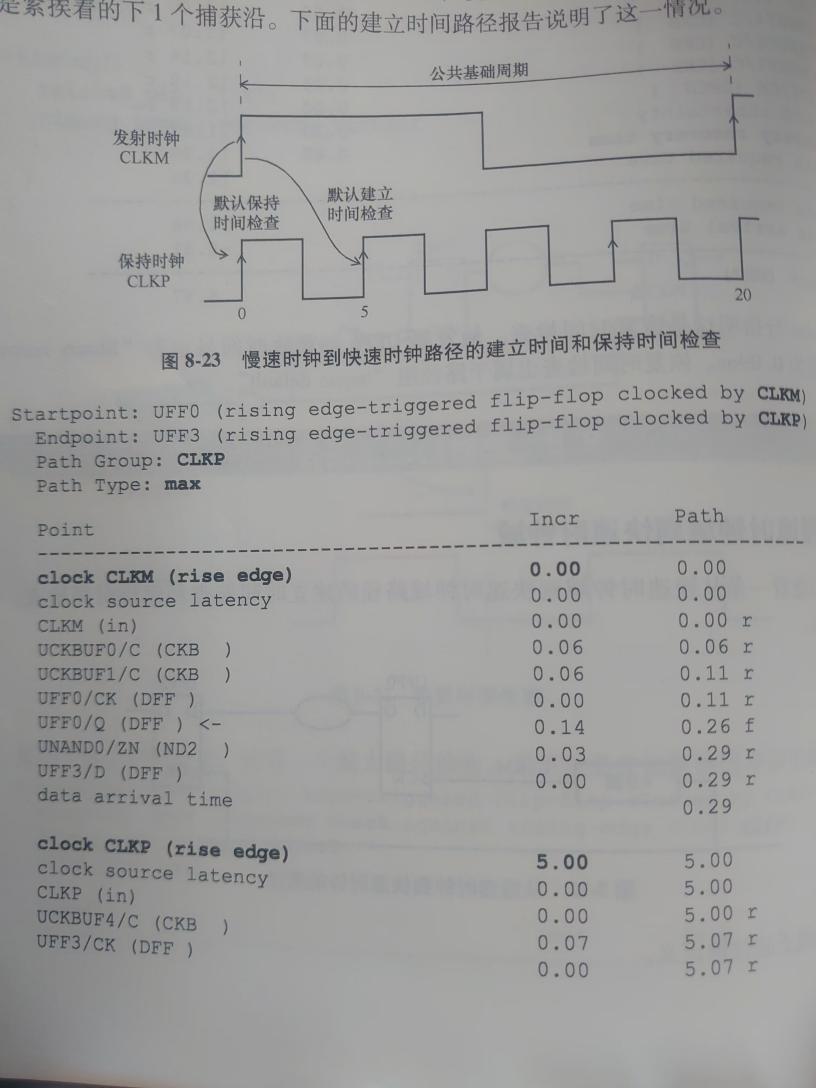
create\_clock -name CLKP -period 5 -waveform {0 2.5} [get\_ports CLKP]

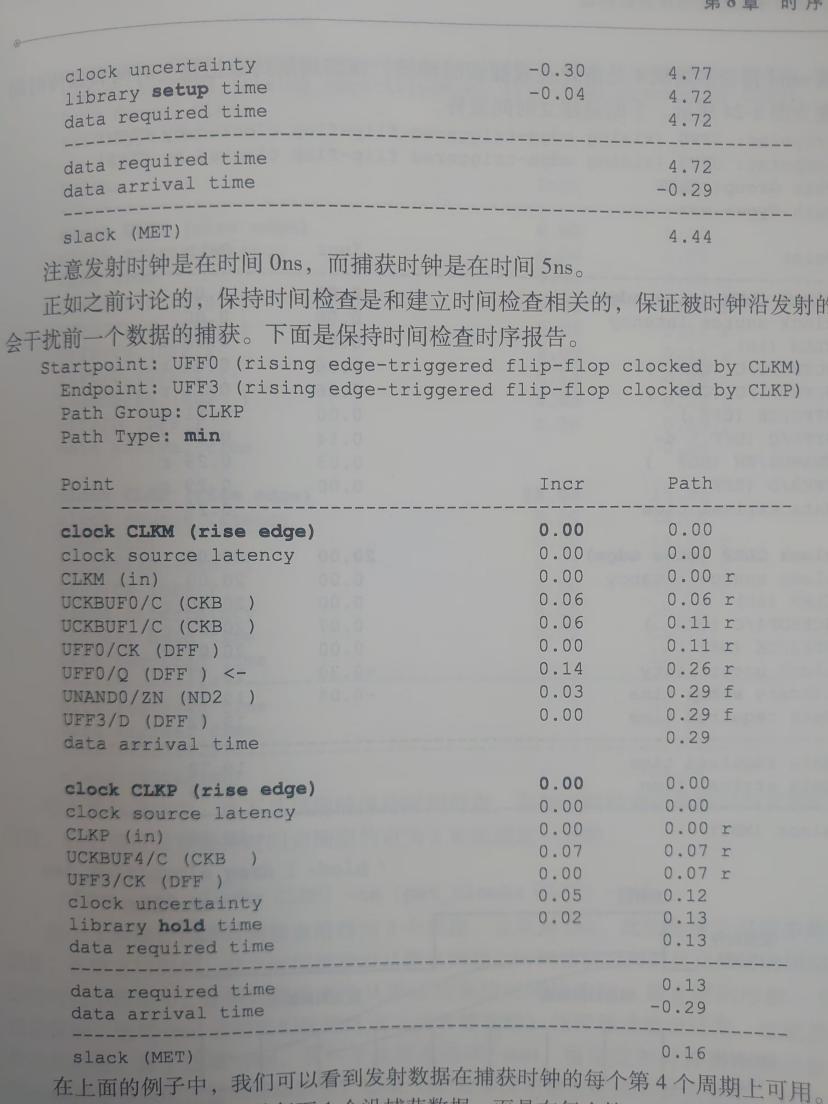
当发射触发器与捕获触发器的时钟频率不同，STA首先需要确定共同基础周期（Common Base Period）。在上面两个时钟进行STA时，更快的时钟周期会被扩展，以得到共同周期：

Expanding clock ‘CLKP’ to base period of 20.00

(old period was 5.00, added 6 egdes, 3 periods)







默认情况下，使用最紧的建立时间沿关系，保持时间沿也默认是捕获沿之前的时钟沿。上例中依据CLKP时钟周期作建立/保持时间检查。

如想在CLKP第4个周期捕获CLKM发射的数据，可设置多周期约束：

set\_multicycle\_path 4 -setup -from [get\_clocks CLKM] to [get\_clocks CLKP] -end

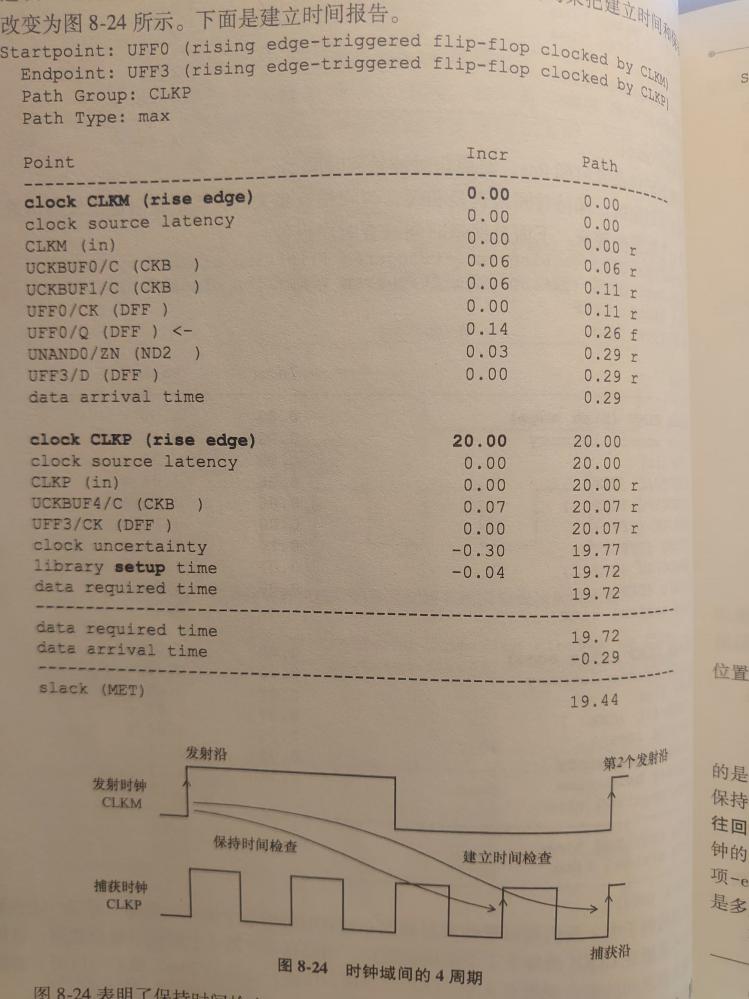
# set\_multicycle\_path的默认值是1，就是最紧的捕获沿。-end指定多周期4是终点或捕获时钟沿（相对于CLKP的移动数量）。

保持时间检查是从建立时间检查衍生出来的，默认是在预期捕获沿之前的一个沿。

在大部分设计中，这希望保持检查沿回到发射沿位置，所以：

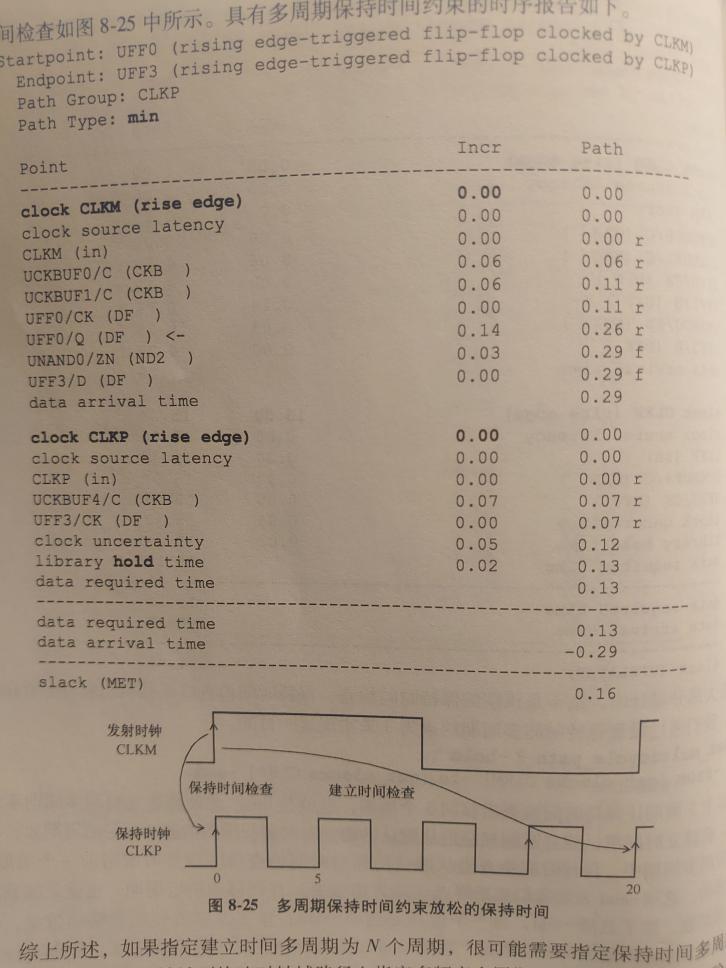
set\_multicycle\_path 3 -hold -from [get\_clocks CLKM] to [get\_clocks CLKP] -end

-end选项是表明把终点或者捕获沿往前移动指定周期（CLKP的周期）。另一个选项是-start，指定发射时钟周期移动数量（相对于CLKM）。-end是多周期建立时间的默认选项，-start是多周期保持时间的默认选项。



set\_mutlticycle\_path 3 -hold -from [get\_clocks CLKM] to [get\_clock CLKP] -end

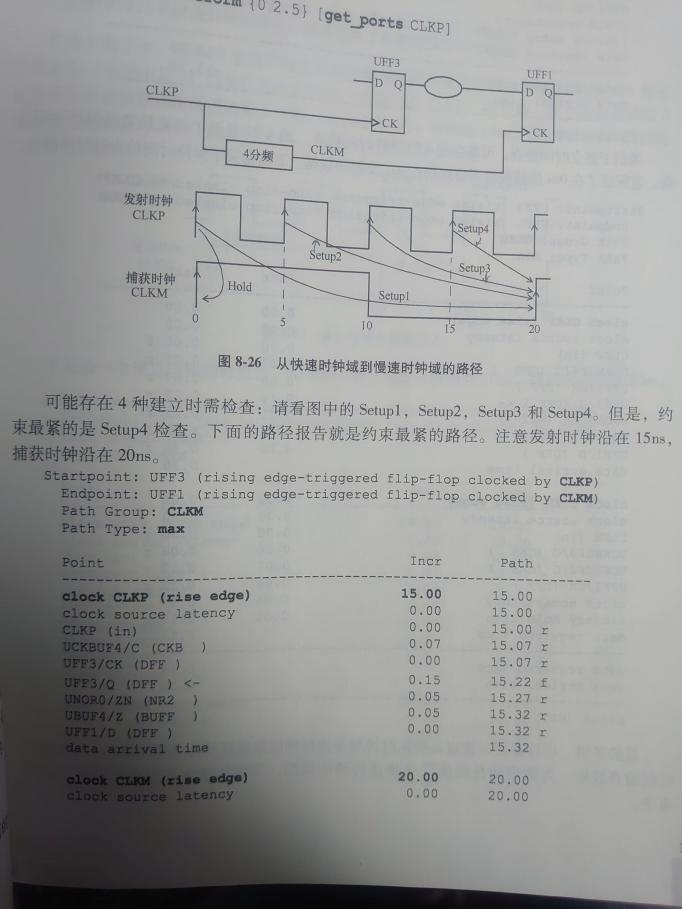
# 将CLKP保持检查沿前移3个周期，默认是0个周期，也就是捕获沿前一周期

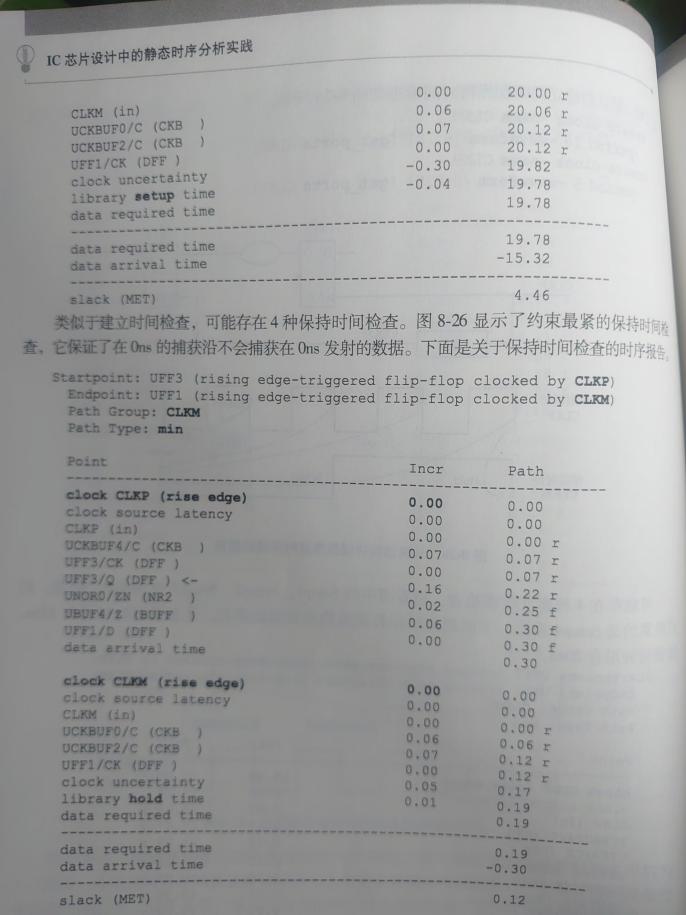


8.5.2 高频时钟域到低频时钟域

create\_clock -name CLKP -period 5 -waveform {0 2.5} [get\_port CLKP]

create\_generate\_clock -name CLKM -source CLKP -divided\_by 4 [get\_ports CLKM]





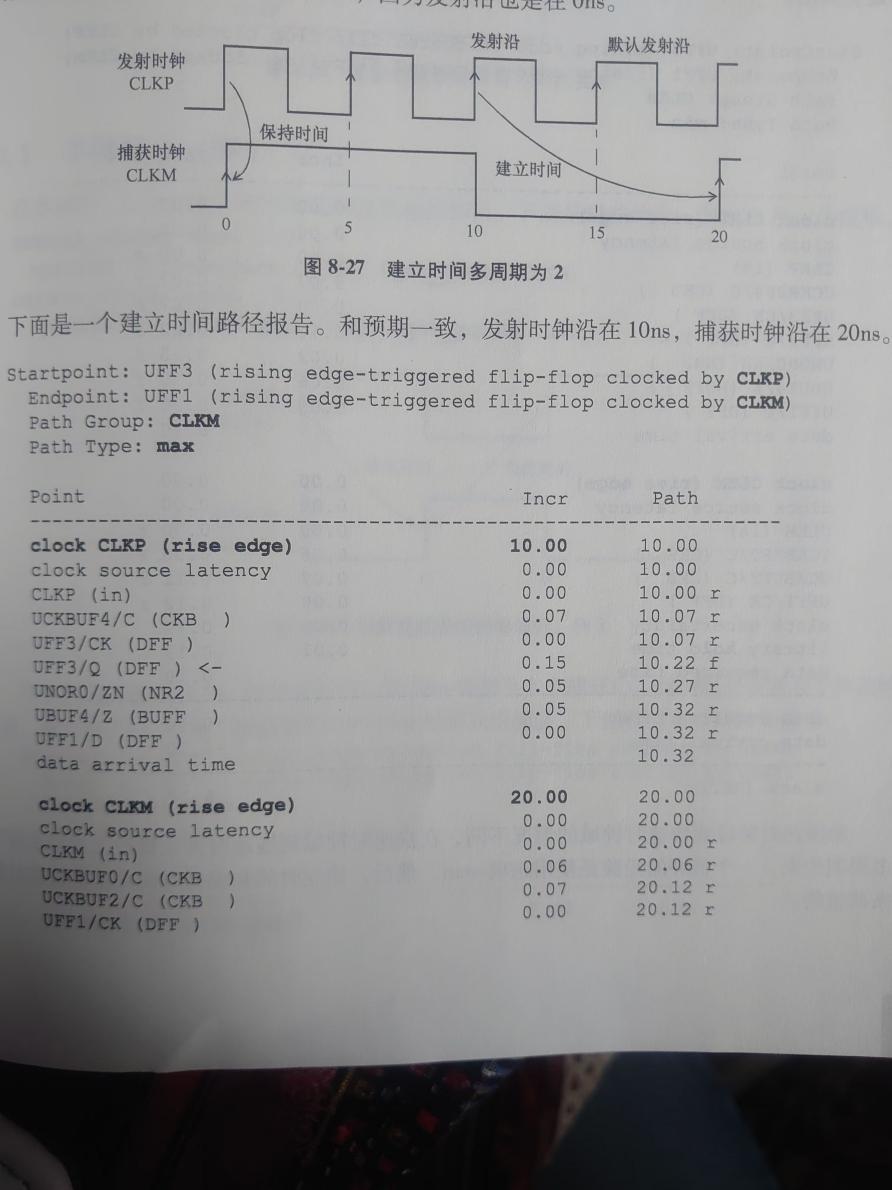
通常在设计中可以指定从快速时钟到慢速时钟的数据路径为多周期路径。建立时间放松2个发射周期。-start选项是相对于发射时钟。

set\_muticycle\_path 2 -setup -from [get\_clocks CLKP] -to [get\_clocks CLKM] **-start**

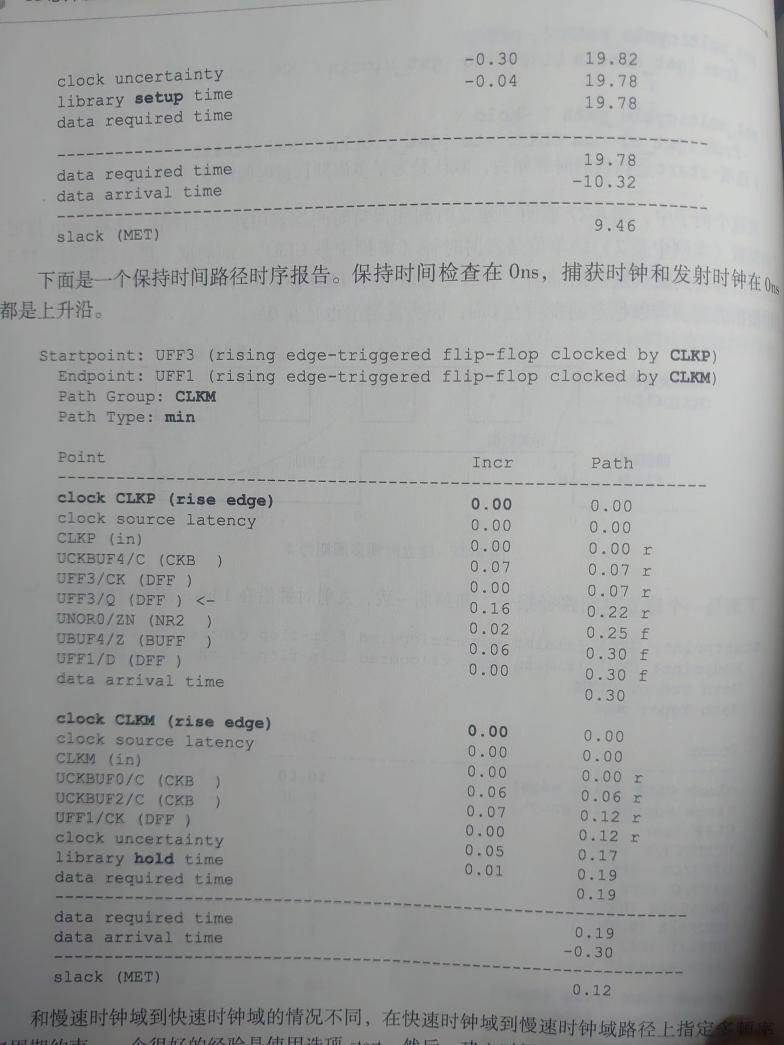
# 相对于clkp，发射沿调整为第2个周期后的上升沿。

set\_muticycle\_path 1 -hold -from [get\_clocks CLKP] -to [get\_clocks CLKM] **-start**

**#** 相对于clkp，保持检查沿调整为发射沿前两个周期的上升沿。



上图中，发射沿在10ns，捕获沿20ns。



对于上图，保持检查沿在0ns。下面命令不太合理，因为保持检查的对象是捕获触发器的驱动时钟（CLKM），

set\_muticycle\_path 1 -hold -from [get\_clocks CLKP] -to [get\_clocks CLKM] **-start**

**# 不合理，这句为什么不能理解成发射上升沿在10ns，保持检查沿在0ns**

**# 这句与上图报告理解存在问题。**

set\_muticycle\_path 0 -hold -from [get\_clocks CLKP] -to [get\_clocks CLKM] **-end**

**# 这一句是相对于捕获触发器，‘0’是默认值，所以不写这句也可。**

**# 这一句符合上图报告**

## 8.6 伪路径

在电路实际运行时，可能存在一些时序路径不是真实的（或者不存在），可以设置伪路径（False Path）来关闭STA检查。

Pseudo-path可以是：

1. 一个时钟域到另一个时钟域
2. 一个触发器时钟引脚到另一个触发器输入引脚
3. 经过一个单元或经过多个单元的引脚
4. 以上情况的组合

当通过一个单元的引脚指定为伪路径，所有经过该引脚的路径都会被STA忽略。

-though 参数去通配伪路径会减慢分析速度。

set\_false\_path -from [get\_clocks SCAN\_CLK] -to [get\_clocks CORE\_CLK]

# 任何从SCAN\_CLK时钟域到CORE\_CLK时钟域的路径都是伪路径

set\_false\_path -through [get\_pins UMUX0/S]

# 任何穿过这个引脚的路径都是伪路径

set\_false\_path -through UINV/Z -through UAND0/Z

# 任何以这个顺序穿过这两个引脚的路径都是伪路径

set\_false\_path -from # 任何从

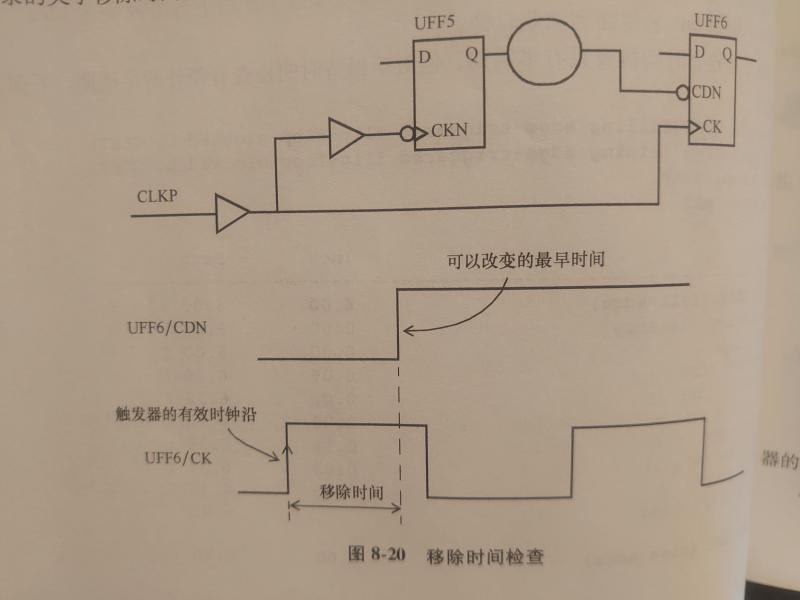
set\_false\_path -to # 任何到

建议减少-through参数，因为增加了复杂度和运行时间。当意图是多周期路径的时候，不要使用伪路径。

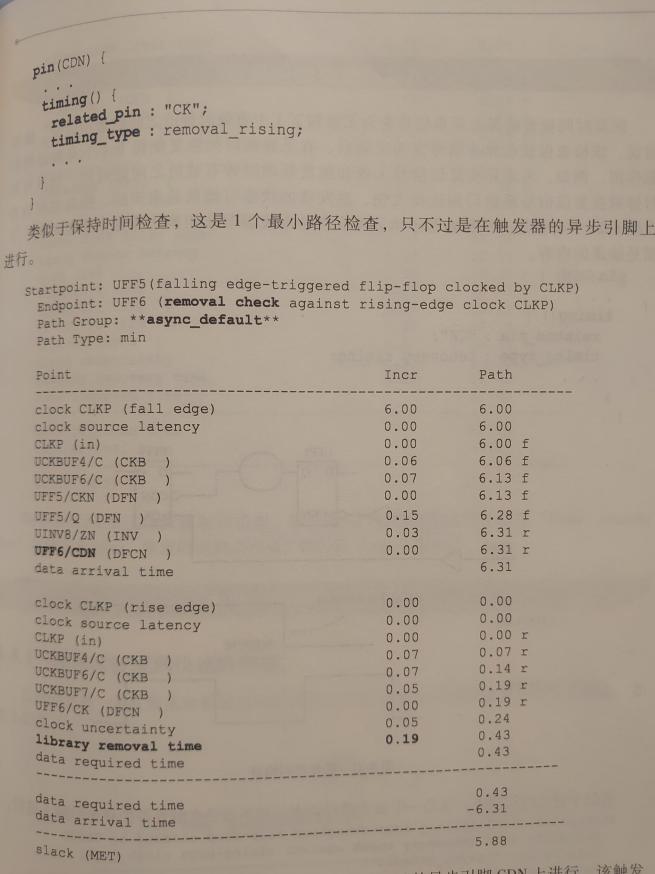
## 8.7 移除时间检查

移除时间检查确保在**当前有效时钟沿**和**异步控制信号**之间有足够的时间。异步控制信号在有效时钟沿之后被释放（变无效），因此有效时钟沿就不会对电路产生影响。

移除时间检查类似于保持时间检查，是一个最小路径检查，只不过是在触发器的异步引脚上进行。（相对比于保持时间检查）



检查引脚是CDN，CDN在圆圈右侧是**高**电平有效，圆圈左侧输入是**低**电平有效。由上图，移除时间是使当前时钟沿无影响，异步控制信号的保持时间。



注意上面时序报告：

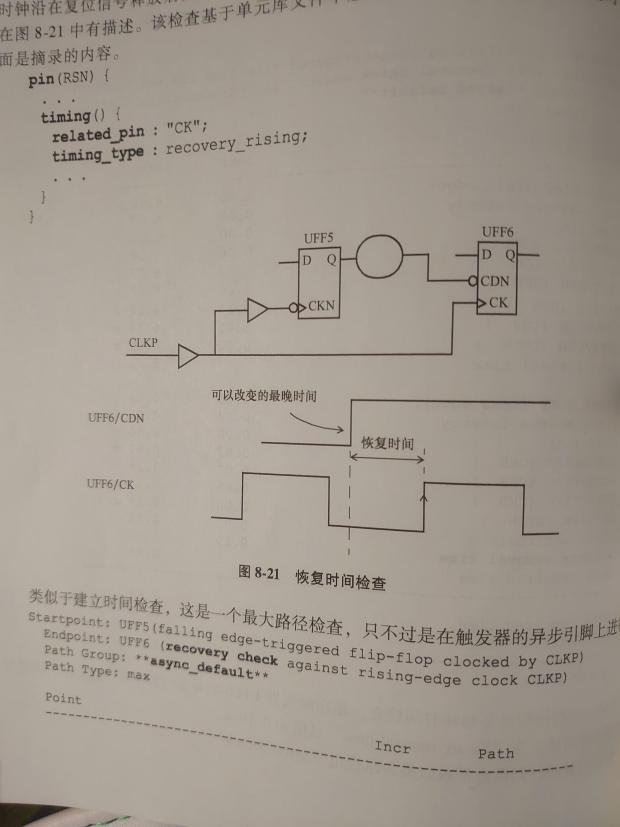
Path Group：“async default”

检查引脚是：UFF6/CDN

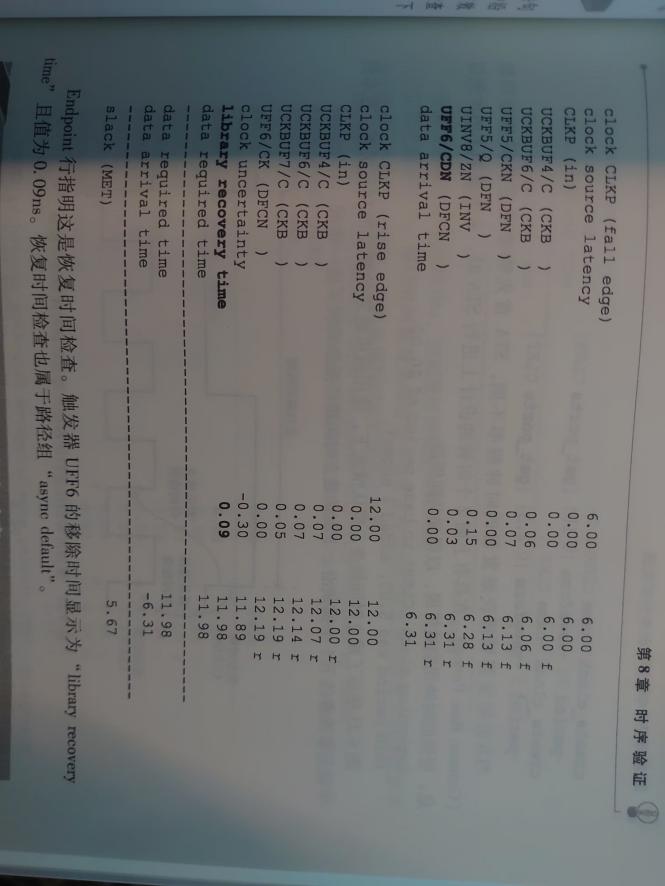
Slack：6.31 - 0.43 = 5.88

## 8.8 恢复时间检查

恢复时间检查确保在异步信号变为无效和下一个有效时钟沿之间的最小时间。在异步信号变为无效后，有足够的时间恢复，使得下一个有效时钟沿启作用。



恢复时间检查类似于建立时间检查，是一个最大路径检查，检查点是触发器异步输入引脚。（相对比于建立时间检查）



注意上面时序报告：

Path Group：“async default”

检查引脚是：UFF6/CDN

Slack：11.98 - 6.31 = 5.67

为什么？？？ library recovery time 不是 -0.09

移除时间与恢复时间限制了**异步信号无效沿**在当前时钟与下一个时钟之间的范围。

## 8.9 实例

## 8.10 多倍时钟

8.10.1 整数倍

8.10.2 非整数倍

8.10.3 相移

# **接口分析**

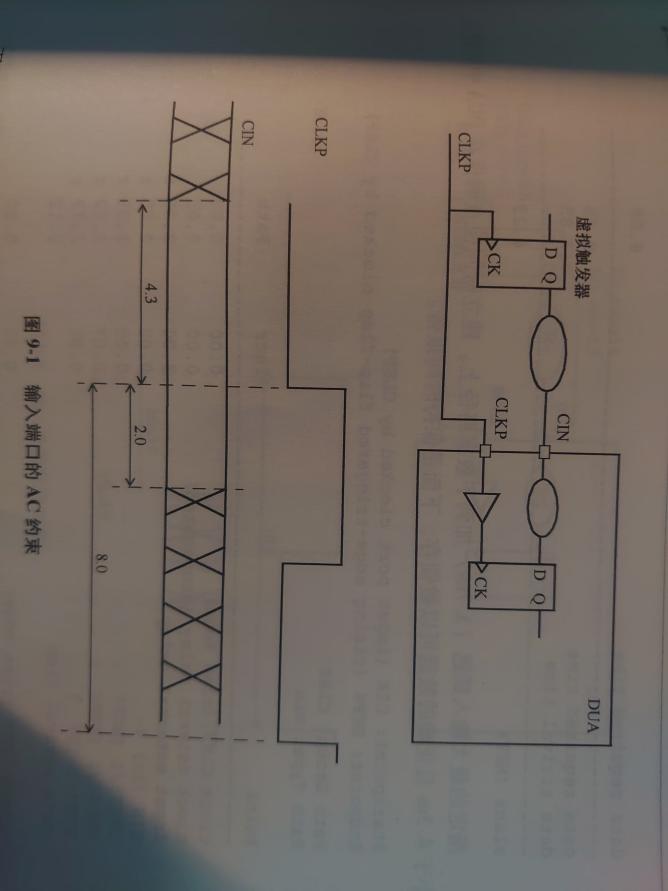
## 9.1 IO接口

9.1.1 输入接口

两种常见的输入时序的指定方法：

1. 以AC约束的形式指定DUA的输入端的波形
2. 指定到输入端的外部逻辑的路径延迟

1、依据输入端口的波形来约束。



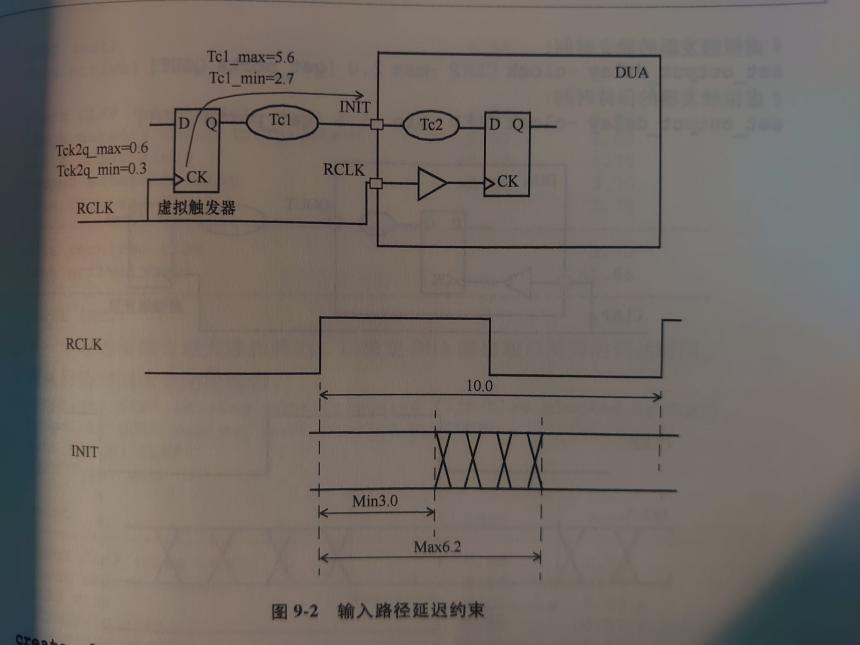
依据上图输入AC约束。该约束包括输入CIN在时钟CLKP上升沿之前的4.3ns稳定，且保持到时钟上升沿之后2ns。

create\_clock -name CLKP -period 8 [get\_ports CLKP]

set\_input\_delay 2 -min -clock CLKP [get\_port CIN]

set\_input\_delay [expr 8 - 4.3] -max -clock CLKP [get\_ports CIN]

1. 依据输入端口的路径延迟来约束



当外部逻辑到输入的路径延迟是已知的，可以直接指定输入延迟。

create\_clock -name RCLK -period 10 [get\_ports RCLK]

set\_input\_delay [expr 0.6 + 5.6] -max -clock RCLK [get\_ports INIT]

set\_input\_delay [expr 0.3 + 2.7] -min -clock RCLK [get\_ports INIT]

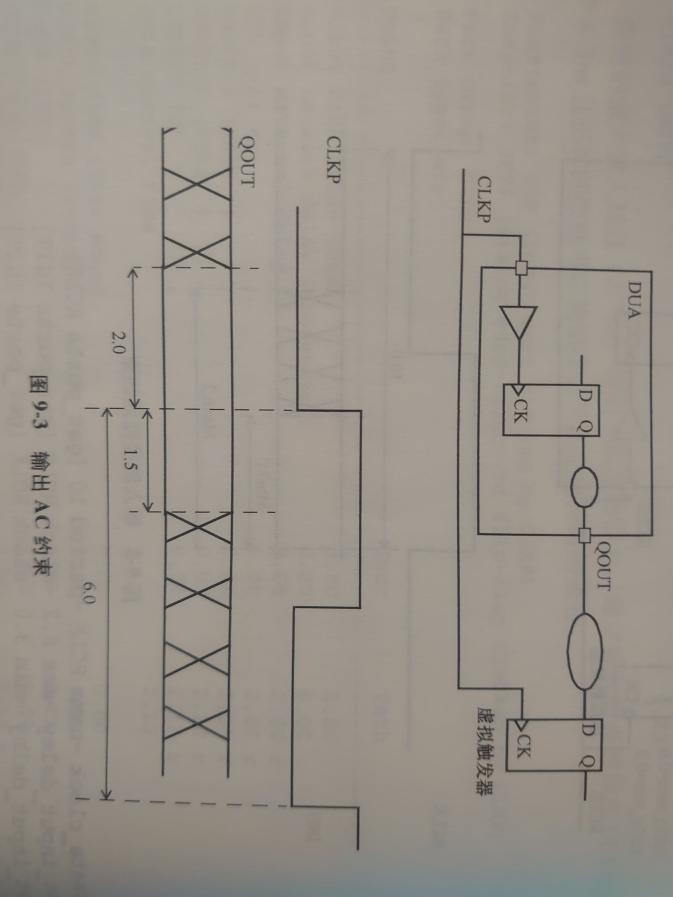
9.1.2 输出接口

两种常见的输出时序的指定方法：

1）以AC约束的形式指定DUA的输出端的波形

2）指定DUA输出端到外部逻辑的路径延迟

1. 输出波形约束



create\_clock -name CLKP -period 6 [get\_ports CLKP]

set\_output\_delay 2 -max -clock CLKP [get\_ports QOUT]

# **设置最大输出延时（QOUT到虚触发器CK）max =（QOUT到D）max + Tsetup**

# 分析上图CLKP到虚拟触发器/D端的最大路径延迟：

# tqout\_max + tc2\_max < tcapture - Tsetup

# tqout\_max < tcapture - (tc2\_max + Tsetup), 令**output\_delay\_max = (tc2\_max + Tsetup)** = +2

# tqout\_max < tcapture - output\_delay\_max = tcapture - 2

# tqout\_max 在tcapture前2ns可以变化，之后需要保持不变。

# tcapture - output\_delay\_max是data\_require\_time\_max,最大数据路径需求时间

# setup\_slack = 数据最大路径需求时间 - 数据最大路径到达时间

# 注意注意！！！从数据路径的空间位置来看，数据从QOUT出发，经C2到达# 虚拟触发器D，然后再对齐到 tcapture。所以在时间上有：

# tqout\_max + **output\_delay\_max** < tcapture ，这与后面的最小输出延时形式一致。

set\_output\_delay -1.5 -min -clock CLKP [get\_ports QOUT]

# **设置最小输出延时（QOUT到虚触发器CK）min =（QOUT到D）min - Thold**

# 分析上图CLKP到虚拟触发器/D端的最小路径延迟：

# tqout\_min + tc2\_min > tlaunch + Thold

# tqout\_min > tlaunch **+** (Thold - tc2\_min) = tlaunch **-** (tc2\_min - Thold)

# (Thold - tc2\_min) = 1.5ns，即tqout\_min在tlaunch后1.5ns可以变化，之前须稳定

# 令**output\_delay\_min = (tc2\_min - Thold)** = -1.5ns,

# tqout\_min > tlaunch - output\_delay\_min,此句与最大路径情况在形式上一致

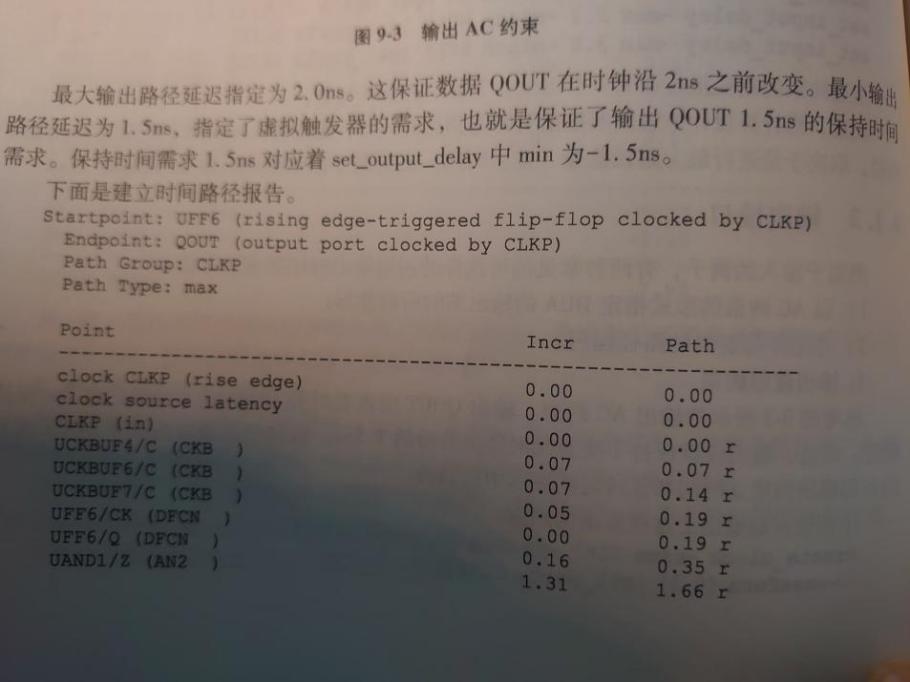
# tlaunch - output\_delay\_min是data\_require\_time\_min,最小路径需求时间

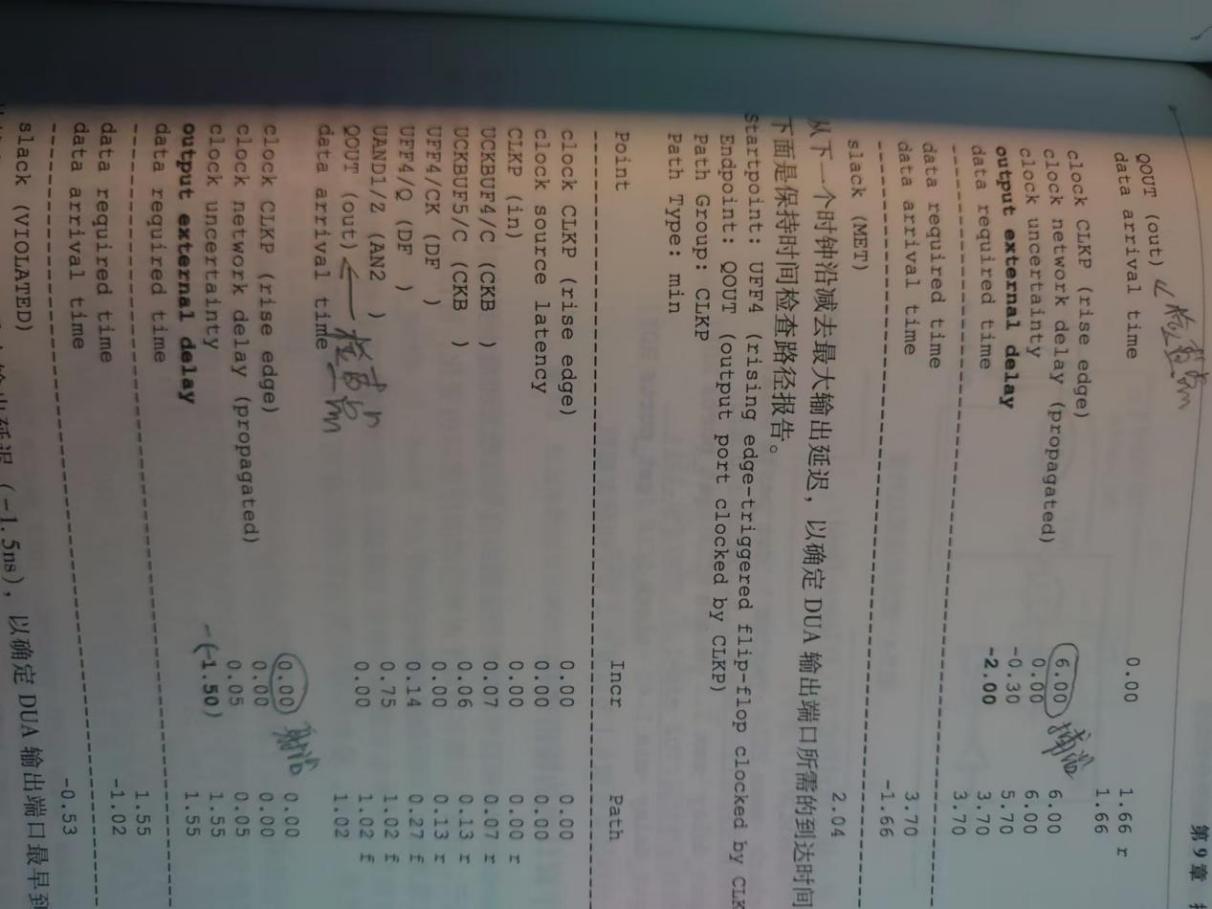
# hold\_ slack = 数据最小路径到达时间 - 数据最小路径需求时间

# 注意注意！！！从数据路径的空间位置来看，数据从QOUT出发，经C2到达

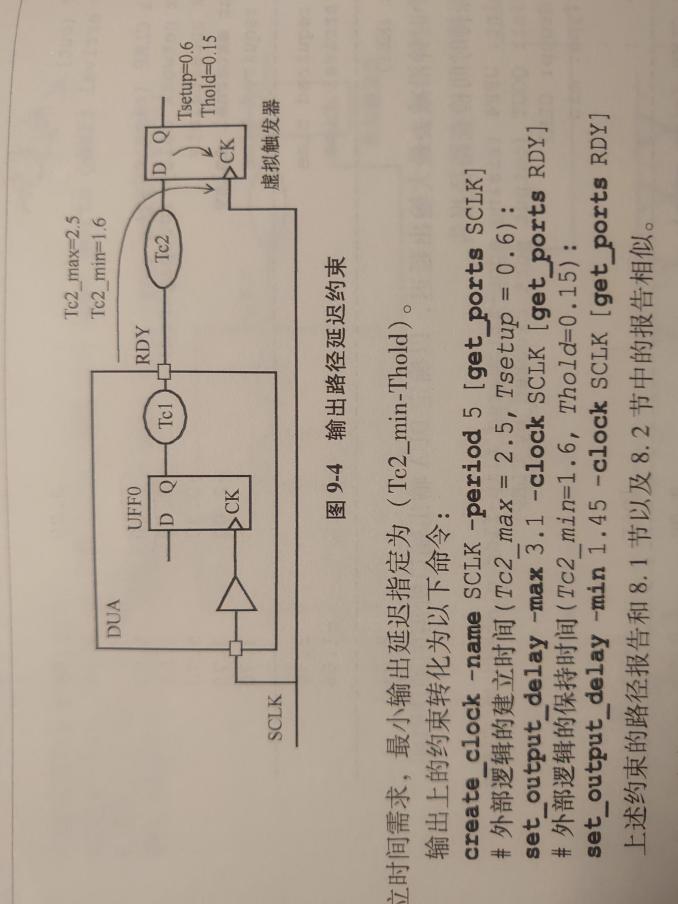
# 虚拟触发器D，然后再对齐到 tlaunch。所以在时间上有：

# tqout\_min + output\_delay\_min > tlaunch ，这与前面的最大输出延时形式一致。





1. 输出到外部路径的约束



set\_output\_delay -min 设置值是(Tc2\_min - Thold)，也就是说DUA/RDY比SCLK有效沿**提前**(Tc2\_min - Thold)，TDUA\_min > Tlaunch - (Tc2\_min - Thold)。

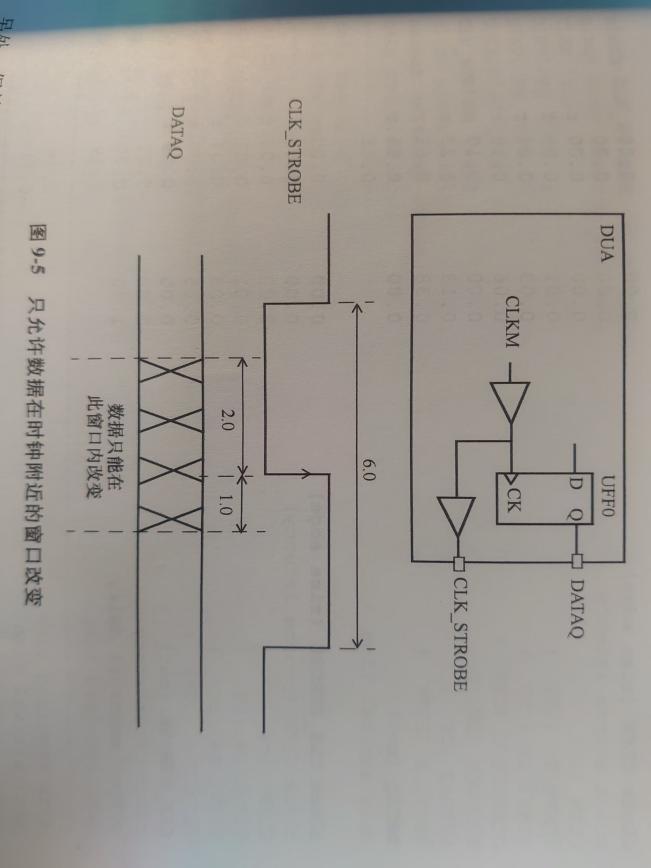
而依据输出波形约束中的示意图，DUA/RDY比SCLK有效沿应**滞后**(Thold - Tc2\_min)，TRDY\_min > Tlaunch + (Thold - Tc2\_min)。

set\_output\_delay -max [expr Tc2\_max + Tsetup] -clock SCLK [get\_port RDY]

set\_output\_delay -min [expr Tc2\_min - Thold] -clock SCLK [get\_port RDY]

9.1.3 时序窗口内的输出变化

9.1.2节描述了DUA输出端口在指定时钟沿窗口内保持稳定而不能变化的情况（系统同步）；而对于DUA输出端口只能在相对于时钟沿窗口内变化，窗口外保持稳定的情况（源同步），也可约束。



在源同步接口，时钟与数据均作为输出，通常对两者的时序关系有要求。例如上图中，DATAQ只能在时钟CLK\_STROBE上升沿前2ns和后1ns的窗口内变化。

create\_clock -name CLKM -period 6 [get\_ports CLKM]

create\_generated\_clock -name CLK\_STROBE -source CLKM -divide\_by 1 \

[get\_ports CLK\_STROBE]

set\_multicycle\_path 0 -setup -to [get\_ports DATAQ]

#将捕获沿提前一个周期

set\_multicycle\_path -1 -hold -to [get\_ports DATAQ]

# 这句为什么不是0 而是-1

# 感觉这句应该取消，没有意义

参考7.5.1及7.5.2，知：

set\_output\_delay -max [expr 0\*T - (+1.0)] -clock CLK\_STROBE [get\_ports DATAQ]

#这句与set\_multicycle\_path -1 -hold -to [get\_ports DATAQ]矛盾

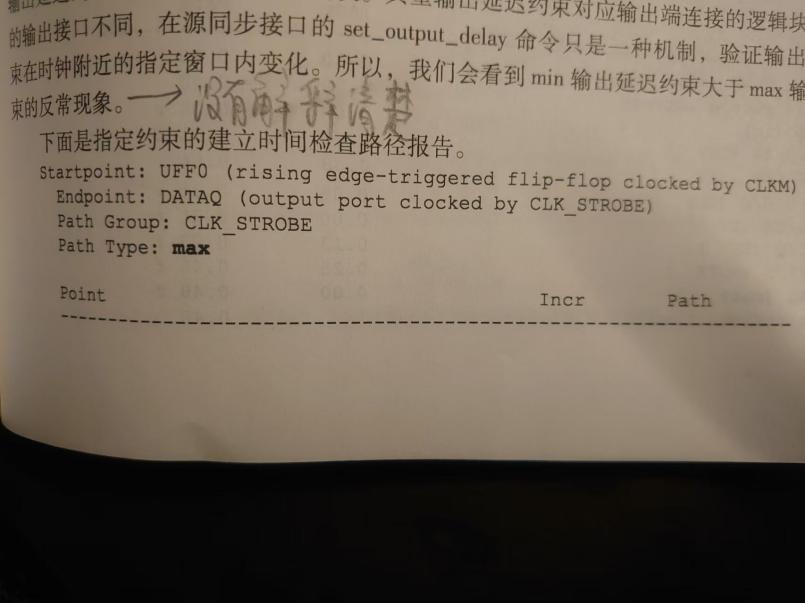
#如果依据上句，则[expr T - (+skew)] = 0\*T - 1.0 = -1.0，结果不变，但是下面

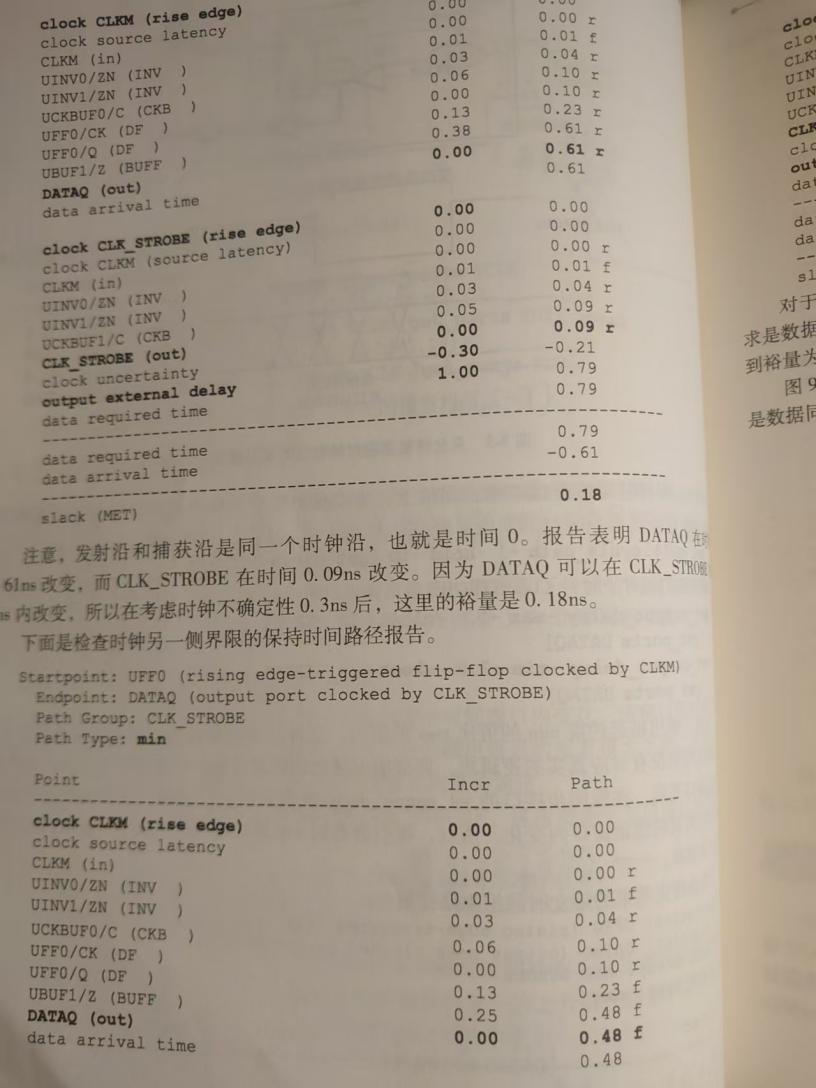
set\_output\_delay -min [expr 0\*T - (-2.0)] -clock CLK\_STROBE [get\_ports DATAQ]

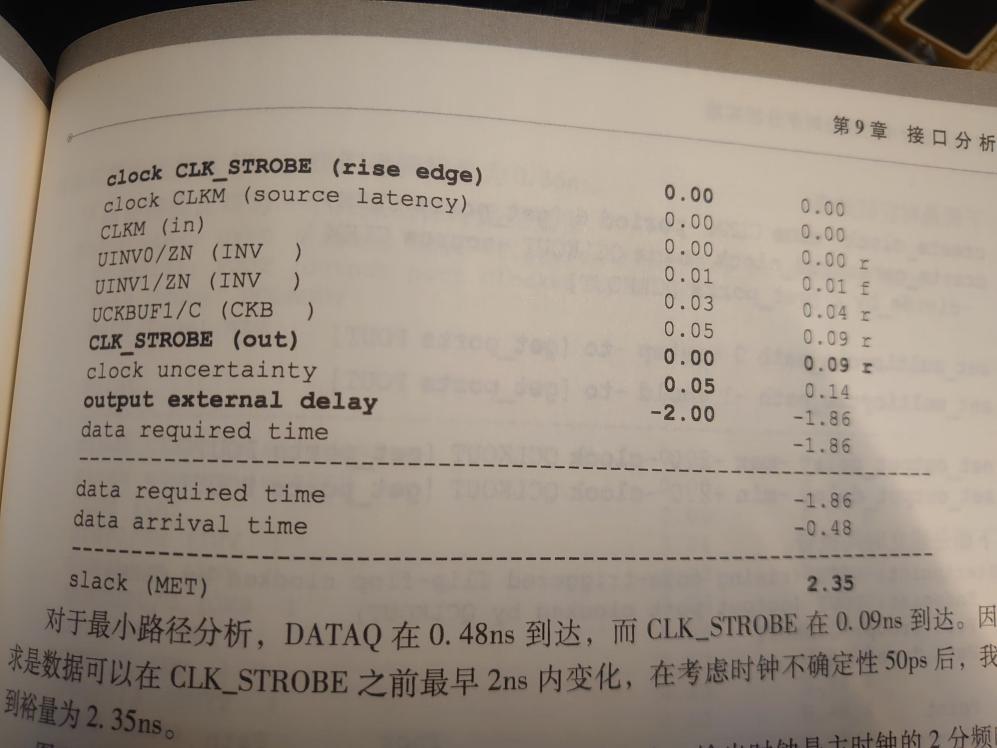
# 这句与set\_multicycle\_path -1 -hold -to [get\_ports DATAQ]矛盾

# 如果依据上句，则应该是[expr (-1)\*T - (-2.0)] = -6 + 2.0 = -4，表示

# output\_delay\_min 提前 -6ns时刻 -4ns，即滞后-6ns时刻4ns，在坐标轴上# 是-2ns时刻







关于上述时序约束与时序报告的理解：

出现数据在有效时钟沿附近变化，而其他时间稳定的约束要求，与通常的时序要求相反的情况，其主要原因是参考时钟的选择问题。

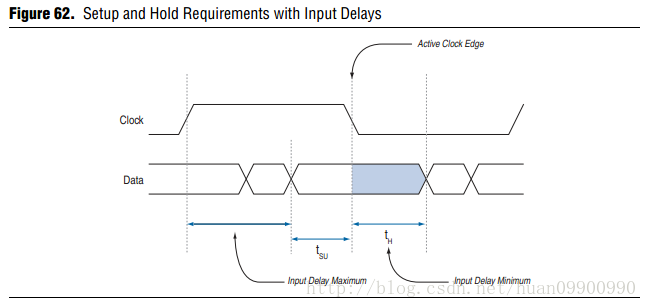
对于DUA的IO接口同步电路，从参考时钟选择的角度触发，可有两种时钟选择方式，即系统同步时钟方式和源同步时钟方式。

* 系统同步方式（System Synchronization, System Centric Synchronization）。从整体的角度，系统各部份电路的参考时钟是统一的时钟（组）。各部分电路的时序约束均以统一的时钟（组）为依据。

根据7.5节表单，分析设置set\_input/output\_delay -min/max -clock ...

* 源同步方式（Source Synchronization, DUA Centric Synchronization）。从局部的角度，电路的参考时钟由上游电路输入，或者DUA向下游电路输出参考时钟，DUA的时序约束以IO接口中的时钟（组）为依据。源同步是指上下游电路间的时序同步仅依据它们之间的传递的时钟信号，而无需第三方参考时钟。DUA Centric Synchronization是指仅依据DUA自身的时序特征来建立时序要求或约束，这与源同步的定义有区别，但两者均为局部视角。

1. 外围电路在上游，DUA在下游，依据DUA的建立和保持要求约束输入。



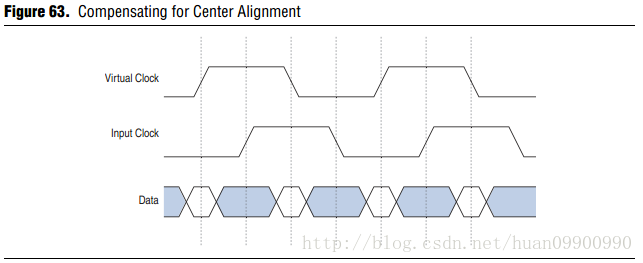
Unit Interval，对于SDR是时钟周期，对于DDR是时钟周期/2。

set\_input\_delay -max [expr **<unit interval> - <setup time>**] -clock\

[get\_clocks Clock] [get\_ports Data]

set\_input\_delay -min <hold time> -clock [get\_clocks Clock]\ [get\_ports Data]

2)外围电路在上游，DUA在下游，依据DUA数据输入对自身虚拟时钟的前后偏斜大小约束输入。



假设上图中Data浅蓝色（Blue）部分表示稳定不变，白色（White）部分表示可变化。在Input Clock的有效上沿和下沿的前后，Data保持不变；在Virtual Clock的有效上沿和下沿的前后，Data可以变化。

上图中的Unit Interval = T/2。Tvirtual\_clock + (Unit Interval)/2 = tinput\_clock, also tvirtual\_clock + T/4 = tinput\_clock

以Input Clock为参考，捕获沿为图片正中的下降沿，时刻为T/2ns，发射沿为捕获下降沿的前一个上升沿，时刻为0ns，输入约束为：

set\_input\_delay -max [expr T/2 - Blue/2 - 0] -clock Input\_Clock \ [get\_ports Data]

# 最大输入延迟是相对从Input Clock在0ns的上升沿起到半个Blue加

# 一个White的时间。

# 从图中可以看出 T/2 = Blue + White => T/4 = Blue/2 + White/2

# 从图中可以看出 T/2 - Blue/2 = Blue/2 + white =0 + T/4 + White/2

set\_input\_delay -min [expr Blue/2 - 0] -clock Input\_Clock \

[get\_ports Data]

# 最小输入延迟是相对从Input Clock在0ns的上升沿起到半个Blue的时

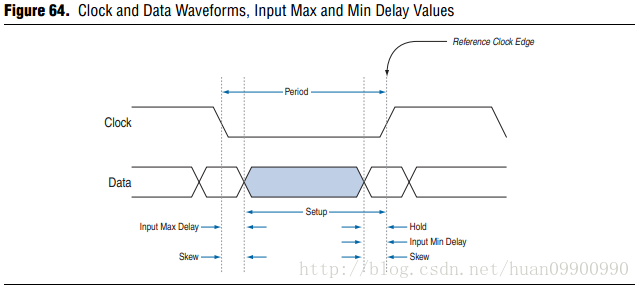
# 间。即是说Data仅能在最小输入延迟于最大输入延迟之间变化

以Virtual Clock为参考，捕获沿只能是图片中线右侧的上升沿，发射沿只能是图片中心左侧的下降沿。同时，建立时间要求与保持时间要求也必须参考Virtual Clock作出调整。虚拟时钟是虚拟出来帮助分析和建立制约条件的。DUA要能正常工作必须依据实际的时钟，用虚拟时钟作为参考，那么约束条件必须比实际时钟更加保守。

建立时间要求调整为：Data还是必须在图片中心线Blue块左端点的右侧保持稳定，Blue块左端点的左侧可以变化。稳定时刻到Virtual Clock的上升捕获沿的距离是Blue + White/2。所以相对于Virtual Clock的下降发射沿，最大输入延迟为White/2 - 0。

保持时间要求调整为：Data可以在Virtual Clock的下降发射沿对应White块的左端点之后变化。White左端点与Virtual Clock的下降发射沿的距离是 White/2。所以相对于Virtual Clock的下降发射沿，最小输入延迟为

-White/2 - 0。如下图：



图中可以看出相对于Virtual Clock，Data在其有效上、下沿左右可以变化。这就是9.1.3节中时序窗口是数据可以变化的范围的情况。设White块右端点对时钟有效沿的偏斜为+skew，White块左端点对时钟有效沿的偏斜为-skew

set\_input\_delay -max [expr +skew - 0] -clock Virtual\_clock [get\_ports Data]

set\_input\_delay -min [expr -skew - 0] -clock Virtual\_clock [get\_ports Data]

上述1）与2）是外围电路在上游，DUA在下游，set\_input\_delay的情况。

3）DUA在上游，外围电路在下游，依据DUA的建立和保持要求约束输出。

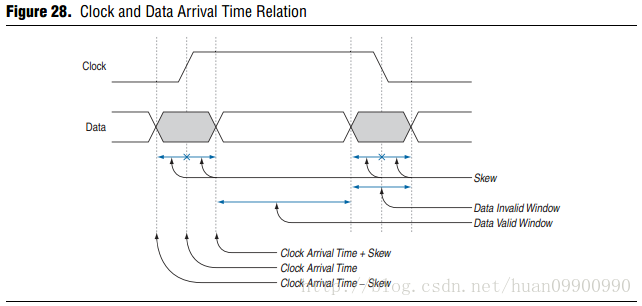
output maximum delay value = maximum trace delay for data + tsu of external register

output minimum delay value = minimum trace delay for data - th of external register

set\_output\_delay -max <output maximum delay value> -clock [get\_ports output\_clock] [get\_ports data\_out]

set\_output\_delay -min <output minimum delay value> -clock [get\_ports output\_clock] [get\_ports data\_out]

4）DUA在上游外围电路在下游，依据DUA输出时钟与数据的偏斜约束输出。



依据上图DUA输出IO的时钟与数据偏斜值关系，灰色块是数据可以变动区间，白色块是数据稳定区间，数据最早到达时间是clock arrive time - skew，数据最晚到达时间是clock arrive time + skew。

clock arrive time - skew < **data arrive time** < clock arrive time + skew

数据最早达到时间必须晚于（clock arrive time - skew），否则不能满足下游器件的保持时间th要求。

数据最晚到达时间必须早于（clock arrive time + skew），否则不能满足下游器件的建立时间tsu要求。

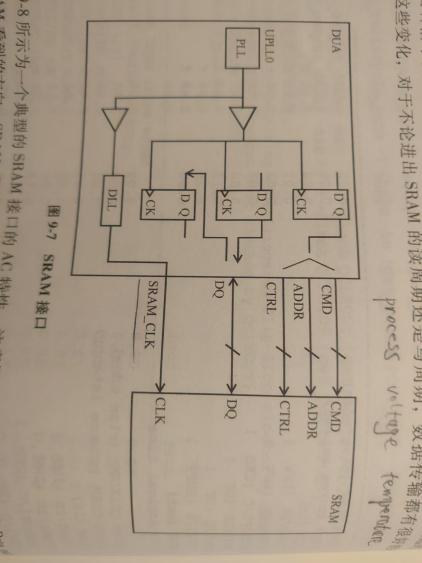
set\_output\_delay -max [expr unit\_interval - (+skew)] -clock Clock [get\_ports \ Data\_out\*]

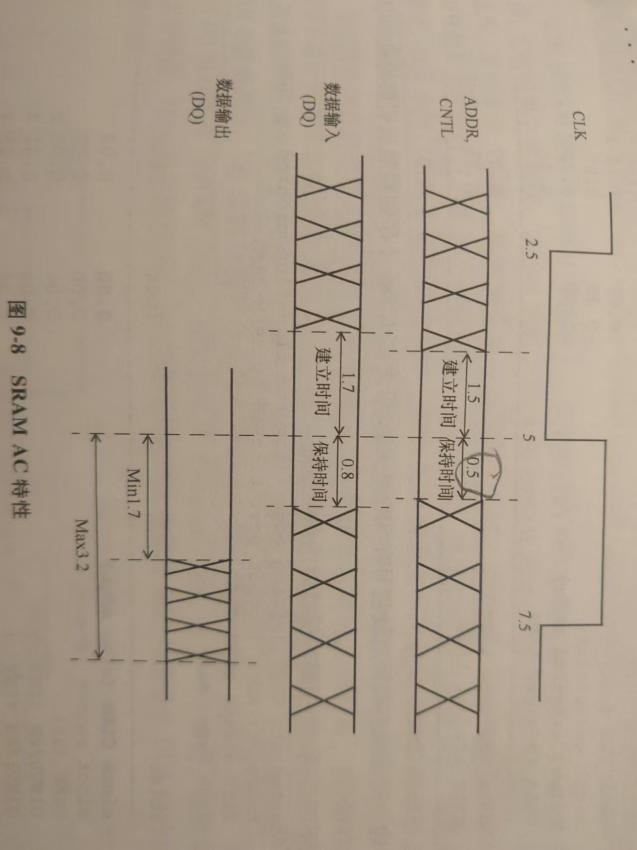
set\_output\_delay -min [expr 0 - (-skew)] -clock Clock [get\_ports Data\_out\*]

## 9.2 SRAM或SDRAM接口

所有SRAM（Static RAM）或SDRAM（Synchronous Dynamic RAM）的数据传输只发生在时钟的有效沿。组成SRAM接口的信号包括：

1. 命令（command）、地址（address）、控制输出总线（control output bus）统称CAC通道。
2. 双向数据总线（bidirectional data bus，DQ）通道。
3. 时钟（clock）。





上图时序图是sram的角度，CAC、DQ、Clock是输入；而DUA的角度三组信号是输出。

create\_clock -name PLL\_CLK -period 5 [get\_pins UPLL0/CLKOUT]

create\_generated\_clock -name SRAM\_CLK -source PLL\_CLK -divide\_by 1 \

[get\_ports SRAM\_CLK]

set\_output\_delay -max [expr 5 - 3.5] -clock SRAM\_CLK [get\_pins ADDR\*]

set\_output\_delay -min [expr 5 - 5.5] -clock SRAM\_CLK [get\_pins ADDR\*]

# ADDR\* CNTL\*

set\_output\_delay -max 1.7 -clock SRAM\_CLK [get\_pins DQ\*]

Set\_output\_delay -min -0.8 -clock SRAM\_CLK [get\_pins DQ\*]

# 约束DUA输出

set\_input\_delay -max 3.2 -clock SRAM\_CLK [get\_pins DQ\*]

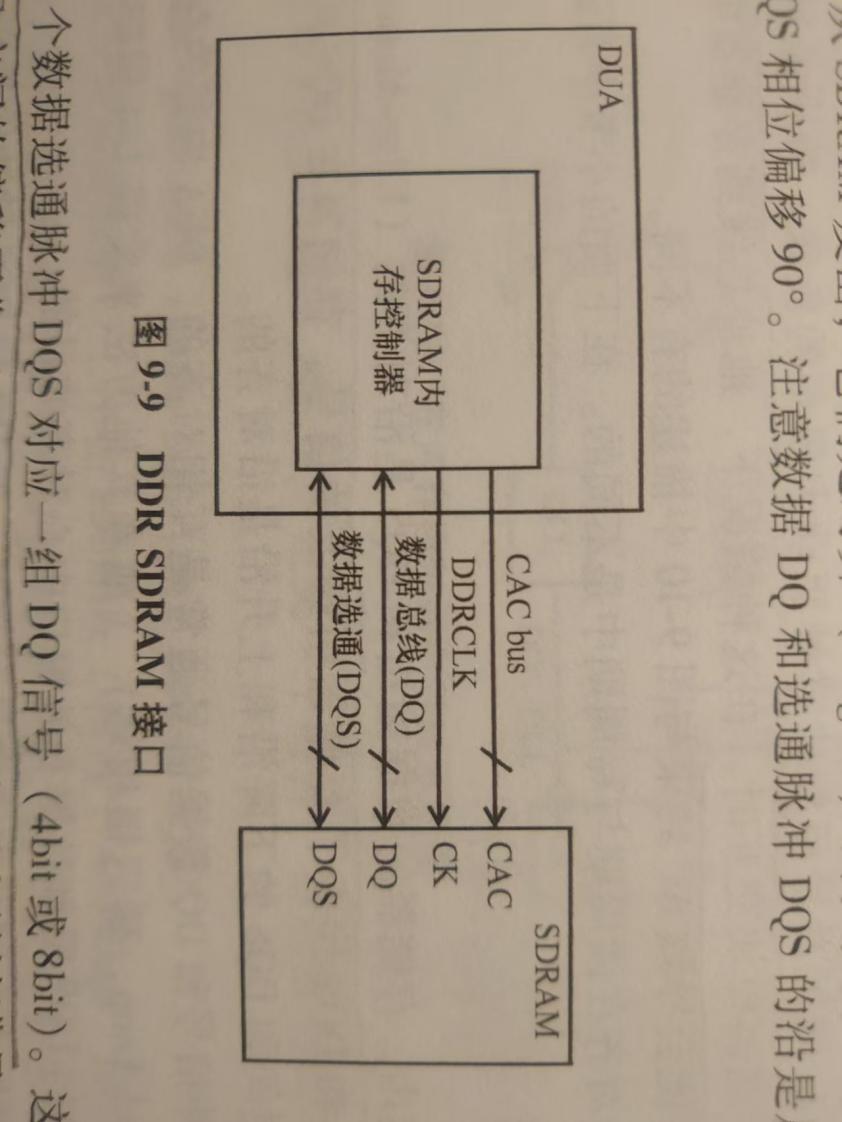
set\_input\_delay -min 1.7 -clock SRAM\_CLK [get\_pins DQ\*]

# 约束DUA输入

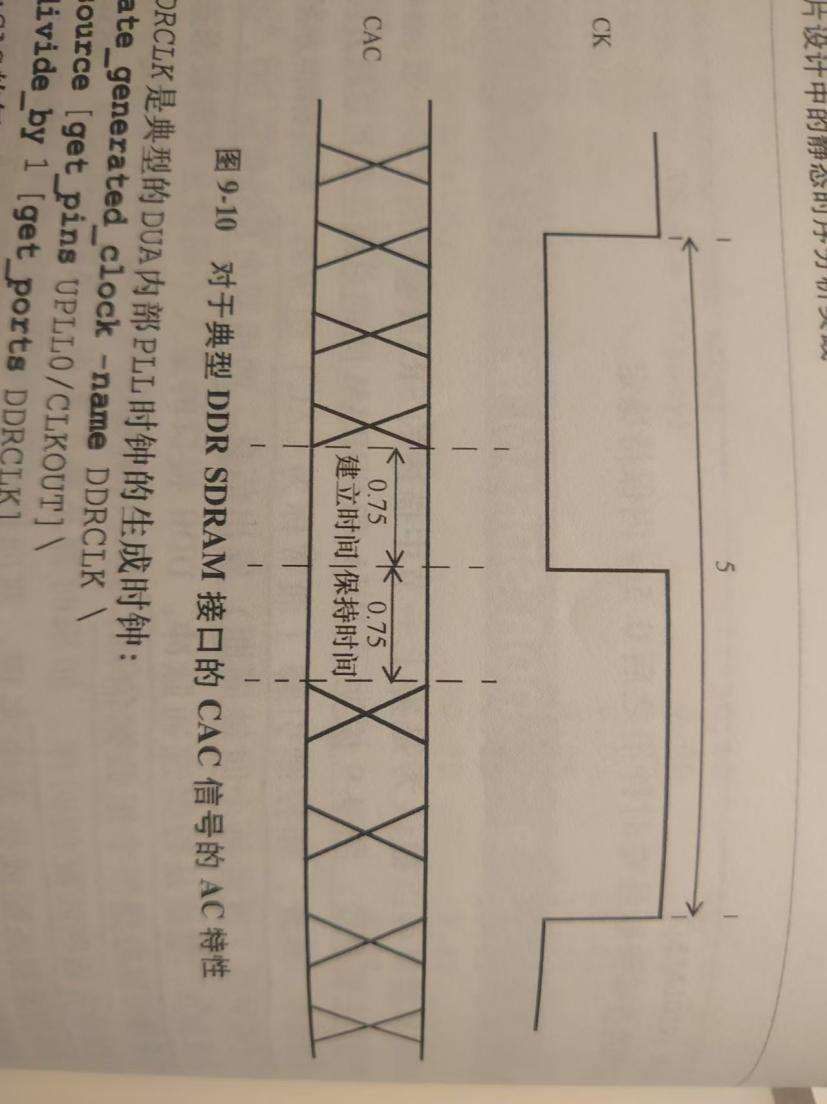
## 9.3 DDR/SDR SDRAM接口

DDR/SDR SDRAM（Double Data Rate / Single Data Rate）同样由CAC通道、双向数据通道DQ/DQS以及时钟Clock通道组成。单速率SDR使用一个时钟周期中的一个有效沿传输数据，DDR使用两个有效沿传输数据。

双向数据选通信号DQS用于读写数据的发射与捕获，所以DQ总线与DQS总线是源同步的，一位DQS对应于4位或8位DQ信号，如每8bit对应1bitDQS，64bitDQ对应8bitDQS，这样DQ通道于DQS通道之间更容易实现偏移平衡（Skew Balancing），这比64bitDQ对应1bitDQS更容易实现。



通常CAC通道对应于DDRCLK时钟



create\_generated\_clock -name DDRCLK -source [get\_pins UPLL0/CLKOUT] \

-divide\_by 1 [get\_ports DDRCLK]

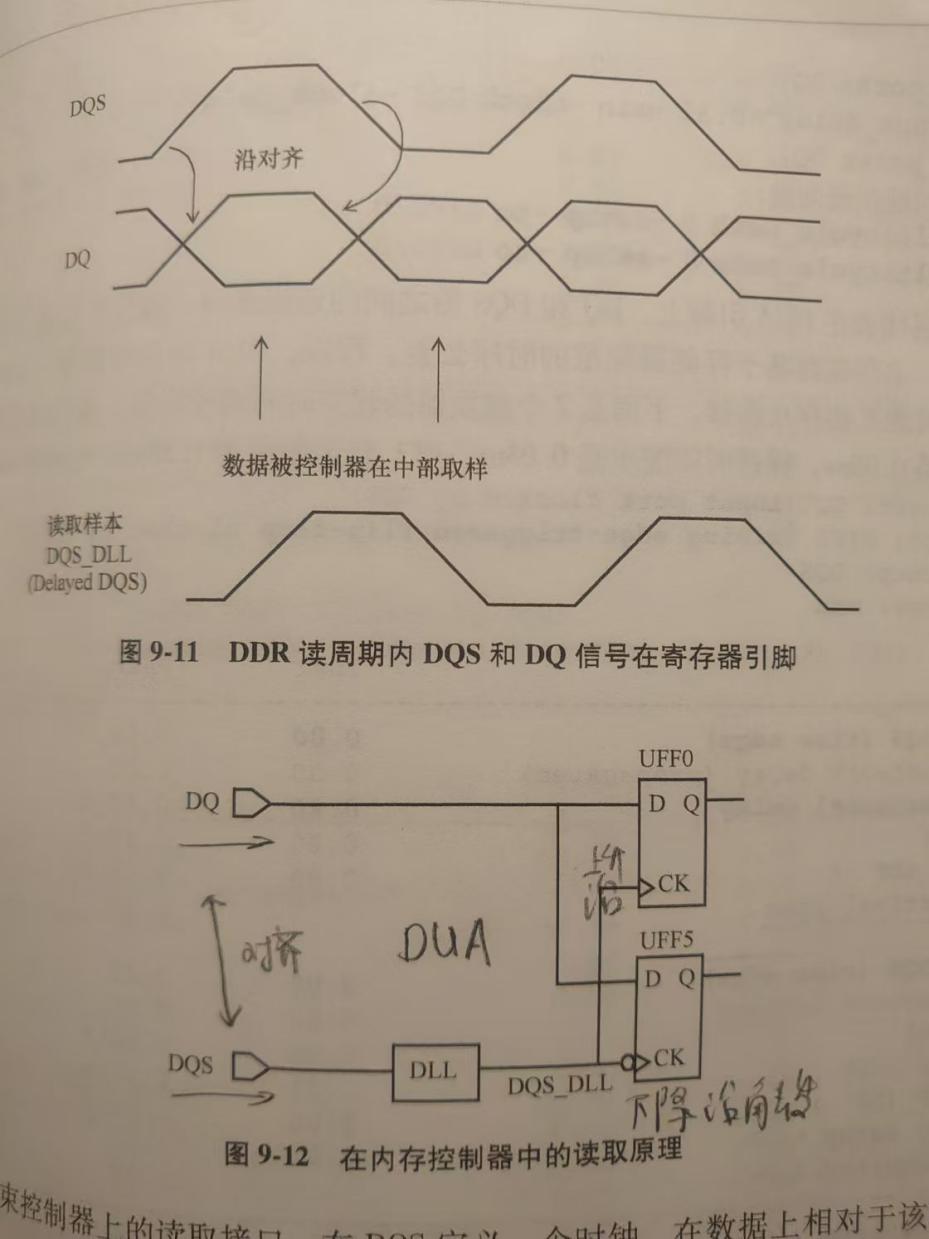
set\_output\_delay -max 0.75 -clock DDRCLK [get\_ports CAC]

set\_output\_delay -min -0.75 -clock DDRCLK [get\_ports CAC]

在某些情况下，地址总线的负载可能比时钟负载大很多，特别是没有输入缓冲器（Unbuffered）的储存器交换时，地址总线的延迟会比较大，这与上图会有不同。

9.3.1 读周期

数据从存储器输出到DUA。在一个读周期，存储器的数据输出DQ于DQS是沿对齐的（Edge Aligned）。在DUA内存控制器的输入端，DQS信号被DLL电路延迟为DQS\_DLL。DQS\_DLL对中于DQ有效数据。



set\_clock -period 5 -name DQS [get\_ports DQS]

set\_input\_delay -min -0.4 -clock DQS [get\_ports DQ]

set\_input\_delay -max 0.4 -clock DQS [get\_ports DQ]

# 约束上升沿

set\_input\_delay -0.35 -min -clock DQS -clock\_fall [get\_ports DQ]

set\_input\_delay 0.35 -max -clock DQS -clock\_fall [get\_ports DQ]

# 约束下降沿

set\_multicycle\_path 0 -setup -to UFF0/D

set\_multicycle\_path 0 -setup -to UFF5/D

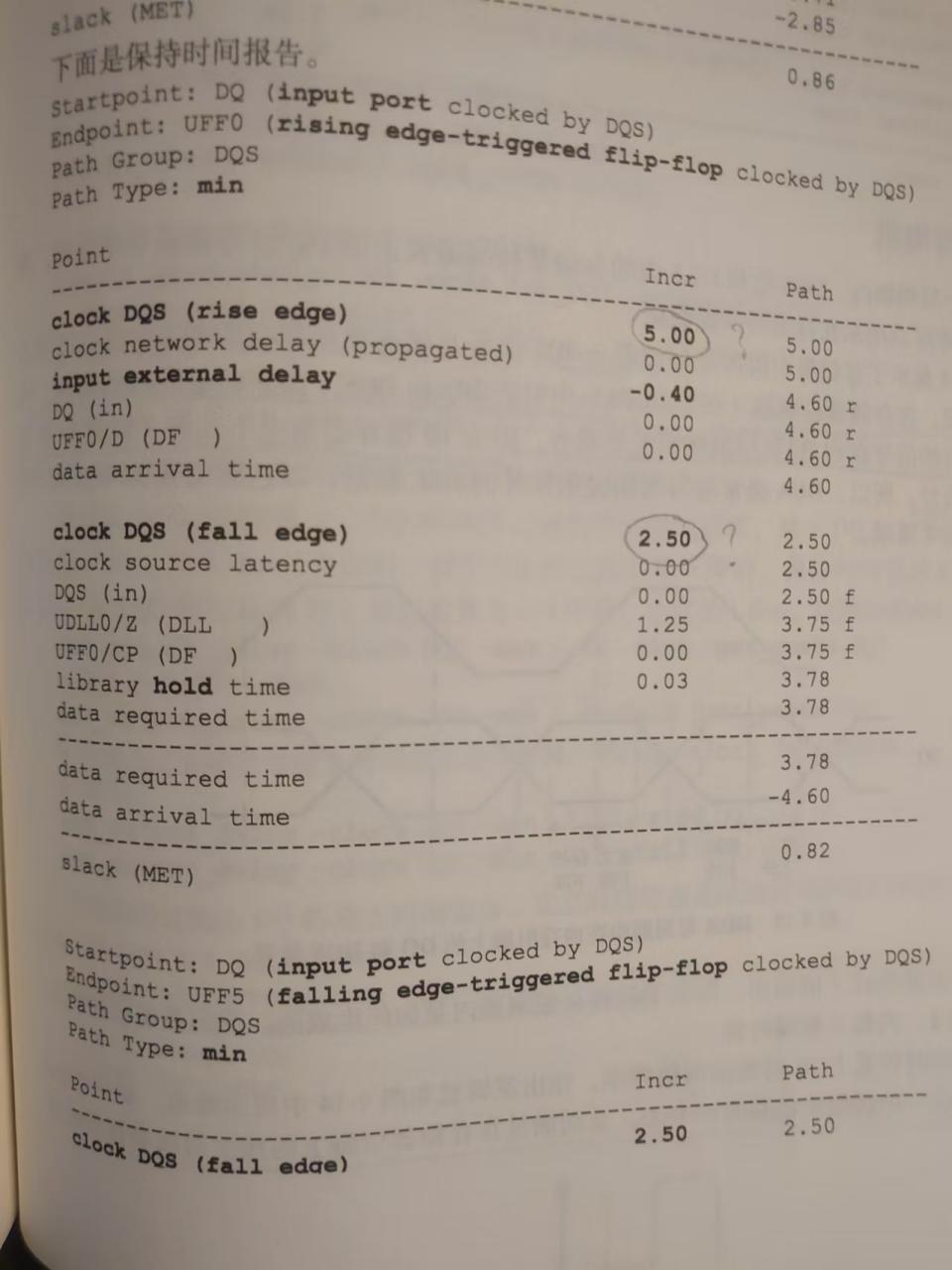
# 发射和捕获是同沿。因为DQS在DUA内部通常被延迟1/4周期，变为DQS\_DLL，# 所以setup沿不能在下一个周期而是1/4周期时刻。也就是说，**DQS发射沿为# 0ns时刻，DQS\_PLL捕获沿为1.25ns（DQS捕获沿为0ns） ；DQS发射沿为2.5ns**

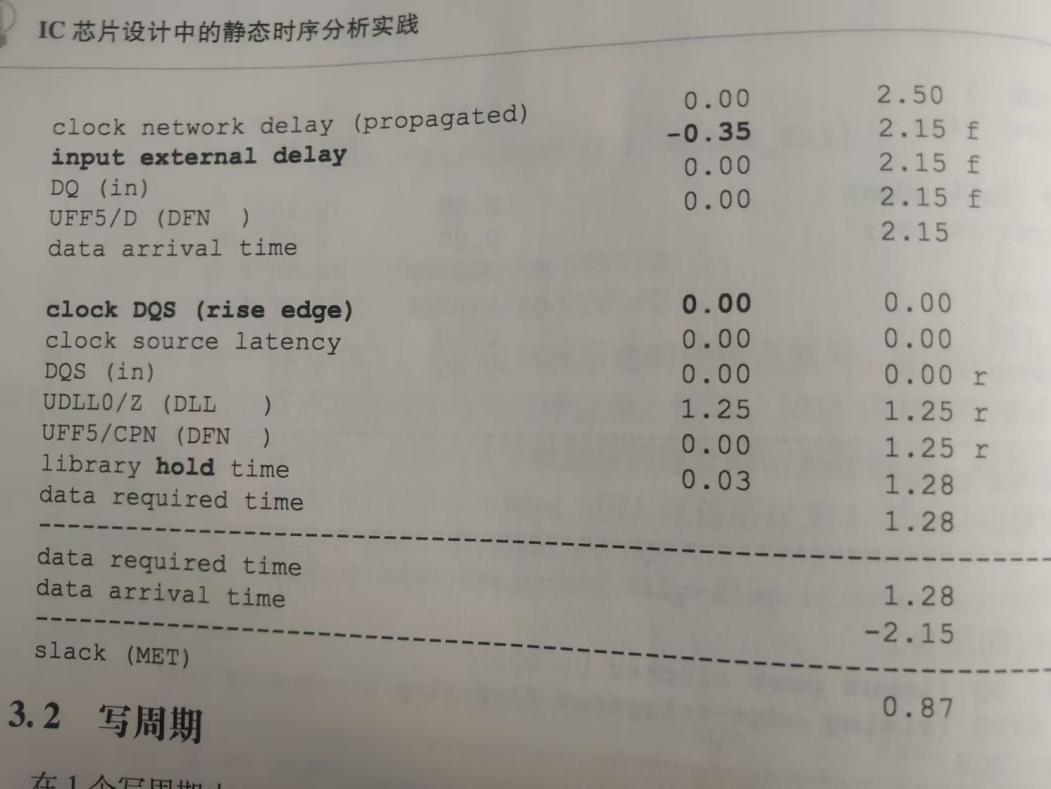
**# 时刻，DQS\_PLL捕获沿为3.75ns时刻（DQS捕获沿为2.5ns）。有两个触发器# UFF0和UFF5，要作两个建立时间检查报告。**

# 注意保持检查的时刻：**DQS本次发射沿为 0ns时刻，DQS\_PLL本次捕获沿为**

**# 1.25ns（DQS本次捕获沿为0ns）；DQS下次发射沿为2.5ns，此时下次发射数# 据不能干扰本次发射数据，所以保持时间检查时，新数据达到时间应从2.5ns**

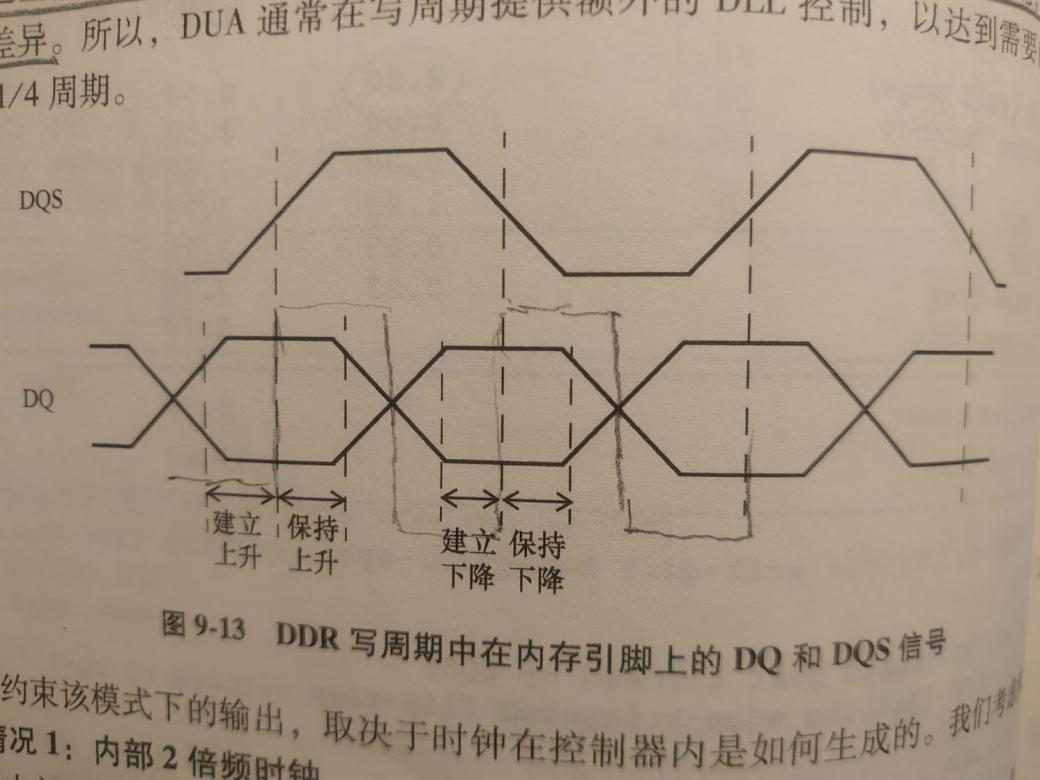
**# 时刻算起，老数据需求时间从0ns时刻算起。同样两个触发器，两份报告。**

****



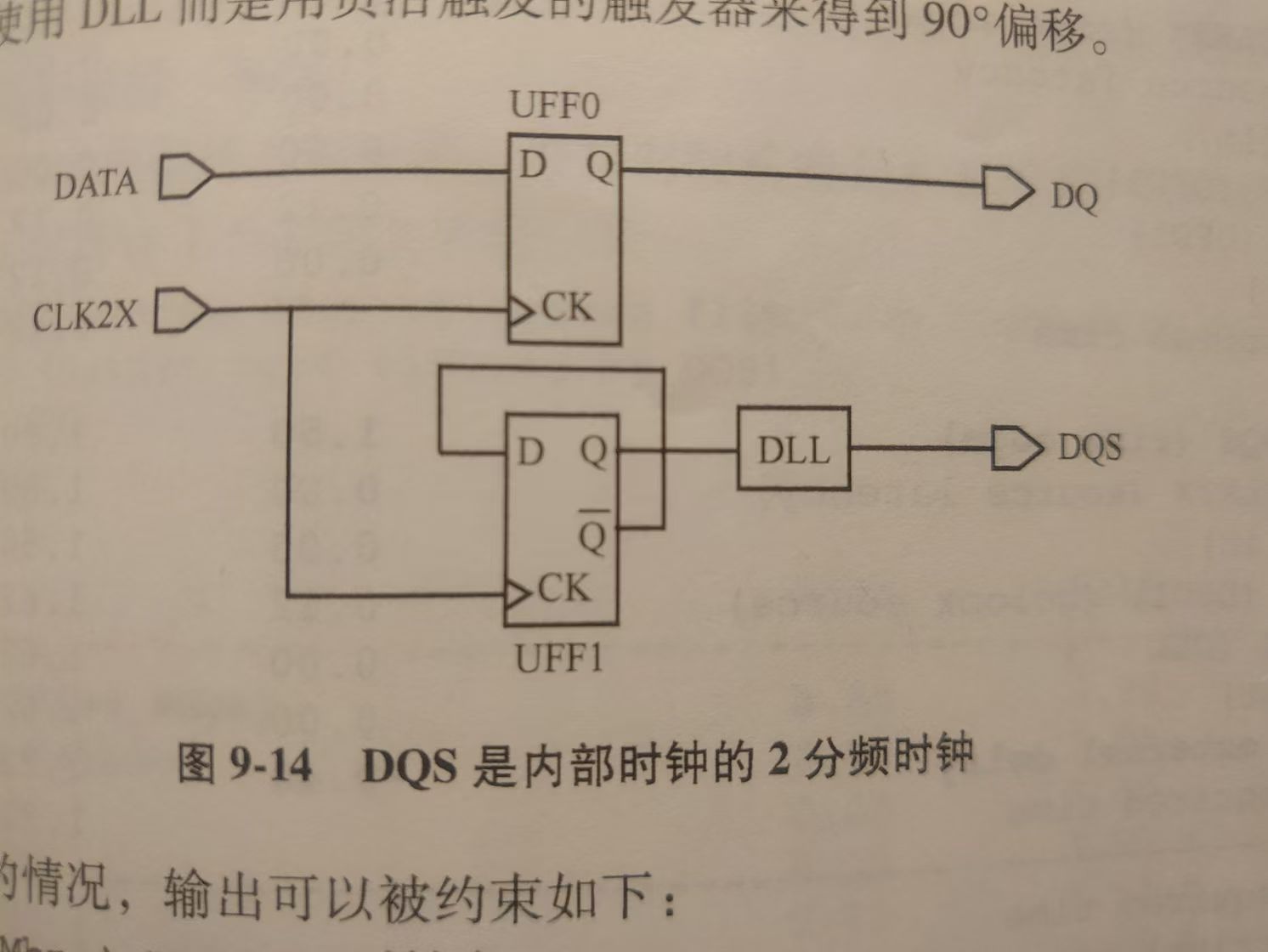
9.3.2 写周期

在一个写周期，DUA发出的DQS信号，对中于其发出的DQ信号。



情况1：DUA内部2倍频时钟

DUA存储控制器时钟频率时DQS的两倍。



create\_clock -name CLK2X -period 3 [get\_ports CLK2X]

# 333MHz CLK2X，166MHz DDR

create\_generated\_clock -name pre\_DQS -source CLK2X [get\_port UFF1/Q]

create\_generated\_clock -name DQS -source pre\_DQS -edge {1, 2, 3} -edge\_shift{1.5,\

1.5, 1.5} [get\_port DQS]

# DQS相对于pre\_DQS延迟1.5ns

设在DDR输入端，DQ对于DQS上升及下降沿的建立时间要求分别是0.25ns和0.4ns；保持时间要求分别是0.15ns和0.2ns。则对于DUA；

set\_output\_delay 0.25 -max -clock DQS -rise [get\_ports DQ]

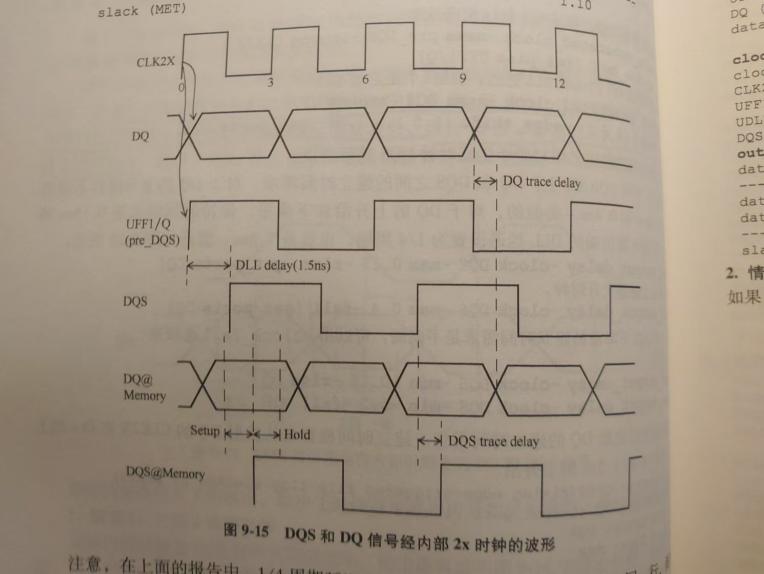
# 默认上升沿

set\_output\_delay 0.4 -max -clock DQS -fall [get\_ports DQ]

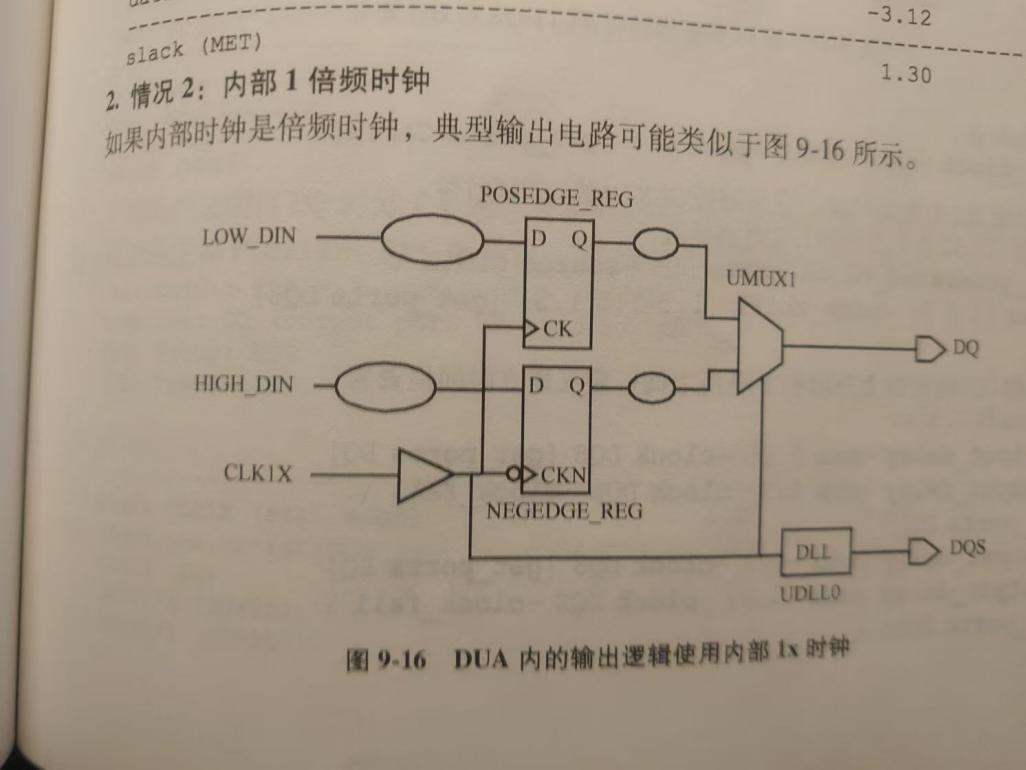
# 等效于-clock\_fall

set\_output\_delay [expr 0 - 0.15] -min -clock DQS -rise [get\_ports DQ]

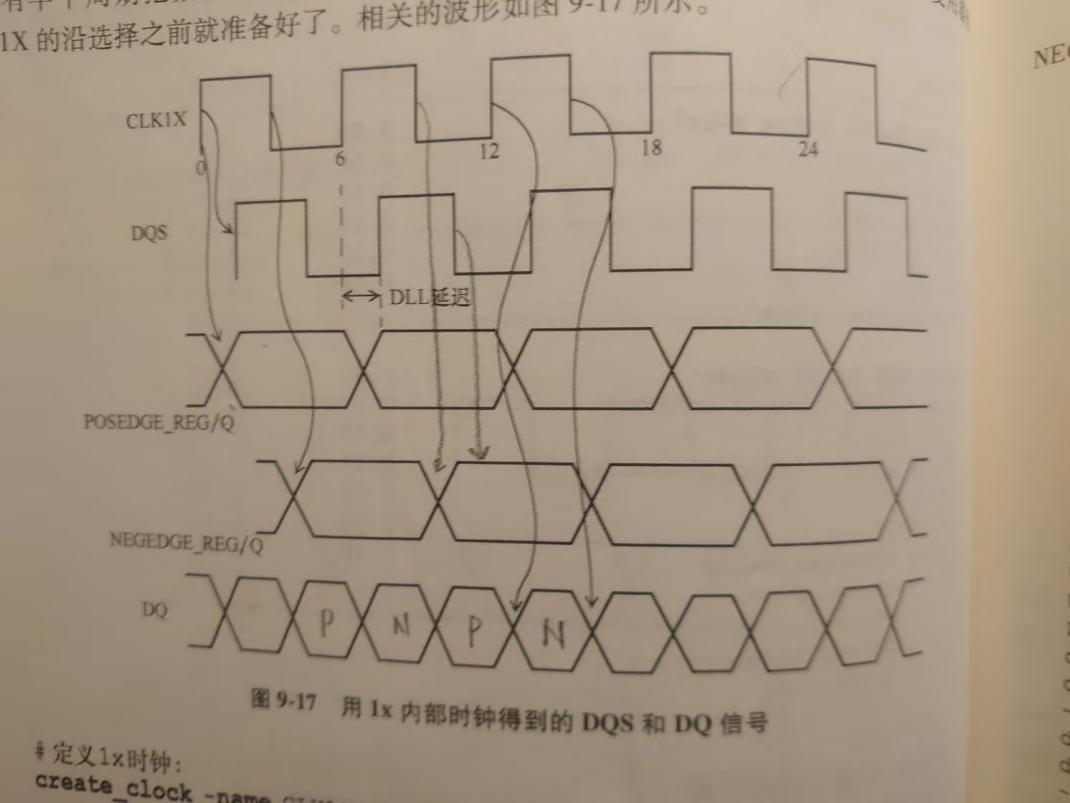
set\_output\_delay [expr 0 - 0.2] -min -clock DQS -fall [get\_ports DQ]



情况2：DUA内部1倍时钟，配正/负沿两个触发器



第一个NEGEDGE\_REG被PLK1X下降沿触发，第二个POSEDGE\_REG被PLK1X上升沿触发。CLK1X作为复用器UMUX1的选择信号，高电平时N\_REG的Q输出到DQ，低电平时P\_REG的Q输出到DQ。（只有这样才能使触发器的输出与复用的输入错开）



create\_clock -name CLK1X -period 6 [get\_ports CLK1X]

Create\_generated\_clock -name DQS -source CLK1X -edge {1, 2, 3} \

-edge\_shift {1.5, 1.5, 1,5} [get\_ports DQS]

# 设UDLL0的延迟是1.5ns，1/4周期。

set\_output\_delay 0.25 -max -clock DQS [get\_ports DQ]

set\_output\_delay 0.3 -max -clock DQS -clock\_fall [get\_ports DQ]

# 设DQ和DQS上升沿及下降沿的建立时间需求是0.25ns和0.3ns

set\_output\_delay -0.2 -min -clock DQS [get\_ports DQ]

set\_output\_delay -0.27 -min -clock DQS -clock\_fall [get\_port DQS]

# 设DQ和DQS的保持时间需求是0.2ns和0.27ns

**注意：DDR接口的DQ和DQS信号通常在读取和写入模式下使用片上终端（On-Die Termination，ODT），来减少DUA与DRAM之间传输线的阻抗不匹配性。在有ODT的情况下，STA时序模型不能保证精度。需要使用电路仿真来验证DRAM接口的时序和信号完整性。**

## 9.4 DAC接口

# **健壮性验证**

主要描述特殊STA，如时序借用，门控时钟以及非时序单元的时序检查。OCV、统计时序、功耗与时序平衡等概念。

## 10.1 片上变化（OCV）