



西安电子科技大学  
XIDIAN UNIVERSITY

# 静态时序分析 与 FPGA 综合约束

主讲人：张峻荣



## 讲述结构

- STA基础
- 建立保持时间检查
- 特殊时序检查
- 其他特殊检查



# 什么是静态时序分析 (Static Timing Analysis)

**静态时序分析 (简称STA)** 是用来验证数字设计时序的技术之一，另外一种验证时序的方法是**时序仿真**，时序仿真可以同时验证功能和时序。“时序分析”这个术语就是用来指代“静态时序分析”或“时序仿真”这两种方法之一，简单来说，时序分析的目的就是为了解决设计中的各种时序问题。

STA被称为静态的原因是其对于设计的分析是静态地执行的，并不依赖于施加在输入端口上的激励。相比之下，时序仿真则可以被视作动态地执行对设计的分析，具体过程描述如下：施加一组激励，观察在这组激励下电路行为是否符合要求，然后换一组激励再重复以上过程，以此类推。

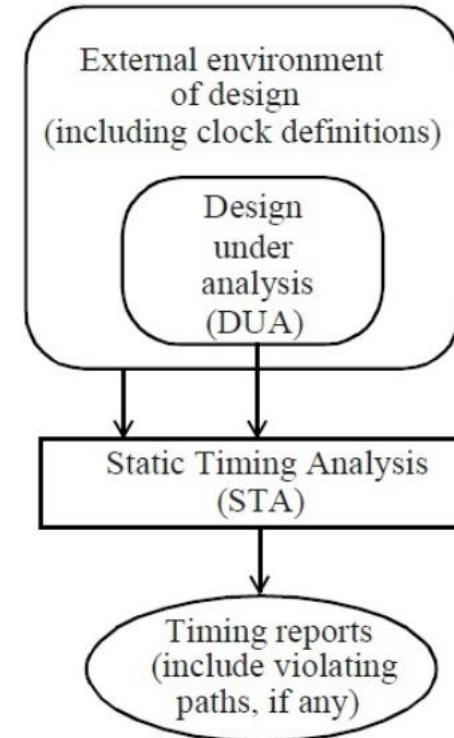
给定了一个设计、输入时钟以及外部环境，STA的目的就是验证这个设计是否能够运行在预期的速度，即这个设计可以安全地运行在给定的时钟频率下且没有时序违例。

时序分析

静态时序分析

时序仿真

时序仿真=功能仿真？



## 为什么使用静态时序分析?

STA更重要的意义在于：**整个设计只需要被分析一次**，就可以对所有情况下设计中的全部路径进行所需的时序检查。因此，STA是能够被用来验证设计时序的一种**完全且详尽**的方法。

其他时序分析方法例如时序仿真则只能验证到被当前激励执行到的那部分时序路径。基于时序仿真的验证完备性取决于施加激励的完备性。如果使用时序仿真来验证一个千万门级别的设计，速度将会非常慢，并且实际上也无法充分验证。因此，想要基于时序仿真 的方法来进行详尽的时序验证是非常困难的。

相比之下，STA则提供了一种更快更简单的方法去分析并检查设计中的全部时序路径。

DUA通常使用硬件描述语言，例如VHDL或者Verilog HDL；  
外部环境，包括时钟定义，通常使用SDC或等价格式的时序约束进行描述；  
时序报告通常以ASCII格式呈现，一般报告中会有许多列，每一列都会显示路径延时的一个属性。



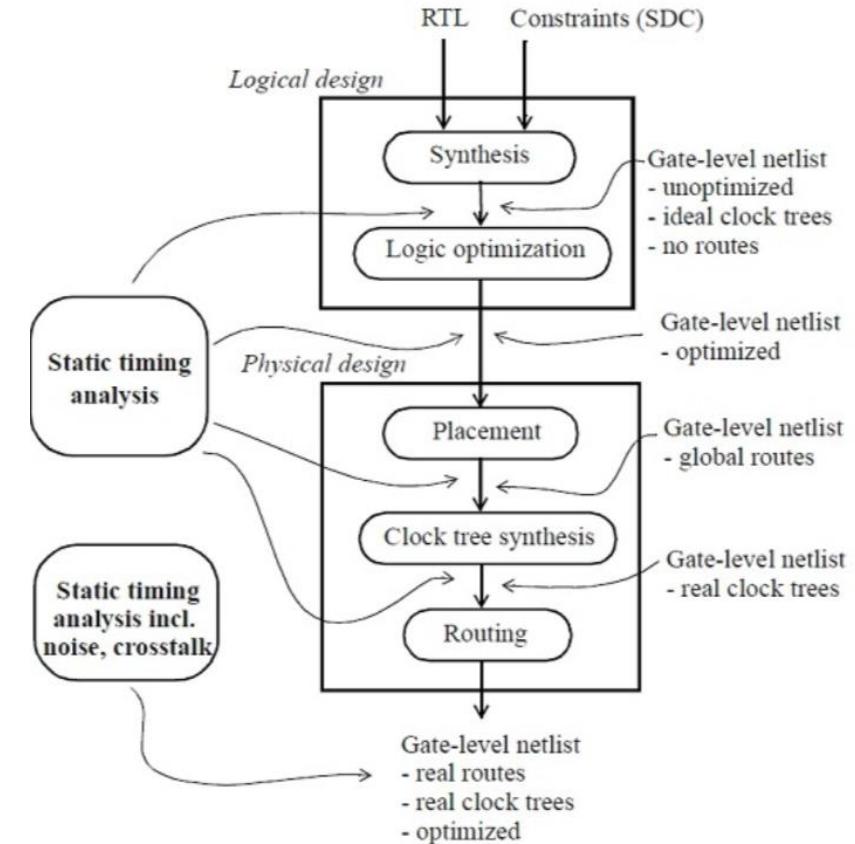
## STA使用的阶段

在CMOS数字设计流程中，STA会在实现的**各个阶段**里被使用到。

STA很少在RTL级完成，因为在这一抽象层级上，验证设计的功能更为重要，而非时序。同样，由于块（block）的描述处于行为级，因此时序信息也并非都是可用的。**一旦将RTL级的设计综合到门级，就可以使用STA来验证设计的时序。** STA也可以在执行逻辑优化之前运行，其目标是确定最差或关键的时序路径。可以在逻辑优化后重新运行STA，以查看是否还有剩余的时序违例路径需要优化，或者确定关键路径。

## 时钟域

芯片中时钟域大部分都是全局异步，局部同步，**静态时序分析（STA）主要对同步电路进行分析，对于跨时钟域的路径使用虚假路径进行约束**，但是并不是说STA不适用于异步设计。





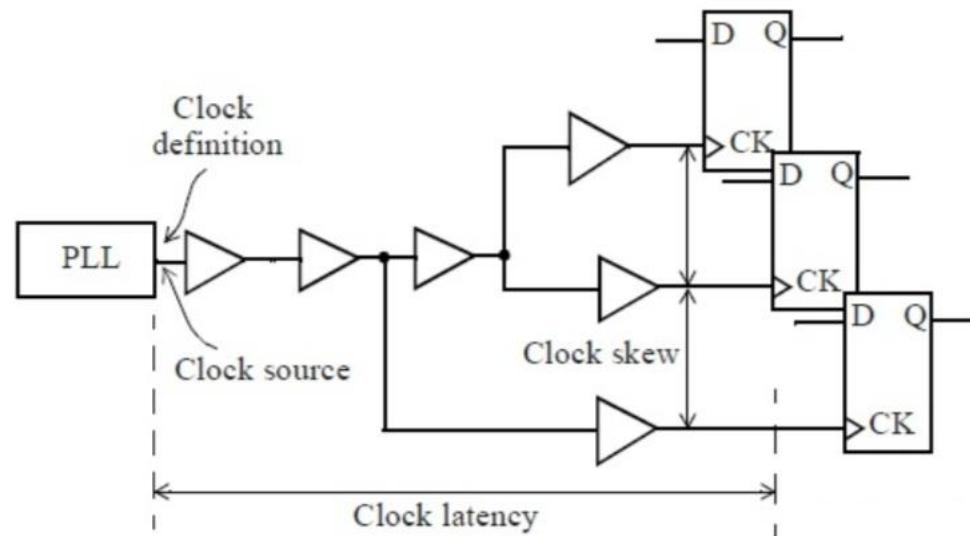
## STA的局限性

- **复位顺序**: 检查所有触发器在异步或同步复位后是否都复位为所需的逻辑值，这是无法使用静态时序分析来检查的。芯片可能不会退出复位状态
- **未知态X的处理**: STA技术仅处理逻辑0和逻辑1（或高电平/低电平）的逻辑域，或者是上升沿和下降沿的逻辑域。设计中的未知态X导致不确定的值在整个设计中传播，这也是无法使用STA进行检查。
- **PLL设置**: PLL的配置可能未被正确加载或设置。
- **跨异步时钟域**: STA不检查是否使用了正确的时钟同步器，需要其他工具来确保在任何跨异步时钟域的地方都有正确的时钟同步器。
- **IO接口时序**: 可能仅根据STA约束无法规定IO接口要求。
- **模拟模块和数字模块之间的接口**: 由于STA不处理模拟模块，因此验证方法需要确保这两种类型的模块之间的连接正确。
- **伪路径 (false path)**: 静态时序分析会验证通过逻辑路径的时序是否满足所有约束，如果通过逻辑路径的时序不符合要求的规范，则标记违例。在许多情况下，即使逻辑可能永远无法传播通过该路径，STA也会将该逻辑路径标记为时序违例路径。
- **FIFO指针不同步**: 当两个预期要同步的有限状态机实际上不同步时，STA无法检测到该问题。
- **时钟同步逻辑**: STA无法检测到时钟生成逻辑与时钟定义不匹配的问题。
- **跨时钟周期的功能行为**: STA无法建模或仿真跨时钟周期变化的功能行为。



## 信号偏斜 (skew) / 时钟延迟 (latency)

**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。



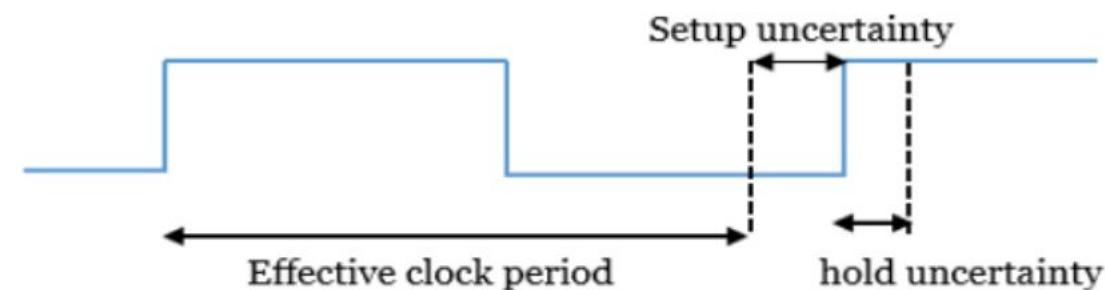
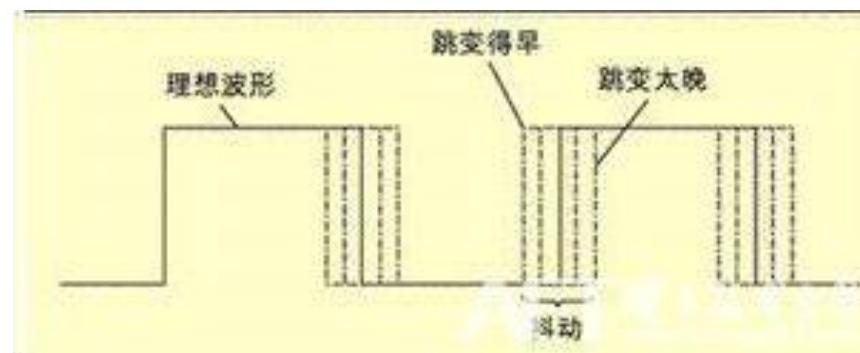
在理想的时钟树中，默认情况下时钟偏斜为0ps



## 时钟抖动 (Jitter)

抖动的定义为“信号的定时事件与其理想位置之间的偏差”。

在理想情况下，一个频率固定的完美的脉冲信号（以1MHz为例）的持续时间应该恰好是1us，每500ns有一个跳变沿。但不幸的是，这种信号并不存在。如图1所示，信号周期的长度总会有一定变化，从而导致下一个沿的到来时间不确定。这种不确定就是**抖动** (jitter)。



## 时钟不确定度 (Clock Uncertainty)

时钟不确定度是时钟沿的实际到达时间相对于理想到达时间的偏差。在理想模式下，时钟信号可以同时到达所有时钟引脚。但事实上，完美是不可能实现的，应该引入一些不确定的变量来模拟真实的传播模式。如 Skew , Jitter , PVT , Crosstalk (串扰) or any other pessimism.



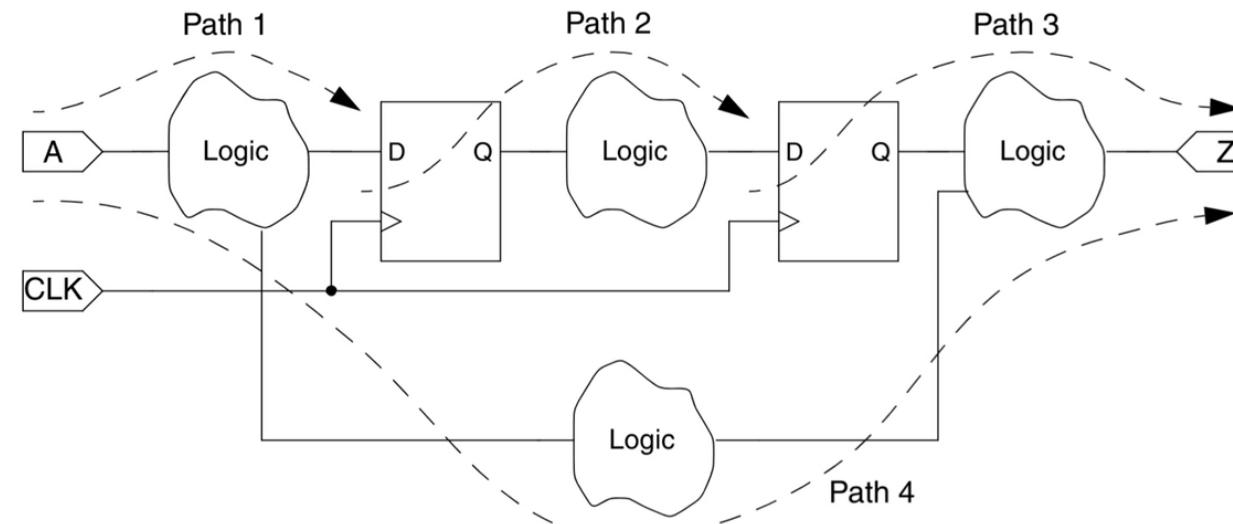
## 时序路径 (Timing path)

时序路径是指设计中数据信号传播过程中所经过的逻辑路径。每一条时序路径都存在与之对应的一个始发点和一个终止点。

时序路径的起点：① clock pin ② input port

时序路径的终点：① data input pin (D触发器的D端) ② output port

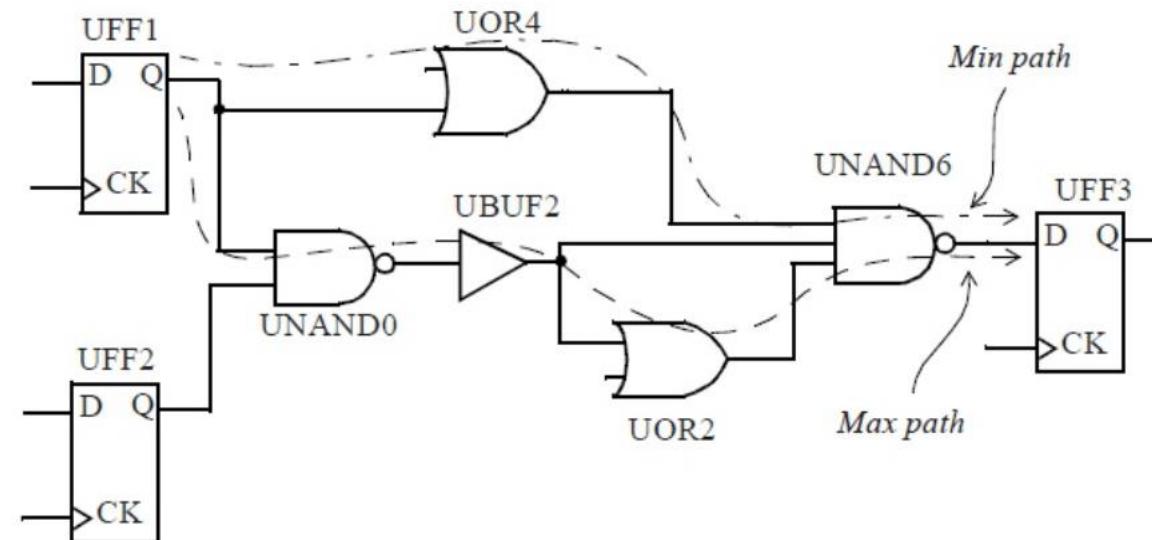
一共有四种路径，可以保证覆盖到所有时序路径。





## 最大时序路径/最小时序路径/关键路径

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。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。



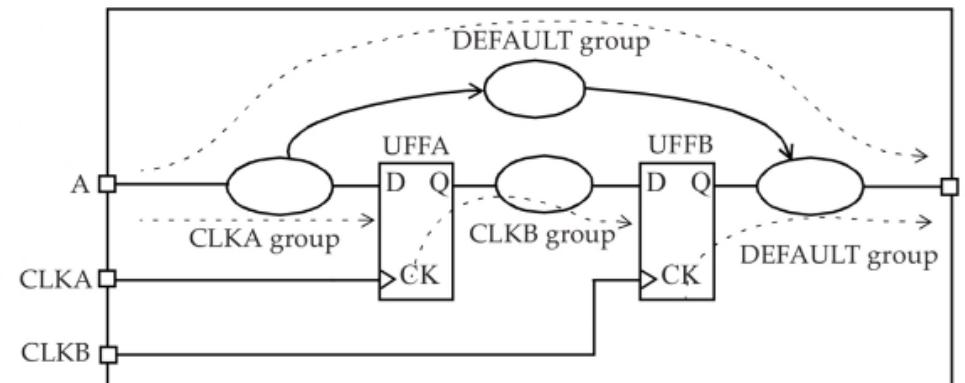
**关键路径：**设计中从输入到输出经过的最大延时的组合逻辑路径。



## 时序路径组

时序路径根据路径的**终点**被分到不同的时序路径组中，因此，每个时钟都有一组与其相关的路径。

除此之外，还有一个默认路径组，其中包括所有**非时钟(异步)**路径。



## 时序域 (Clock Domains)

芯片中时钟域大部分都是全局异步，局部同步，**静态时序分析 (STA) 只对同步电路进行分析**，对于跨时钟域的路径使用虚假路径进行约束。

## 工作条件 (Operating Conditions)

静态时序分析通常在特定的工作条件下进行。

工作条件被定义为**工艺 (Process) 、电压 (Voltage) 和温度 (Temperature)**(PVT)的组合。

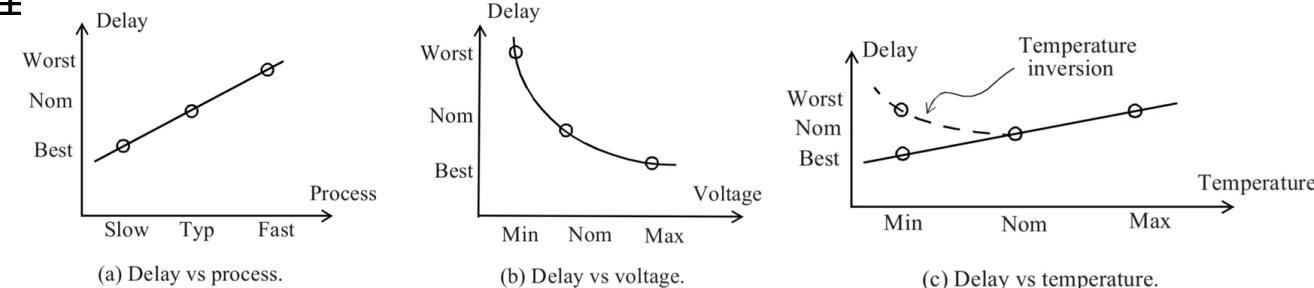
单元延迟和互连延迟是基于特定的操作条件下计算的。



## 工作条件 (Operating Conditions) – 续

半导体代工厂为数字设计提供了三种制造工艺模型

- Slow process model
- Typical process model
- Fast process model



慢速和快速工艺模型代表了制造过程的**极端角** (extreme corners)。

为了实现鲁棒性的设计，设计会在制造工艺的极端角以及温度和电压的极端环境中进行验证。

□ **WCS (Worst-Case Slow):** Process is slow, temperature is highest (say 125C) and voltage is lowest (say nominal 1.2V minus 10%).

□ **TYP (Typical):** Process is typical, temperature is nominal (say 25C) and voltage is nominal (say 1.2V).

□ **BCF (Best-Case Fast):** Process is fast, temperature is lowest (say -40C) and voltage is highest (say nominal 1.2V plus 10%).

**高温低电压 (SS)** :延迟最大，速度最慢，用于检查 setup

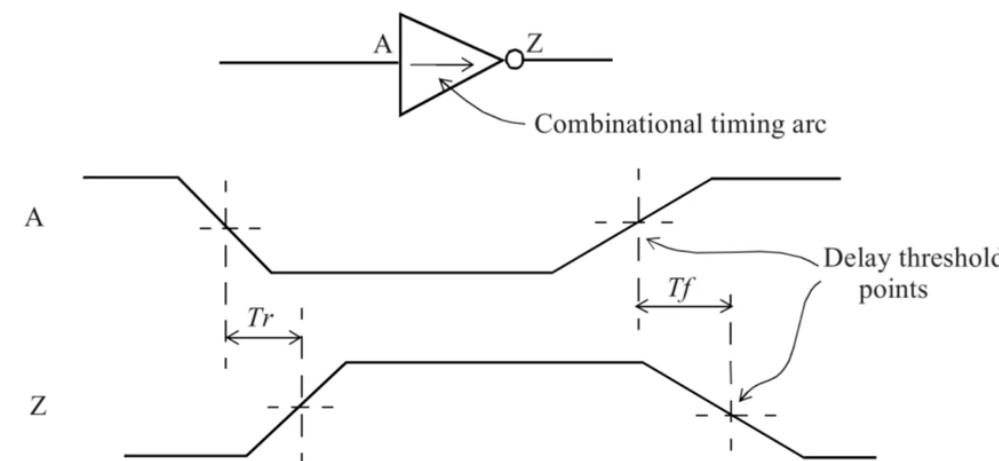
**低温高电压 (FF)** :延迟最小，速度最快，用于检查 hold



## 时序模型 (Timing Model)

以反相器为例，通过反相器的 timing arc 的延迟取决于两个因素：

- **输出负载** (output load) , 即反相器器输出引脚处的电容负载
- 信号在输入端的**转换时间** (transition time)



延迟与负载电容有直接的关系，**负载电容越大，延迟越大**。

在大多数情况下，**延迟随输入转换时间的增加而增加**。

时序模型主要分为两类：线性时序模型和非线性时序模型（不重要，实际查表）



# 建立/保持时间检查

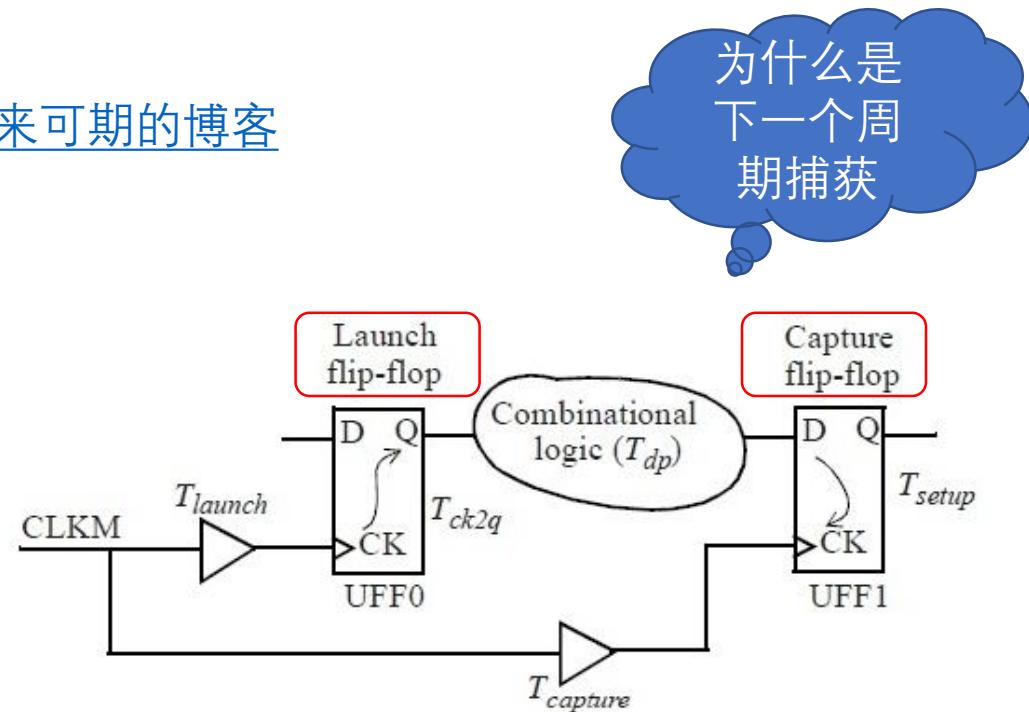
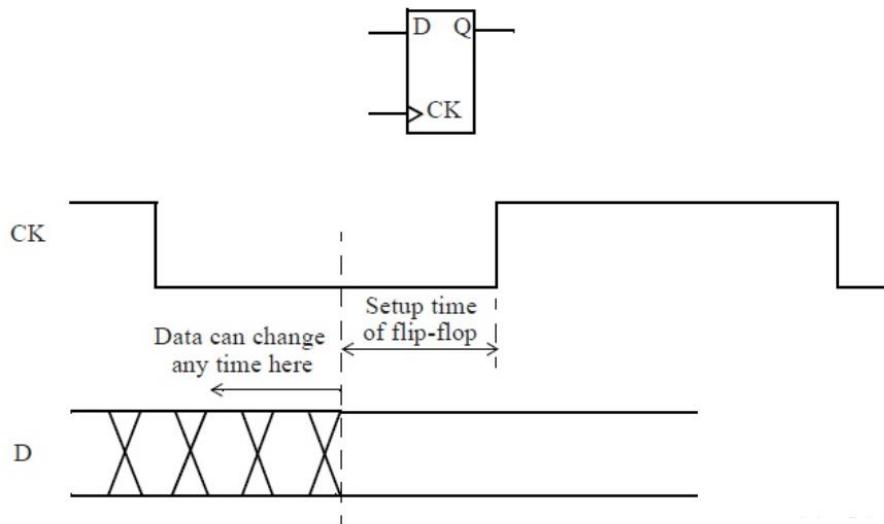
- 建立时间检查
- 保持时间检查



## 什么是建立时间？

在时钟的有效沿到达触发器之前，数据应在一定时间内保持稳定，即触发器的建立时间，该要求将确保数据可靠地被捕获到触发器中。

[为什么D触发器需要建立时间与保持时间\\_日拱一卒\\_未来可期的博客](#)

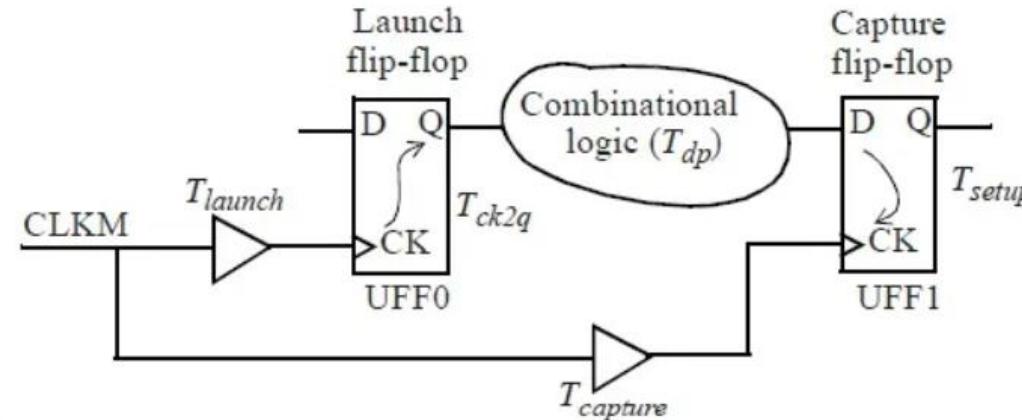


通常，有一个**发起触发器**（用于发起数据的触发器）和**捕获触发器**（用于捕获数据的触发器），这个捕获触发器的建立时间要求必须满足。**建立时间检查将验证从发起触发器到捕获触发器的最长（或最大）路径，这两个触发器的时钟可以相同也可以不同。建立时间检查是从发起触发器中时钟的第一个有效沿到捕获触发器中时钟后面最接近的那个有效沿。**建立时间检查将确保上一个时钟周期发起的数据准备好在一个周期后被捕获。

为什么是  
下一个周  
期捕获



## 一个简单的例子

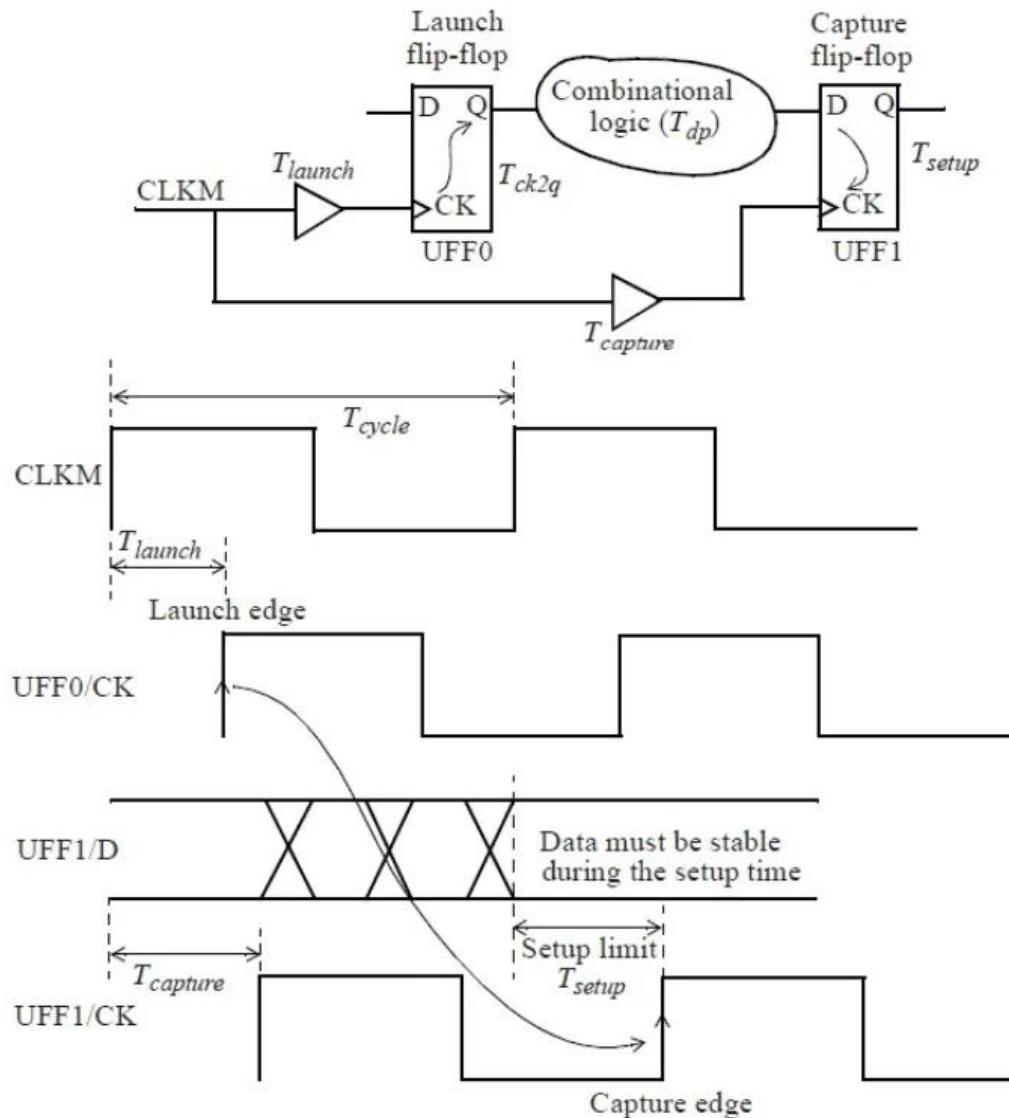


$T_{launch}$ :发起触发器UFF0的时钟树延迟  
 $T_{dp}$ :组合逻辑数据路径的延迟  
 $T_{cycle}$ :时钟周期  
 $T_{capture}$ :是获触发器UFF1的时钟树延迟

- 发起触发器和捕获触发器具有相同的时钟；
- 时钟CLKM的第一个上升沿在  $T_{launch}$  时间后出现在发起触发器的时钟引脚上，由该时钟沿发起的数据出现在触发器UFF1的D引脚的所需时间为 **Data arrival time =  $T_{launch} + T_{ck2q} + T_{dp}$** 。
- 时钟CLKM的第二个上升沿（通常在一个周期后检查建立时间）出现在捕获触发器UFF1的时钟引脚上的时间为**Clock arrival time =  $T_{cycle} + T_{capture}$** 。
- 这两个时间之差必须大于触发器UFF1的建立时间要求，以确保触发器UFF1可靠地捕获数据。

**Clock arrival time - Data arrival time > Tsetup**

**Data arrival time < Clock arrival time - Tsetup**



## 建立时间检查：

$$T_{\text{launch}} + T_{\text{ck}2q} + T_{\text{dp}} < T_{\text{capture}} + T_{\text{cycle}} - T_{\text{setup}}$$

## 时钟偏斜：

$$\text{Clock\_skew} = T_{\text{capture}} - T_{\text{launch}}$$

## 建立时间余量：

$$\begin{aligned} \text{Setup Slack} &= \text{Clock arrival time} - T_{\text{setup}} - \text{Data arrival time} \\ &= T_{\text{cycle}} + T_{\text{capture}} - T_{\text{setup}} - (T_{\text{launch}} + T_{\text{ck}2q} + T_{\text{dp}}) \end{aligned}$$

## 数据需要保持稳定的最短时间：

$$\text{Data require time} = \text{Clock arrival time} - T_{\text{setup}} = T_{\text{cycle}} + T_{\text{capture}} - T_{\text{su}}$$

建立时间检查受到 $-max$ 的约束，因此建立时间检查始终使用**最长或最大的时序路径**。出于同样的原因，通常在**延迟最大的慢工艺角 (slow corner)**下执行建立时间检查。



Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKM)  
Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKM)

Path Group: CLKM

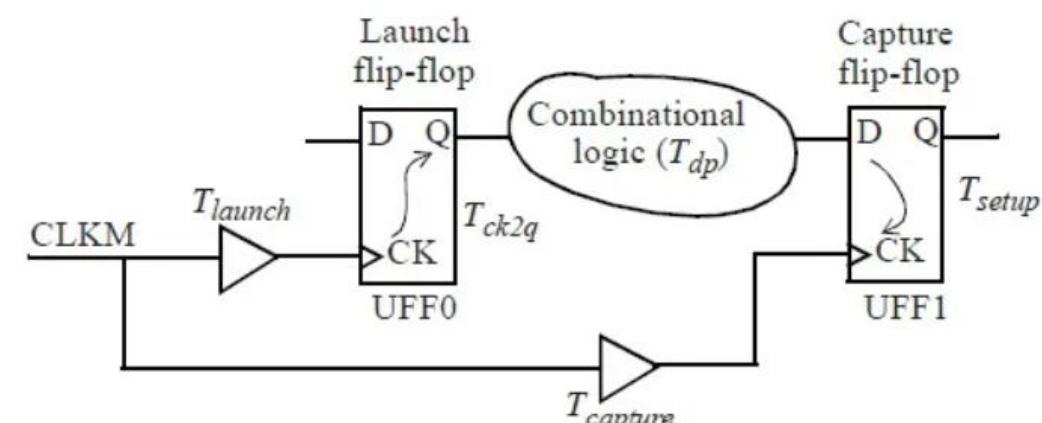
Path Type: max

Point

根据endpoint划分  
Max代表建立时间分析

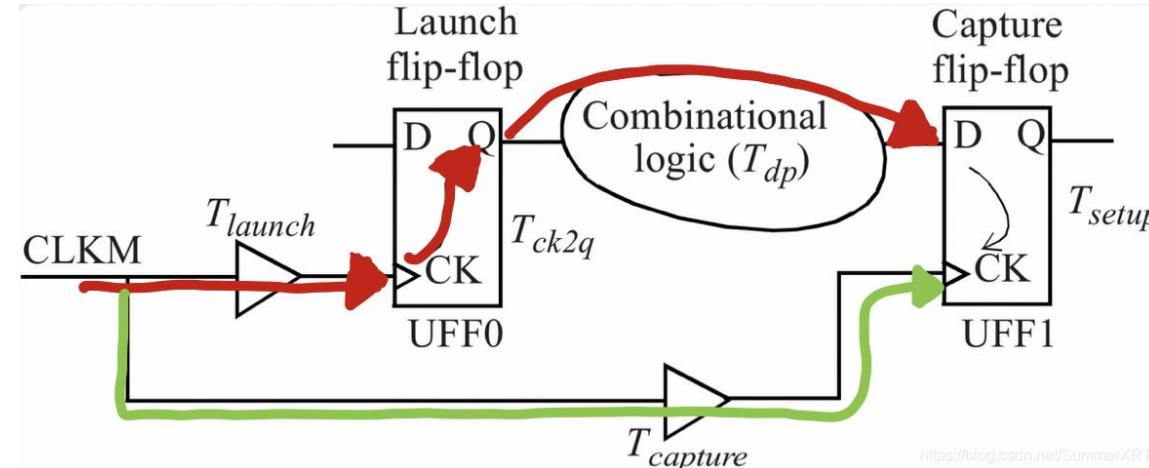
元件 (路径) 延迟  
Incr Path 总延迟

clock CLKM (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
UFF0/CK (DFF )	0.00	0.00 r
UFF0/Q (DFF ) <-	0.16	0.16 f
UNOR0/ZN (NR2 )	0.04	0.20 r
UBUF4/Z (BUFF )	0.05	0.26 r
UFF1/D (DFF )	0.00	0.26 r
data arrival time		0.26
clock CLKM (rise edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00
clock uncertainty	-0.30	9.70
UFF1/CK (DFF )		9.70 r
library setup time	-0.04	9.66
data required time		9.66
data required time		9.66
data arrival time		-0.26
slack (MET)		



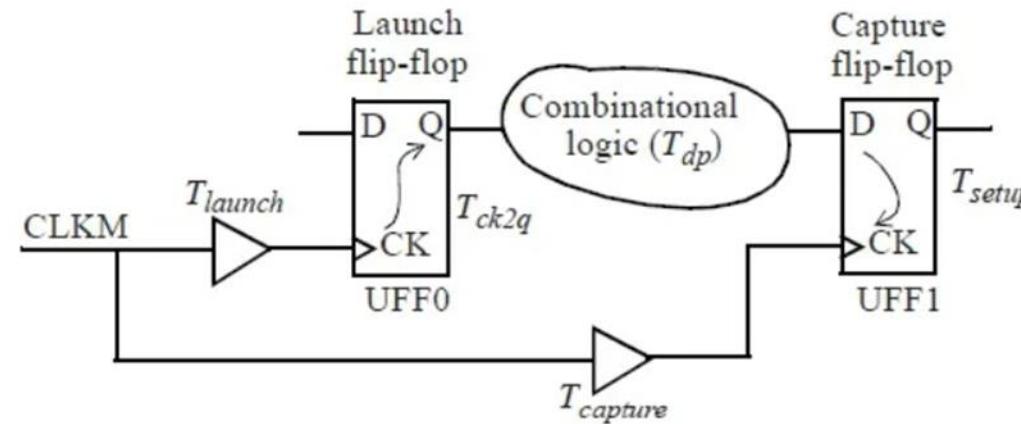


## 解决建立时间不满足



当建立时间不满足时，即  $T_{setup}$  值要大于实际的  $T_{cycle} + T_{skew} - T_{dp} - T_{ck2q}$ ，为了满足建立时间：

- 可以通过改进工艺，采取具有更小的  $T_{setup}$  值的芯片；
- 加强约束，重新进行综合，对违规的路径进行进一步的优化，但是一般效果可能不是很明显；
- 采用延迟更低的触发器，降低  $T_{ck2q}$ ，即减小 D->Q 的传输延迟  $T_{co}$ ，更换更快的器件，使用更先进的器件库；
- 增大时钟周期  $T_{cycle}$ ，但这会降低电路的性能，并且时钟一般是在项目最初的时候决定的，这个时候很难再改变。
- 增大时钟歪斜  $T_{skew} = T_{capture} - T_{launch}$ ，如果时钟歪斜  $T_{skew}$  为正，对 setup 是有利的，对 hold 是有害；
- 尽量减小两个触发器之间的组合逻辑电路的使用，从而降低  $T_{dp}$ ，即减小组合逻辑延时，主要是关键路径的处理。包括拆分组合逻辑，插入寄存器使其流水、重定时等。
- 优化布局布线，减小传输的延时



已知量：

$T_{clkM} = 10\text{ns}; T_{launch} = 2\text{ns}; T_{capture} = 3\text{ns}; T_{dp} = 3\text{ns}; T_{su} = 0.5\text{ns}; T_{ck2q} = 1\text{ns};$

求： Setup Slack = ?

$F_{max} = ?$

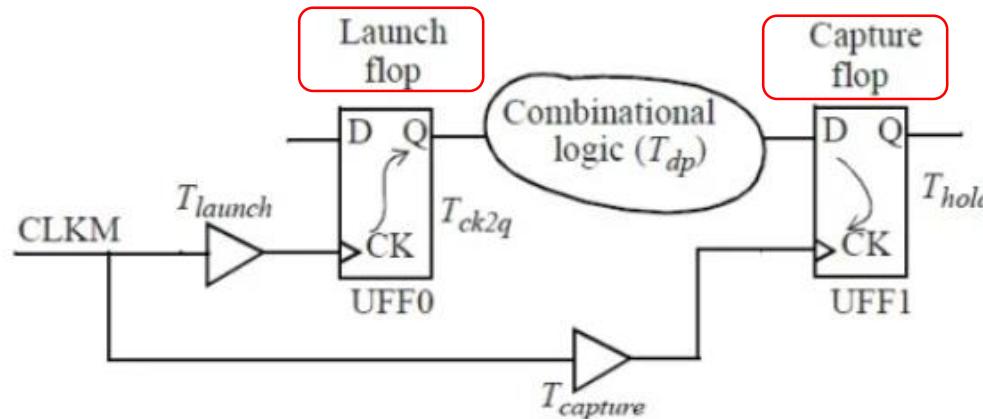
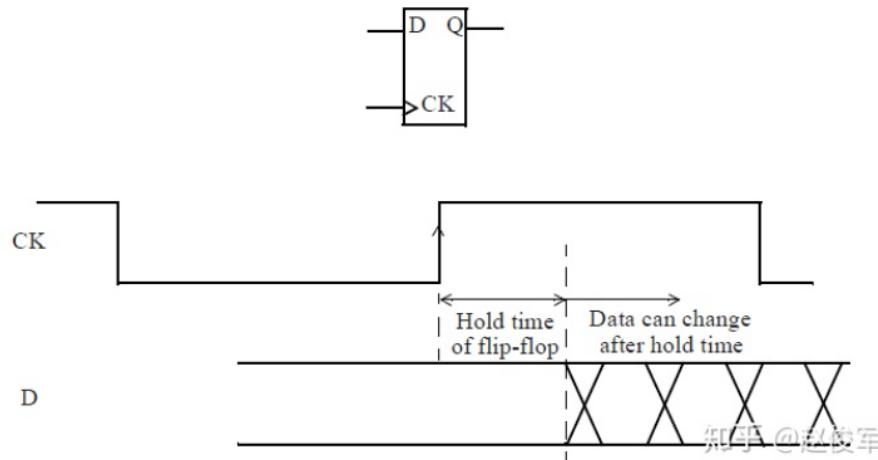
$$\text{Setup Slack} = 10 + 3 - 0.5 - (2 + 1 + 3) = 6.5 \text{ ns}$$

$$F_{max} = 1/(T_{clkM} - \text{Setup Slack}) = 1/3.5$$



## 什么是保持时间？

时钟有效沿到来之后的某段时间内，数据必须稳定，这段时间成为保持时间，用  $T_{hold}$  或者  $T_h$  表示。

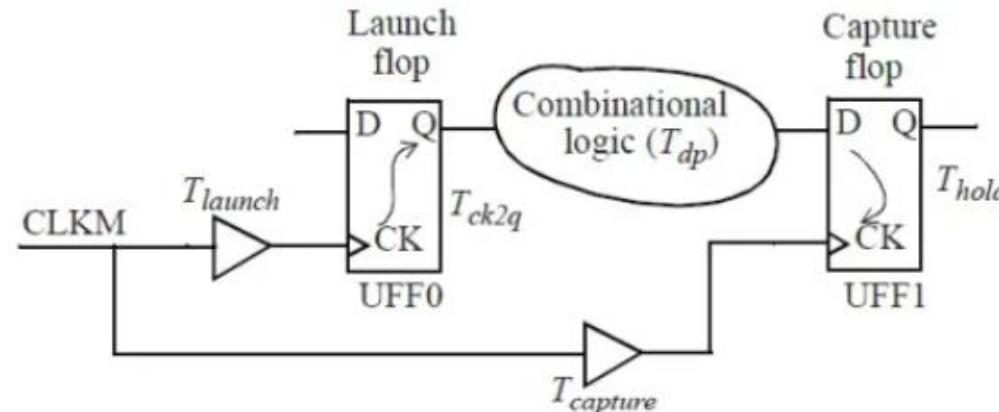


就像建立时间检查一样，是在**发起触发器（发起数据的触发器）**和**捕获触发器（捕获数据的触发器）**之间进行保持时间检查的。这两个触发器的时钟可以相同也可以不同，**保持时间检查从发起触发器时钟的一个有效沿到捕获触发器中相同的时钟沿**。因此，保持时间检查与时钟周期无关，保持时间检查会在捕获触发器时钟的每个有效沿上执行。

为什么是  
同一个时  
钟沿



## 一个简单的例子



$T_{launch}$ :发起触发器UFF0的时钟树延迟

$T_{dp}$ :组合逻辑数据路径的延迟

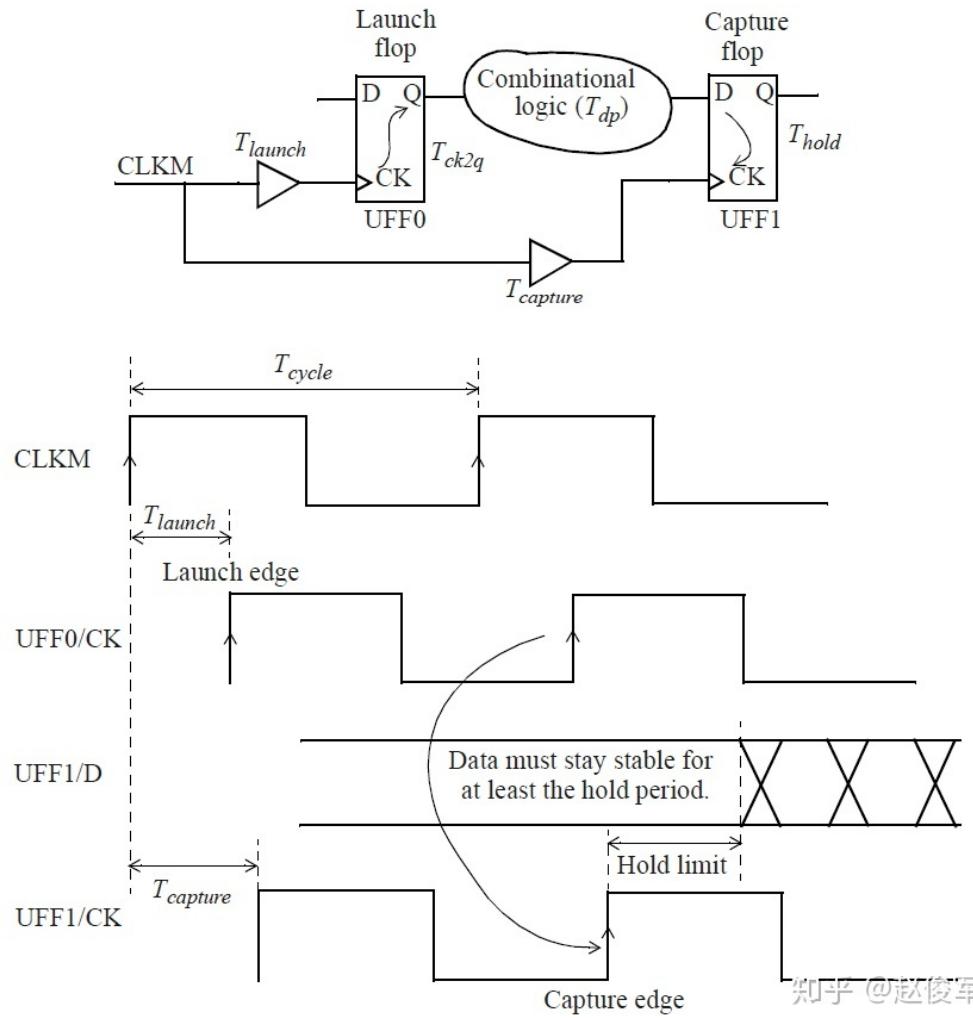
$T_{cycle}$ :时钟周期

$T_{capture}$ :是获触发器UFF1的时钟树延迟

- CLKM 时钟上升沿发起的数据需要 **Data arrival time =  $T_{launch} + T_{ck2q} + T_{dp}$**  时间到达捕获触发器UFF1的D引脚。
- 时钟的同一边沿需要 **Clock arrival time =  $T_{capture}$**  时间才能到达捕获触发器的时钟引脚，目的是使捕获触发器在下一个时钟周期捕获来自发起触发器的数据。
- 如果在同一时钟周期内捕获数据，则捕获触发器中的预期数据（来自上一个时钟周期）将被覆盖，因此保持时间检查旨在确保捕获触发器中的目标数据不会被覆盖。
- 保持时间检查可验证这两个时间之差（**捕获触发器的数据到达时间和时钟到达时间**）必须大于捕获触发器的**保持时间**，这样触发器上的数据才不会被覆盖，并且捕获到可靠的数据。

**Data arrival time - Clock arrival time > Thold**

**Data arrival time > Clock arrival time + Thold**



## 保持时间检查：

$$T_{launch} + T_{ck2q} + T_{dp} > T_{capture} + T_{hold}$$

## 时钟偏斜：

$$\text{Clock\_skew} = T_{capture} - T_{launch}$$

## 建立时间余量：

$$\text{Hold Slack} = \text{Data arrival time} - (\text{Clock arrival time} + T_{hold})$$

$$= T_{launch} + T_{ck2q} + T_{dp} - (T_{capture} + T_{hold})$$

**数据需要保持稳定的最短时间（下一个数据最快到来的时间）：**

$$\text{Data require time} = \text{Clock arrival time} + T_{hold} = T_{cycle} + T_{capture} - T_{su}$$

保持时间检查对捕获触发器的数据路径施加了**最小值 (-min)**约束，需要确定到捕获触发器D引脚的最快路径。这意味着将始终使用**最短时序路径**来进行保持时间检查，同样，通常在**快速工艺角**下进行保持时间检查。

Hold Slack和  
Setup Slack的  
关系？



Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKM)  
Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKM)

Path Group: CLKM  
Path Type: min

根据endpoint划分

元件（路径）延迟

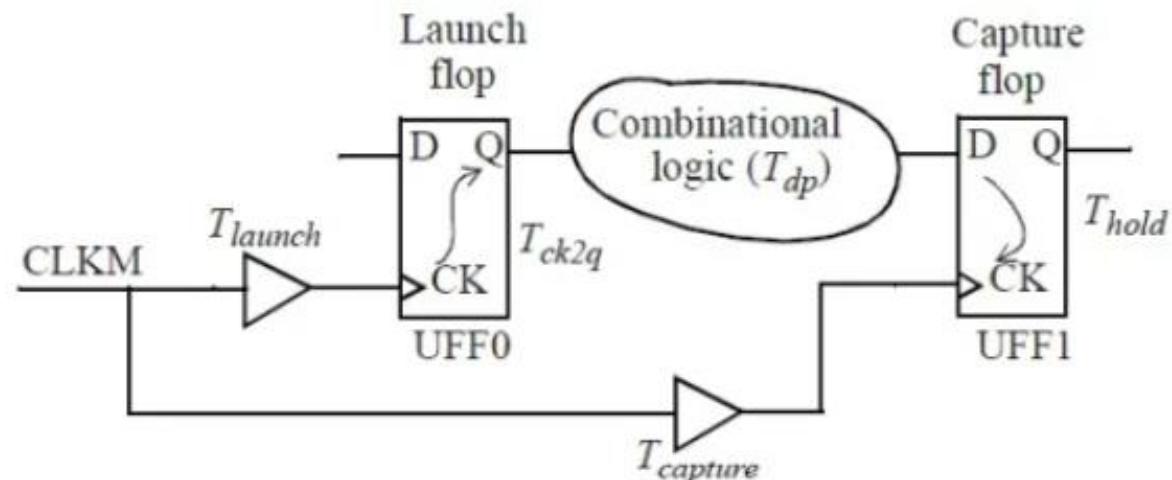
Min代表保持时间分析

Incr

Path

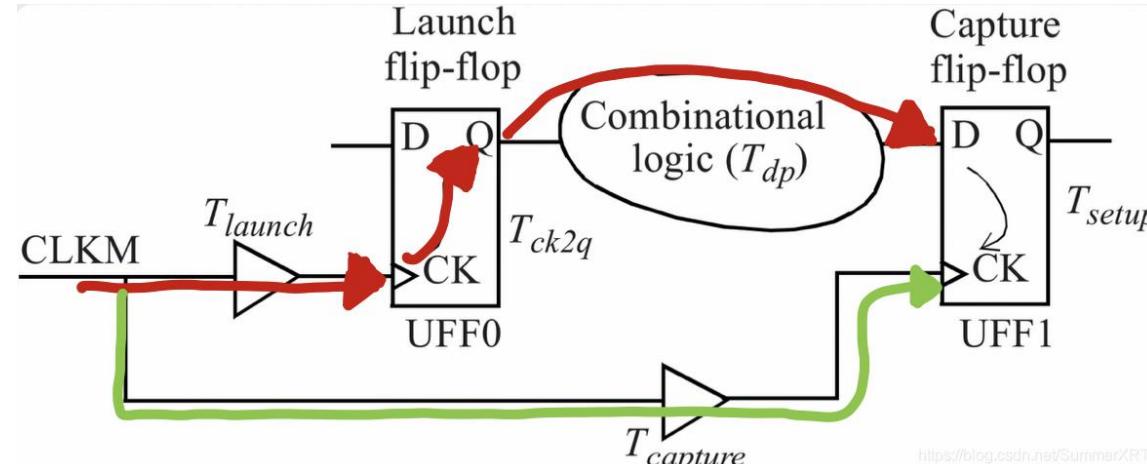
总延迟

Point	Incr	Path
clock CLKM (rise edge)	0.00	0.00
clock source latency	0.00	0.00
CLKM (in)	0.00	0.00 r
UCKBUF0/C (CKB )	0.06	0.06 r
UCKBUF1/C (CKB )	0.06	0.11 r
UFF0/CK (DFF )	0.00	0.11 r
UFF0/Q (DFF ) <-	0.14	0.26 r
UNOR0/ZN (NR2 )	0.02	0.28 f
UBUF4/Z (BUFF )	0.06	0.33 f
UFF1/D (DFF )	0.00	0.33 f
data arrival time		0.33
clock CLKM (rise edge)	0.00	0.00
clock source latency	0.00	0.00
CLKM (in)	0.00	0.00 r
UCKBUF0/C (CKB )	0.06	0.06 r
UCKBUF2/C (CKB )	0.07	0.12 r
UFF1/CK (DFF )	0.00	0.12 r
clock uncertainty	0.05	0.17
library hold time	0.01	0.19
data required time		0.19
data required time		0.19
data arrival time		-0.33
slack (MET)		0.14





## 解决保持时间不满足



<https://blog.csdn.net/GummerXRT>

当保持时间不满足时，也就是 T<sub>capture</sub> + T<sub>hold</sub> 要小于 T<sub>launch</sub> + T<sub>d<sub>p</sub></sub> + T<sub>ck2q</sub>，可以通过：

- 理论上，可以增加T<sub>d<sub>p</sub></sub>和T<sub>ck2q</sub>来解决，也就是增加触发器的D端到Q端的延时，以及两级触发器之间的组合逻辑电路部分。
- 目前大部分芯片的T<sub>hold</sub>时间都可以做到0ns。一般在前端不考虑保持时间违例，因为布局布线之后会自动优化，布线之后当保持时间仍不满足时，通常采用的做法是在传输路径上插入buffer，在不影响逻辑功能前提下，只具有增加组合逻辑延迟的作用，Xilinx 还提到可以插入 LUT1 增加延迟等方式来修复；或者后端布局布线拉长布线以增加延时。
- 减小时钟歪斜T<sub>skew</sub>= T<sub>capture</sub> - T<sub>launch</sub>。如果时钟歪斜 T<sub>skew</sub> 为正，对setup是有利的，对hold是有害。

PS:尤其注意，保持时间T<sub>hold</sub>和时钟频率之间是没有关系，故降低时钟速度不能解决保持时间不满足的问题，很多面试题喜欢在这里挖坑。



Source Clock Path 源端时钟路径					Destination Clock Path 目的端时钟路径				
Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)	Delay Type	Incr (ns)	Path ...	Location	Netlist Resource(s)
(clock sys_clk rise edge)	(r) 0.000	0.000			(clock sys_clk rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	Site: U20	sys_clk		(r) 0.000	0.000	Site: U20	sys_clk
net (fo=0)	0.000	0.000		sys_clk	net (fo=0)	0.000	0.000		sys_clk
			Site: U20	sys_clk_IBUF_inst/l				Site: U20	sys_clk_IBUF_inst/l
IBUF (Prop_ibuf_I_O)	(r) 0.211	0.211	Site: U20	sys_clk_IBUF_inst/O	IBUF (Prop_ibuf_I_O)	(r) 0.400	0.400	Site: U20	sys_clk_IBUF_inst/O
net (fo=1, routed)	0.634	0.845		sys_clk_IBUF	net (fo=1, routed)	0.689	1.089		sys_clk_IBUF
			Site: BUF...TRL_X0Y0	sys_clk_IBUF_BUFG_inst/l				Site: BUF...TRL_X0Y0	sys_clk_IBUF_BUFG_inst/l
BUFG (Prop_bufq_I_O)	(r) 0.026	0.871	Site: BUF...TRL_X0Y0	sys_clk_IBUF_BUFG_inst/O	BUFG (Prop_bufq_I_O)	(r) 0.029	1.118	Site: BUF...TRL_X0Y0	sys_clk_IBUF_BUFG_inst/O
net (fo=8, routed)	0.595	1.466		sys_clk_IBUF_BUFG	net (fo=8, routed)	0.866	1.984		sys_clk_IBUF_BUFG
FDRE			Site: SLICE_X0Y3	cnt_reg[1]/C	FDRE			Site: SLICE_X0Y3	cnt_reg[3]/C
					clock pessimism	-0.518	1.466		
					FDRE (Hold_fdre_C_D)	0.105	1.571	Site: SLICE_X0Y3	cnt_reg[3]
					Required Time		1.571		

## Clock Pessimism Removal 时钟悲观度

时钟悲观度即为**共同路径最大延时 - 最小延时**。综合软件计算建立时间余量的过程中为了模拟最坏情况会将数据到达时间的延时设置为最大，对于共同路径延时应该相同但是也会采用最坏时序，时钟到达时间的延时设置为最短，对于共同路径延因此需要考虑悲观度；

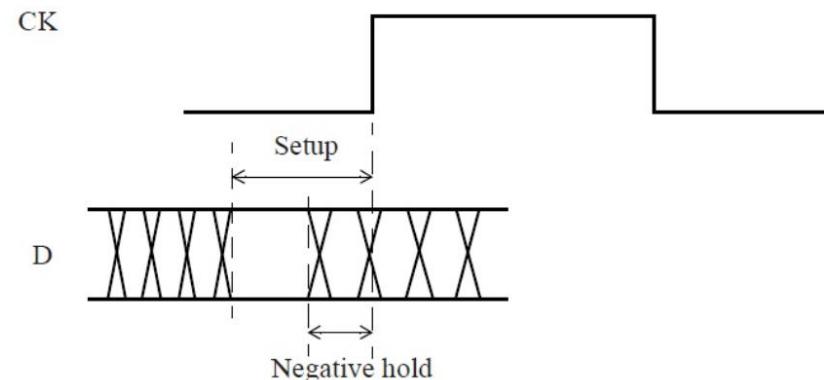
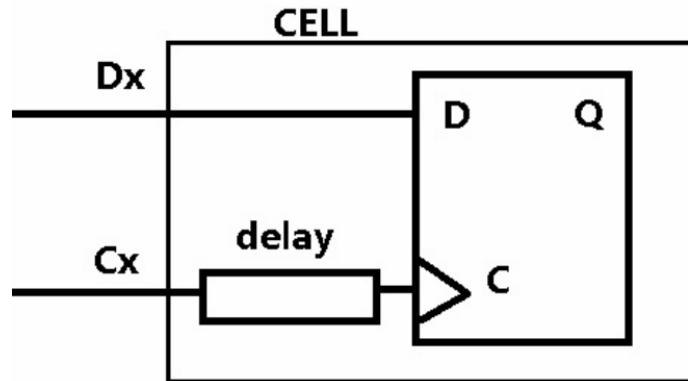
相同的路径结果最后计算出来的路径延迟是不一样的，明显是不合理的，所以xilinx就采用了**时钟悲观度（Clock Pessimism Removal (CPR)）**这个概念用来补偿两条路径之间的相同部分。

仔细观察时序报告便可以发现在报告路径的Slack之前有一行显示clock pessimism已经被考虑在内，在进行Setup Check时会加上一定的clock pessimism，而Hold Check时则会减去一定的clock pessimism。



## 建立/保持时间有可能是负值吗？

建立时间和保持时间是寄存器的固定属性，不同的FPGA器件都应该是一个不同的值，且理论上这两个值都应该是正数。但是有时候在时序分析报告中显示的建立时间/保持时间却是个负数。这是为什么呢？



这种情况在芯片设计中很常见，比如使用standard cell，分析的是CELL外部的时序，不过如果CELL内部clock的延迟大于data的延迟。假设这个延迟是delay， CELL内部寄存器固有的setup时间为tsu，则CELL的 $setup\_time = tsu - delay$ ，当clock延迟足够大，那么从CELL看建立时间就是负值。

负的保持时间检查意味着触发器的数据引脚可以在时钟引脚之前改变，并且仍然满足保持时间的检查要求。

尽管这种情况的建立时间从整体看起来是个负值，但这个建立时间已经不是传统意义上的概念了，而应该是一个对于系统的相对值。还要注意的是，setup和hold不能同时为负值，而且二者之和必须为正。



# 特殊时序检查

- 多周期路径
- 半周期路径
- 虚假路径
- 组合路径延迟
- 撤销时间检查
- 恢复时间检查



FPGA的设计约束分为物理约束和时序约束：

**物理约束**主要包括 I/O 接口约束，布局约束，布线约束以及配置约束。其中 I/O 接口约束主要为引脚分配、电平标准设定等物理属性的约束。

**时序约束**是涉及FPGA内部的各种逻辑或走线的延时，反应系统的频率和速度的约束。

**时序约束：用来描述设计人员对时序的要求，比如时钟频率，输入输出的延时等。**

比如，对时钟频率的约束最简单的理解就是，设计者需要告诉EDA工具设计中所使用的时钟频率是多少；然后工具才能按照所要求的时钟频率去优化布局布线，使设计能够在要求的时钟频率下正常工作。

**在时序分析的过程中，主要有三种特殊时序检查，需要进行特殊约束**

- 虚假路径
- 多周期路径
- 最小和最大延迟



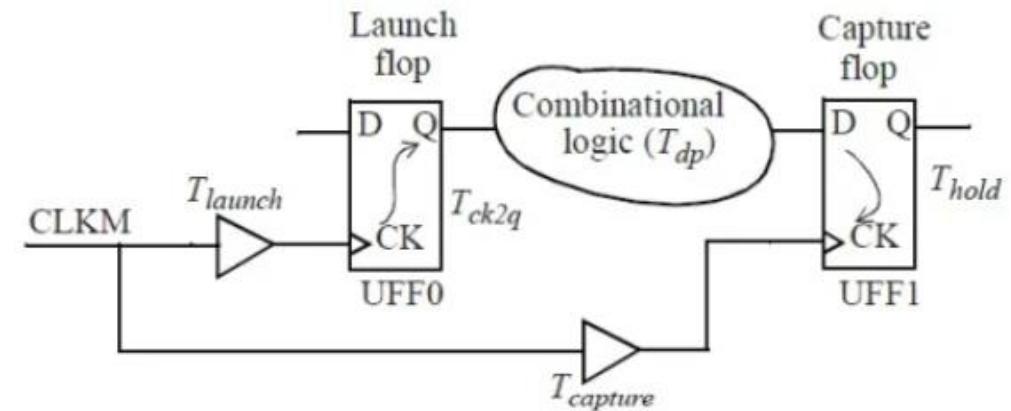
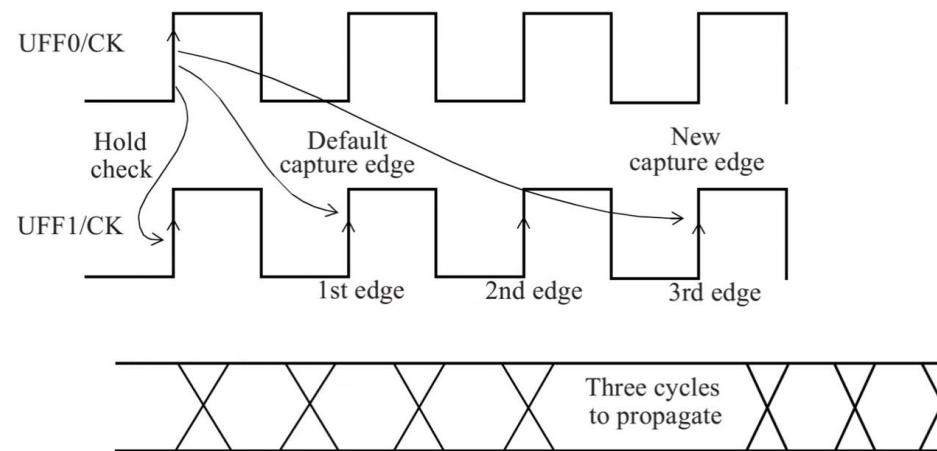
## 多周期路径 (Multicycle Path)

约束: set\_multicycle\_path

### Setup

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

含义: 将对应路径setup时序检查的周期为设置为3, 检查沿延迟到第三个周期处, 初始为第一个周期检查  
对应的波形图为:





## 对应的时序检查报告为：

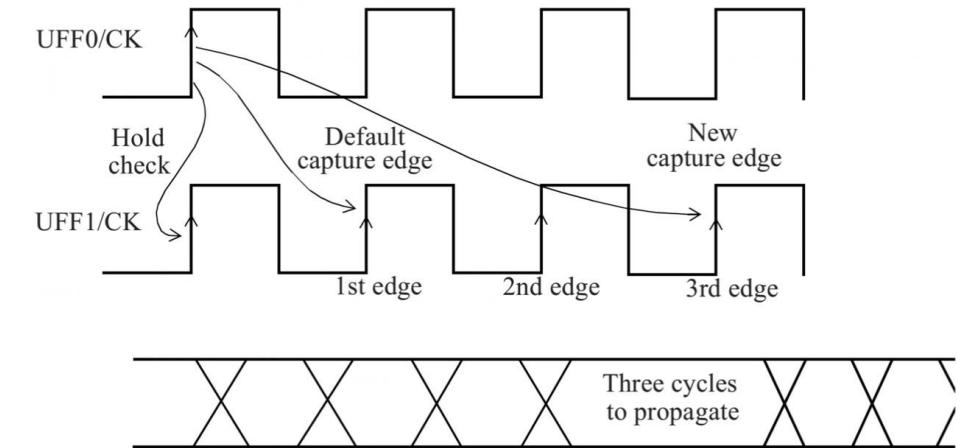
Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKM)

Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKM)

Path Group: CLKM

Path Type: max

Point	Incr	Path
<b>clock CLKM (rise edge)</b>	<b>0.00</b>	0.00
clock network delay (propagated)	0.11	0.11
UFF0/CK (DFF )	0.00	0.11 r
UFF0/Q (DFF ) <-	0.14	0.26 f
UNOR0/ZN (NR2 )	0.04	0.30 r
UBUF4/Z (BUFF )	0.05	0.35 r
UFF1/D (DFF )	0.00	0.35 r
data arrival time		0.35
<b>clock CLKM (rise edge)</b>	<b>30.00</b>	30.00
clock network delay (propagated)	0.12	30.12
clock uncertainty	-0.30	29.82
UFF1/CK (DFF )		29.82 r
library <b>setup</b> time	-0.04	29.78
data required time		29.78
data required time		29.78
data arrival time		-0.35
slack (MET)		29.43



注意约束之后在第三个时钟边沿capture (30ns)



## 多周期路径 (Multicycle Path)

### Hold

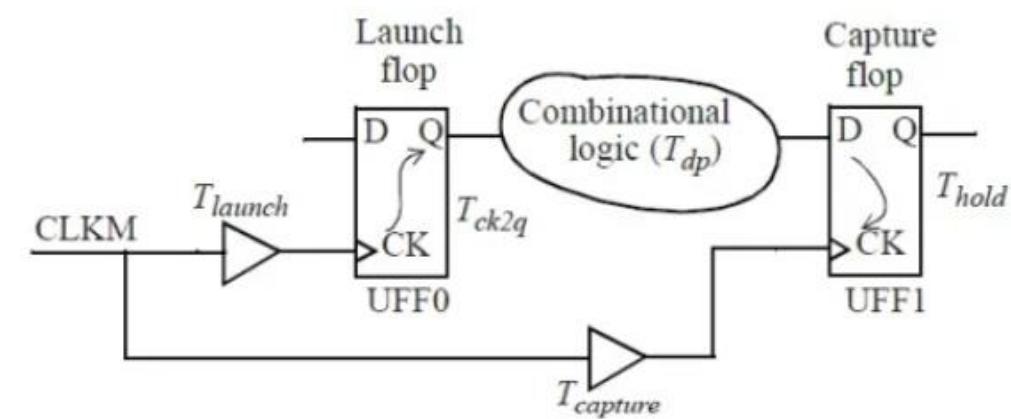
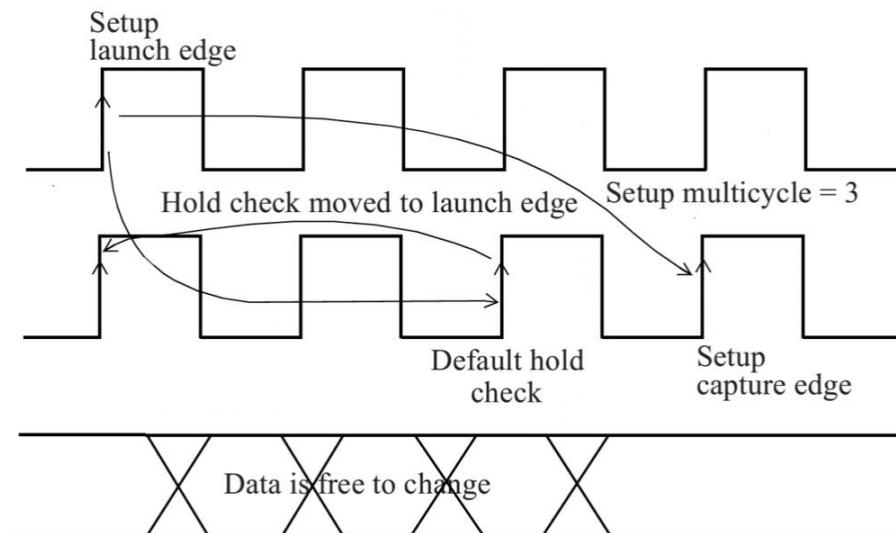
```
set_multicycle_path 2 -hold - from [get_pins UFF0/Q] -to [get_pins UFF1/D]
```

含义：将对应路径 hold 时序检查**提前2个周期**，即检查沿提前两个周期，注意与setup周期数的区别。

setup 设置为第三个触发沿的时候，此时默认检查hold time的沿为setup前一个沿，对于多周期路径来说是不太科学的，因为一个多周期的路径有可能在0-3个周期内变化，而hold却默认为第二个沿，太过于严苛。

在大部分的设计中，**一个N周期的多周期 setup 约束应该伴随着一个N-1周期的多周期hold约束。**

对应的波形图为：





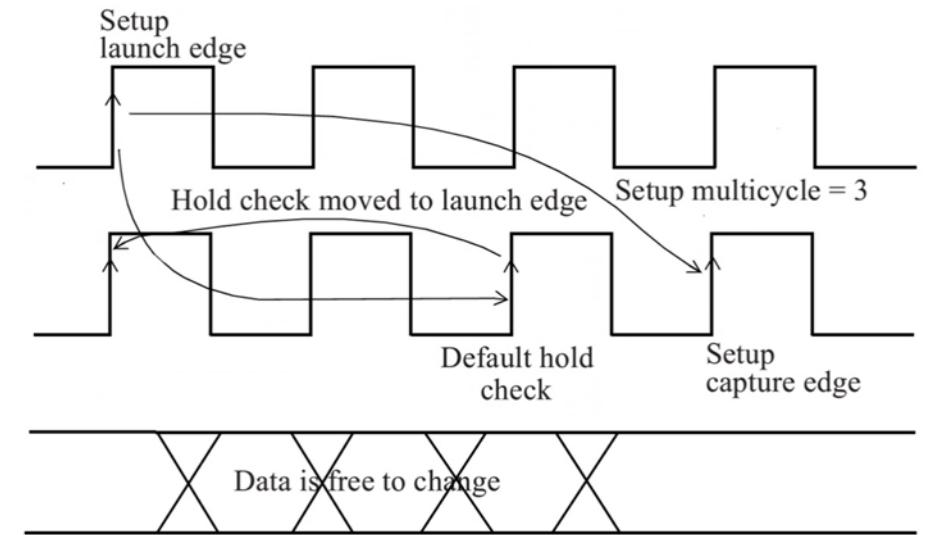
## 对应的时序检查报告为：

Startpoint: UFF0 (rising edge-triggered flip-flop clocked by CLKP)  
Endpoint: UFF1 (rising edge-triggered flip-flop clocked by CLKP)

Path Group: CLKP

Path Type: min

Point	Incr	Path
<b>clock CLKP (rise edge)</b>	<b>0.00</b>	0.00
clock source latency	0.00	0.00
CLKP (in)	0.00	0.00 r
UCKBUF4/C (CKB )	0.07	0.07 r
UFF0/CK (DFF )	0.00	0.07 r
UFF0/Q (DFF ) <-	0.15	0.22 f
UXOR1/Z (XOR2 )	0.07	0.29 f
UFF1/D (DFF )	0.00	0.29 f
data arrival time		0.29
<b>clock CLKP (rise edge)</b>	<b>0.00</b>	0.00
clock source latency	0.00	0.00
CLKP (in)	0.00	0.00 r
UCKBUF4/C (CKB )	0.07	0.07 r
UCKBUF5/C (CKB )	0.06	0.13 r
UFF1/CK (DFF )	0.00	0.13 r
clock uncertainty	0.05	0.18
library <b>hold</b> time	0.01	0.19
data required time		0.19
data required time		0.19
data arrival time		-0.29
slack (MET)		0.11

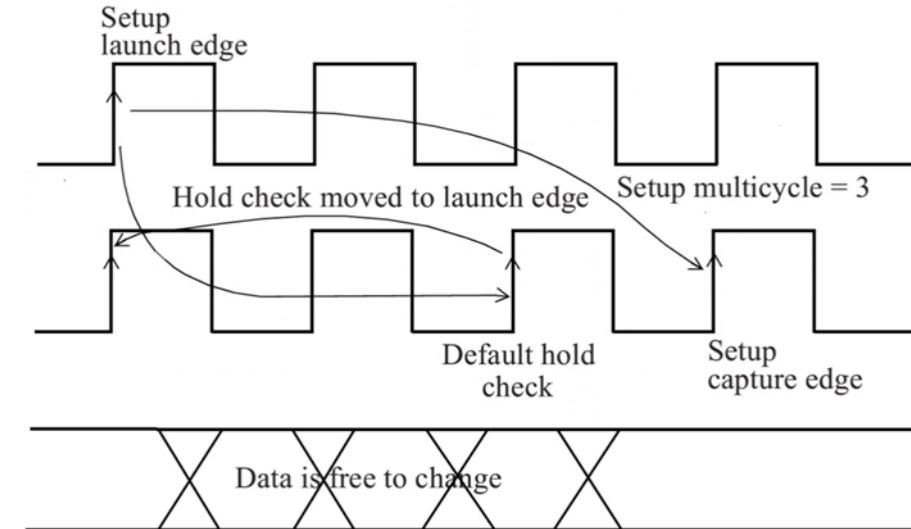


约束之后在原始边沿capture (0ns)



当设置 N周期的setup 而不设置 N-1周期的hold时，起始沿为20ns，发生时序违例：

clock CLKM (rise edge)	20.00	20.00
clock source latency	0.00	20.00
CLKM (in)	0.00	20.00 r
UCKBUFO/C (CKB )	0.06	20.06 r
UCKBUF2/C (CKB )	0.07	20.12 r
UFF1/CK (DFF )	0.00	20.12 r
clock uncertainty	0.05	20.17
library hold time	0.01	20.19
data required time		20.19
<hr/>		
data required time		20.19
data arrival time		-0.33
<hr/>		
slack (VOLATED)		-19.85



时序违例



## 多周期总结

多周期路径的使用场景：

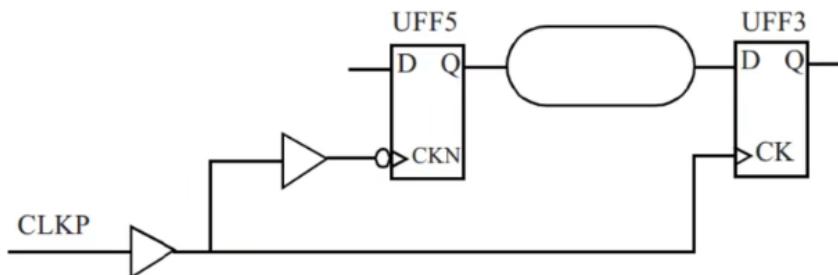
- **数据信号与使能信号路径差异大的时候**，数据与使能信号同时生成，但是使能信号路径可能比较长（可能经过FSM），所以data不需要立即被捕获，此时可以将data的路径约束为数个周期。 (DMUX)
- **源同步接口**
- **复位**：在很多ASIC设计中断言主复位信号保持几个周期，可用多周期路径对断言进行声明。
- **异步时钟**：异步路径虽然被禁止进行时序分析，但是可以对异步路径可能产生的延迟设定一些上限

NOTE: **通过多周期路径移动建立沿的时候，保持沿也会移动，应该坚持保持沿是否需要回复到初始位置（一般需要）**。如果未恢复保持沿，则设计中数据路径可能有额外的缓冲器，以增加延迟来满足保持要求，这一般会导致硅片的面积和功耗的增加。



## 半周期路径 (Half Path)

如果一个设计同时具有下降沿触发的触发器和上升沿触发的触发器，那么很可能在设计中会存在半周期路径。中间的时序路径只剩下半个时钟周期。



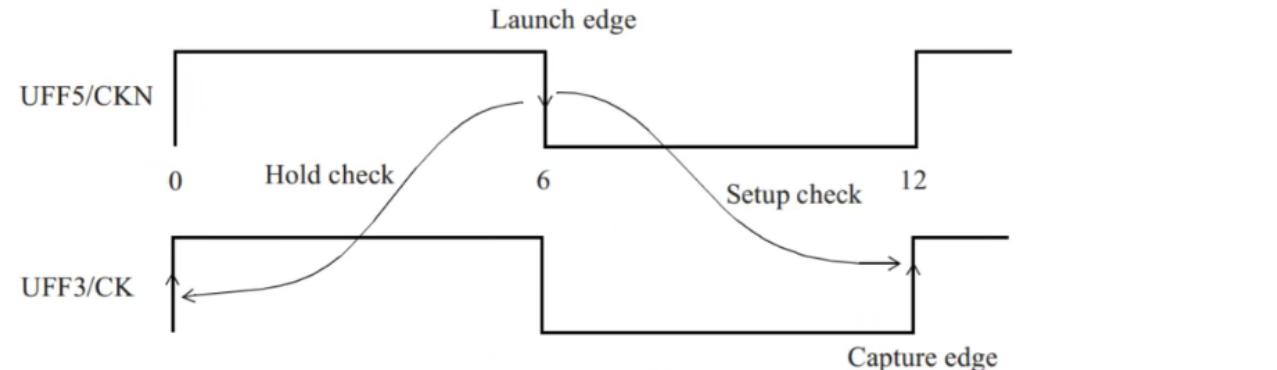
### Setup分析

Startpoint: UFF5 (**falling edge-triggered** flip-flop clocked by CLKP)  
Endpoint: UFF3 (**rising edge-triggered** flip-flop clocked by CLKP)

Path Group: CLKP

Path Type: **max**

Point	Launch 沿从 T/2 开始	Incr	Path
<b>clock CLKP (fall edge)</b>	<b>6.00</b>	6.00	
clock source latency	0.00	6.00	
CLKP (in)	0.00	6.00 f	
UCKBUF4/C (CKB )	0.06	6.06 f	
UCKBUF6/C (CKB )	0.06	6.12 f	
UFF5/CKN (DFN )	0.00	6.12 f	
UFF5/Q (DFN ) <-	0.16	6.28 r	
UNAND0/ZN (ND2 )	0.03	6.31 f	
UFF3/D (DFF )	0.00	6.31 f	
data arrival time		6.31	



**clock CLKP (rise edge)**

clock source latency

CLKP (in)

UCKBUF4/C (CKB )

UFF3/CK (DFF )

clock uncertainty

library **setup** time

data required time

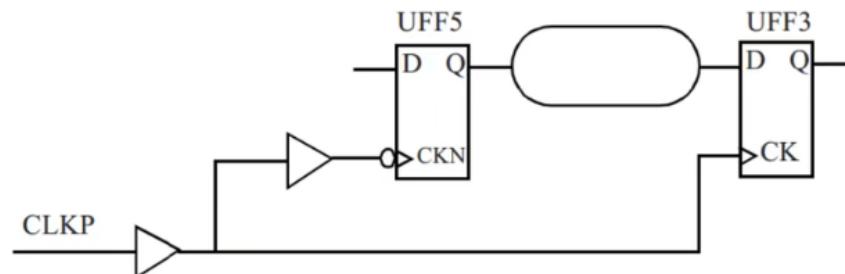
data required time

data arrival time

slack (MET)

12.00	12.00
0.00	12.00
0.00	12.00 r
0.07	12.07 r
0.00	12.07 r
-0.30	11.77
-0.03	11.74
	11.74
	-6.31
	5.43

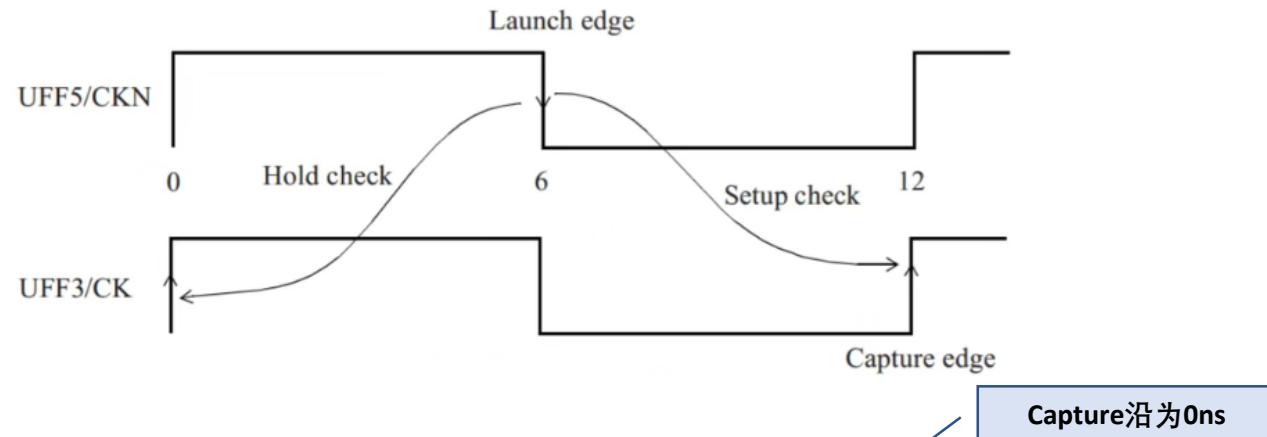
Capture 沿为 T



## Hold分析

Startpoint: UFF5 (**falling edge-triggered** flip-flop clocked by CLKP)  
 Endpoint: UFF3 (**rising edge-triggered** flip-flop clocked by CLKP)  
 Path Group: CLKP  
 Path Type: min

Point	Launch沿从T/2开始	Incr	Path
clock CLKP (fall edge)	6.00	6.00	
clock source latency	0.00	6.00	
CLKP (in)	0.00	6.00 f	
UCKBUF4/C (CKB )	0.06	6.06 f	
UCKBUF6/C (CKB )	0.06	6.12 f	
UFF5/CKN (DFN )	0.00	6.12 f	
UFF5/Q (DFN ) <-	0.16	6.28 r	
UNAND0/ZN (ND2 )	0.03	6.31 f	
UFF3/D (DFF )	0.00	6.31 f	
data arrival time		6.31	



clock CLKP (rise edge)

clock source latency

CLKP (in)

UCKBUF4/C (CKB )

UFF3/CK (DFF )

clock uncertainty

library **hold** time

data required time

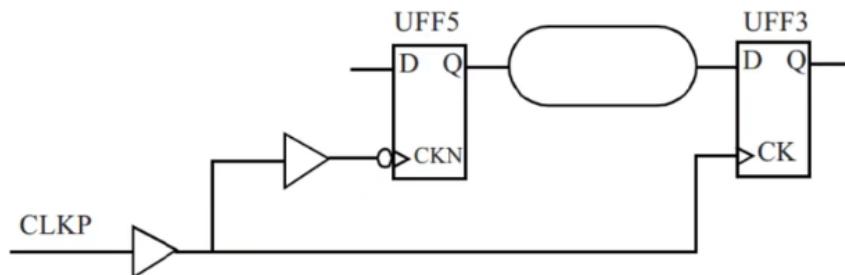
data required time

data arrival time

slack (MET)

0.00	0.00
0.00	0.00
0.00	0.00 r
0.07	0.07 r
0.00	0.07 r
0.05	0.12
0.02	0.13
	0.13
	0.13
	0.13
	0.13
	-6.31
	6.18

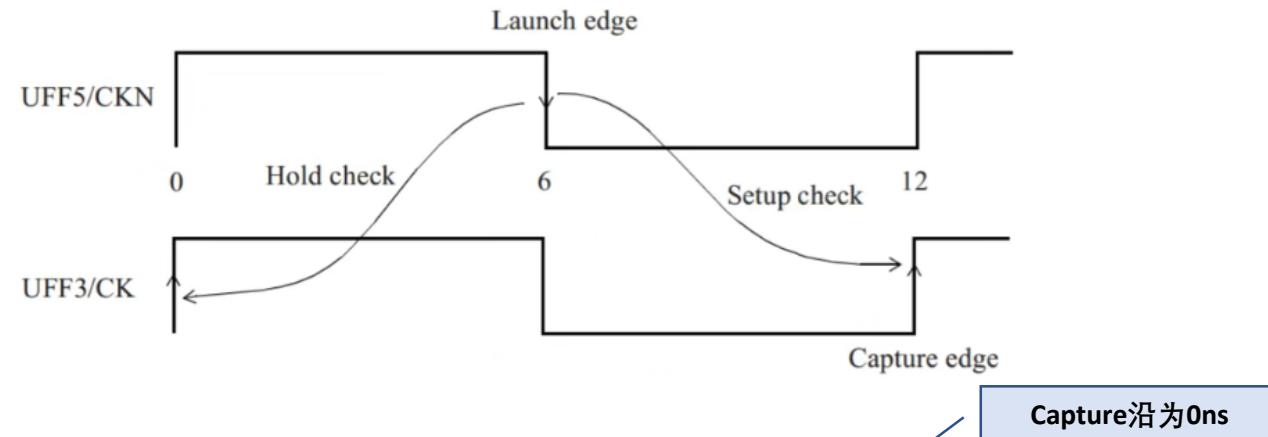
hold检查的capture沿为setup检查的capture沿的上一个沿，即capture沿从0时刻开始，此时**hold自带半个周期的slack**，很容易满足**hold**要求，但是同样的不容易满足**setup**的要求。



## Hold分析

Startpoint: UFF5 (**falling edge-triggered** flip-flop clocked by CLKP)  
 Endpoint: UFF3 (**rising edge-triggered** flip-flop clocked by CLKP)  
 Path Group: CLKP  
 Path Type: min

Point	Launch沿从T/2开始	Incr	Path
clock CLKP (fall edge)	6.00	6.00	
clock source latency	0.00	6.00	
CLKP (in)	0.00	6.00 f	
UCKBUF4/C (CKB )	0.06	6.06 f	
UCKBUF6/C (CKB )	0.06	6.12 f	
UFF5/CKN (DFN )	0.00	6.12 f	
UFF5/Q (DFN ) <-	0.16	6.28 r	
UNAND0/ZN (ND2 )	0.03	6.31 f	
UFF3/D (DFF )	0.00	6.31 f	
data arrival time		6.31	



clock CLKP (rise edge)

clock source latency

CLKP (in)

UCKBUF4/C (CKB )

UFF3/CK (DFF )

clock uncertainty

library **hold** time

data required time

data required time

data arrival time

slack (MET)

0.00	0.00
0.00	0.00
0.00	0.00 r
0.07	0.07 r
0.00	0.07 r
0.05	0.12
0.02	0.13
	0.13
	0.13
	0.13
	0.13
	0.13
	-6.31
	6.18

Hold Slack很大，正常只有0.18

hold检查的capture沿为setup检查的capture沿的上一个沿，即capture沿从0时刻开始，此时**hold自带半个周期的slack**，很容易满足**hold**要求，但是同样的不容易满足**setup**的要求。



## 适用场景：

当两个设计之间进行信号传递的时候可以选择使用半周期路径的设计进行交互，因为这样保证了setup时序比较紧，hold时序比较松，对于修时序而言，**setup的时序违例比较好修，而对于hold 的时序违例没有那么好修**，所以可以尽量保证hold的时序。



## 虚假路径 (False Path)

约束: set\_false\_path

在实际的设计中有些时序路径可能不是真实存在的，可以将这些路径设置为假路径，在静态时序分析器件将这些路径关闭。**虚假路径在STA中会被忽略。**

可能包括

- 异步时钟域之间的路径
- 基于总线协议的虚假路径
- 组合电路的虚假路径：组合逻辑内部保证了该路径永远不会被激活，所以无需分析
- 虚拟时钟和真实时钟之间的虚假路径

注意必须保证虚假路径不能随便设置，必须保证为虚假路径，否则时序分析会出问题。

set\_disable\_timing: 将路径从时序分析中移除



有可能从输入到输出的路径，不会遇到任何的寄存器，这样的路径被称为组合电路路径。

**可以将组合电路路径上的延迟上限限制在一个范围内，使用下面的约束**

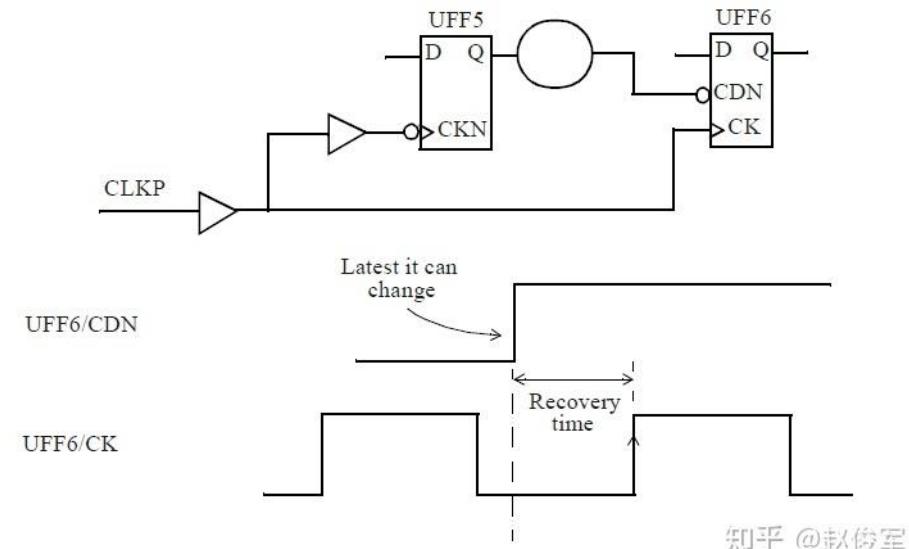
- set\_max\_delay
- set\_min\_delay

也可以使用set\_input\_delay和set\_output\_delay约束得到同样的效果。

如果路径从输入端贯穿到输出端，推荐使用set\_input\_delay和set\_output\_delay，因为可以顺便把寄存器的路径也约束了，后面修改设计也易于维护。



**恢复时间检查 (recovery timing check)** 可确保异步信号变为无效状态的时刻与下一个有效时钟沿之间的时间间隔大于一个最小值。换句话说，此检查可确保在异步信号变为无效状态之后，有足够的时间恢复，以便下一个有效时钟沿可以生效。例如，考虑从**异步复位变为无效的时刻到触发器有效时钟沿之间的时间间隔**。如果该时间间隔太短即有效时钟沿在复位释放后太早出现，则触发器可能进入未知的状态。恢复时间检查如图所示。

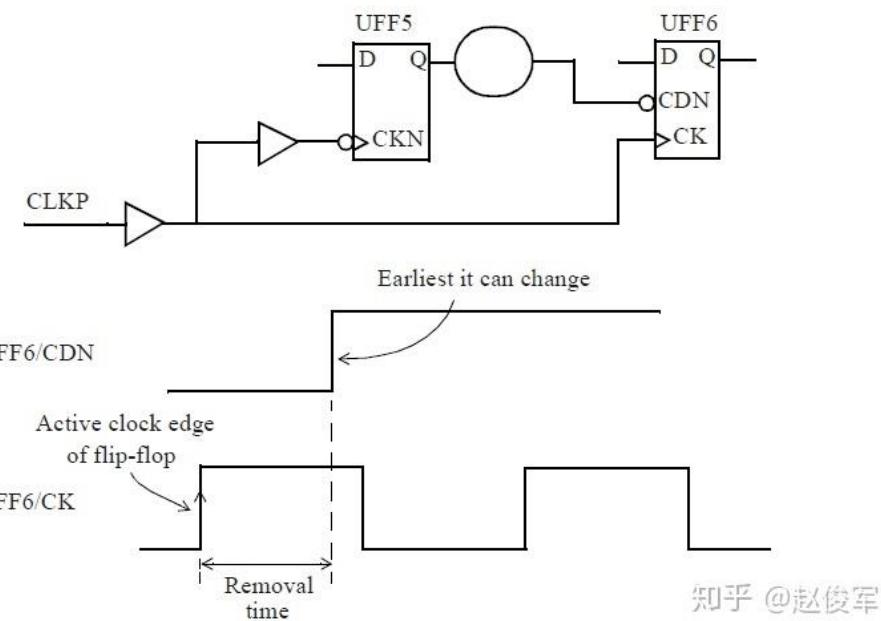


知乎 @赵俊军

与建立时间检查一样，该检查也是针对最大路径的，不过是在触发器的异步引脚上。**可以理解为在异步引脚上的建立时间检查。**



**撤销时间检查** (removal timing check) 可确保在有效时钟沿与释放异步控制信号之间有足够的时  
间。该检查可确保有效时钟沿不带来影响，**因为异步控制信号将保持有效状态，直到有效时钟沿之后一段撤销时  
间为止**。换句话说，异步控制信号会在有效时钟沿之后被释放（变为无效），因此该时钟沿不会产生任何  
影响，如图所示。



知乎 @赵俊军

与保持时间检查一样，该检查也是针对最小路径的，不过是在触发器的异步引脚上。**可以理解为在异步引  
脚上的保持时间检查**。

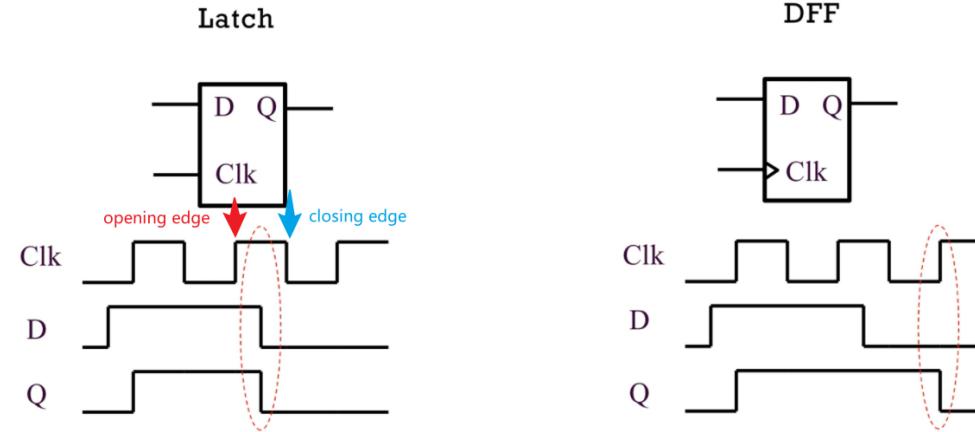


# 其他特殊检查

- **Time Borrowing**
- **Clock\_Gating\_Checks\***
- **Robust Verification**
- **Data to Data check**
- Gray码异步FIFO中的多种约束方式
- 双触发器同步中的约束
- 跨时钟的时序检查\*



## Time Borrowing (Latch)

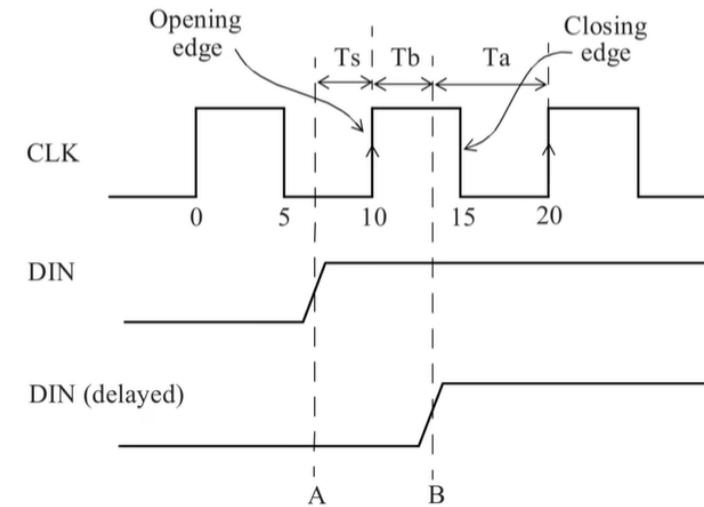
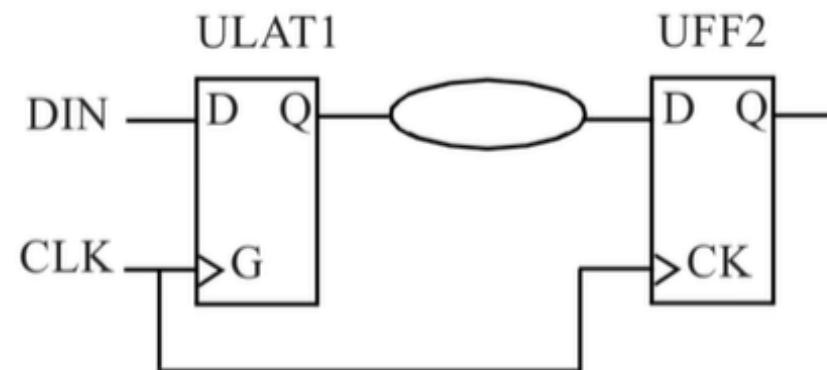


**Latch**在高电平期间对于信号来说是透明的，而DFF只在触发沿处对信号进行采样并维持。

opening edge (latch导通的时候) <==> 触发沿

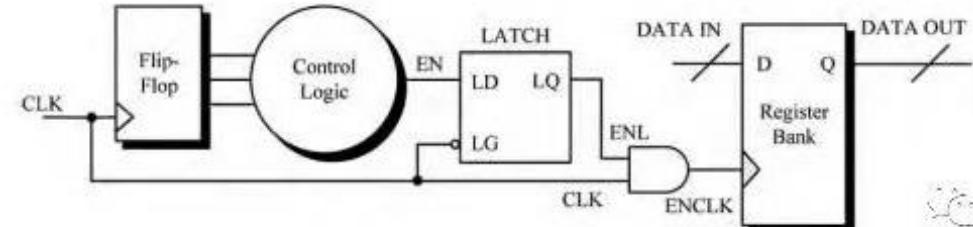
closing edge (latch非导通的时候) <==> 非触发沿

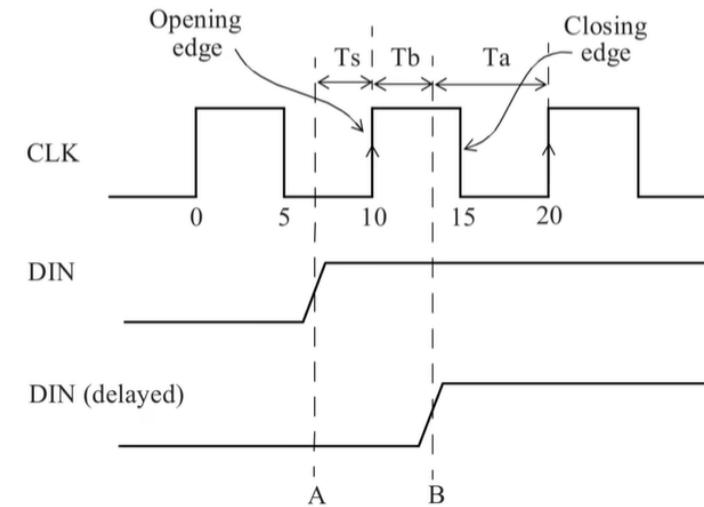
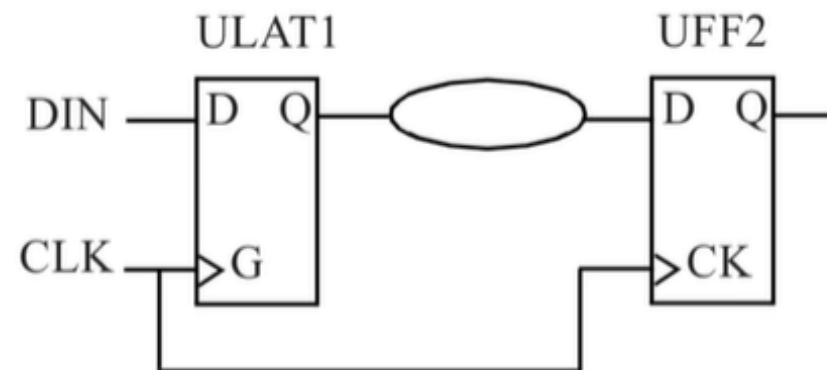
通常来说，在时钟的opening edge之前，数据应该在锁存器的输入端稳定。然而，由于锁存器在有效电平时是透明的，因此**数据可以晚于opening edge到达**，也就是说，它可以从下一个周期借用时间。如果借用了这样的时间，则下一阶段(锁存到另一个连续单元)的可用时间会减少。



如上图所示，若数据在A点到达，即在opening edge之前就保持稳定了，则latch完全可以捕获到数据的变化，若数据在B点到达，则latch在有效电平期间仍然可以捕获到数据的变化，此时**从下一个周期借用了Tb的时间，留给后面组合逻辑计算的时间就相应减少为Ta。**

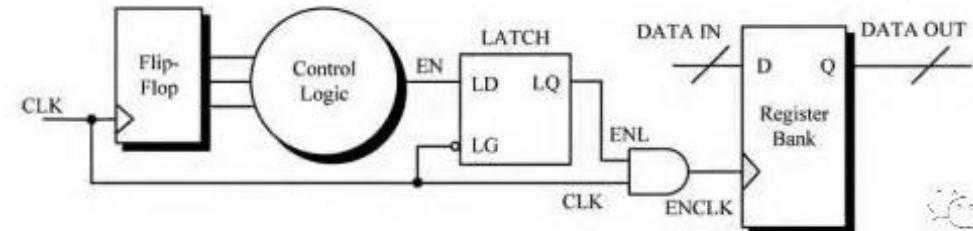
## 经典应用：门控时钟





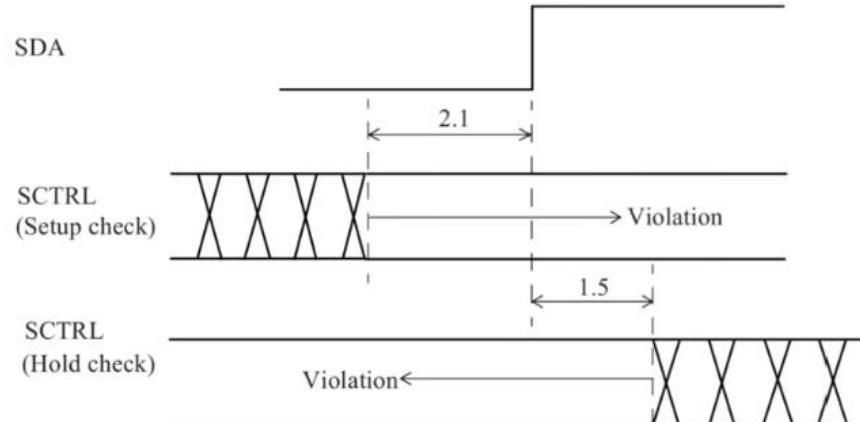
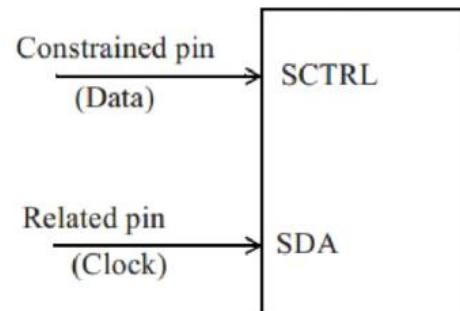
如上图所示，若数据在A点到达，即在opening edge之前就保持稳定了，则latch完全可以捕获到数据的变化，若数据在B点到达，则latch在有效电平期间仍然可以捕获到数据的变化，此时**从下一个周期借用了Tb的时间，留给后面组合逻辑计算的时间就相应减少为Ta。**

## 经典应用：门控时钟





**Data to Data Check:** setup 和 hold 中的检查不只有数据端口和时钟端口之间，也可能发生在任意的两个数据端口之间（如DMUX的两个信号之间）



我们将其中的一个端口 (PIN)设置为约束端口 (constrained pin) , 就像触发器中的数据端口；将另一个端口 (pin)设置为相关端口 (related pin) , 可以类比为触发器中的时钟端口。  
此时的setup check 与 具有时钟端口的setup check之间的区别

- **data to data 的 setup check 会是同一个时钟沿, 即其 launch 和 capture 会在同一个时钟沿。**
- 因此, data to data 的 setup check 也叫作 zero-cycle 或者same-cycle check

data to data check 使用 set\_data\_check 命令进行约束

```
set_data_check -from SDA -to SCTRL -setup 2.1  
set_data_check -from SDA -to SCTRL -hold 1.5
```



## setup check, 对同一边沿进行检查

Startpoint: UDFF1 (rising edge-triggered flip-flop clocked by CLKPLL)			
Endpoint: UAND0 (rising edge-triggered data to data check clocked by CLKPLL)			
Path Group: CLKPLL			
Path Type: max			
Point	Incr	Path	
<b>clock CLKPLL (rise edge)</b>	<b>0.00</b>	0.00	
clock source latency	0.00	0.00	
CLKPLL (in)	0.00	0.00 r	
UDFF1/CK (DF )	0.00	0.00 r	
UDFF1/Q (DF )	0.12	0.12 f	
UBUF0/Z (BUFF )	0.06	0.18 f	
<b>UAND0/A2 (AN2 )</b>	<b>0.00</b>	0.18 f	
data arrival time		0.18	
<b>clock CLKPLL (rise edge)</b>	<b>0.00</b>	0.00	
clock source latency	0.00	0.00	
CLKPLL (in)	0.00	0.00 r	
UDFF0/CK (DF )	0.00	0.00 r	
UDFF0/Q (DF )	0.12	0.12 r	
UBUF1/Z (BUFF )	0.05	0.17 r	
UBUF2/Z (BUFF )	0.05	0.21 r	
UBUF3/Z (BUFF )	0.05	0.26 r	
<b>UAND0/A1 (AN2 )</b>	<b>0.00</b>	0.26 r	
<b>data check setup time</b>	<b>-1.80</b>	-1.54	
data required time		-1.54	
-----			
data required time		-1.54	
data arrival time		-0.18	
-----			
slack (VIOLATED)		-1.72	

## hold check, capture 相对于 setup 的 capture 提前一个周期

Startpoint: UDFF1 (rising edge-triggered flip-flop clocked by CLKPLL)			
Endpoint: UAND0 (falling edge-triggered data to data check clocked by CLKPLL)			
Path Group: CLKPLL			
Path Type: min			
Point	Incr	Path	
<b>clock CLKPLL (rise edge)</b>	<b>10.00</b>	10.00	
clock source latency	0.00	10.00	
CLKPLL (in)	0.00	10.00 r	
UDFF1/CK (DF )	0.00	10.00 r	
UDFF1/Q (DF ) <-	0.12	10.12 r	
UBUF0/Z (BUFF )	0.05	10.17 r	
<b>UAND0/A2 (AN2 )</b>	<b>0.00</b>	10.17 r	
data arrival time		10.17	
<b>clock CLKPLL (rise edge)</b>	<b>0.00</b>	0.00	
clock source latency	0.00	0.00	
CLKPLL (in)	0.00	0.00 r	
UDFF0/CK (DF )	0.00	0.00 r	
UDFF0/Q (DF )	0.12	0.12 f	
UBUF1/Z (BUFF )	0.06	0.18 f	
UBUF2/Z (BUFF )	0.05	0.23 f	
UBUF3/Z (BUFF )	0.06	0.29 f	
<b>UAND0/A1 (AN2 )</b>	<b>0.00</b>	0.29 f	
<b>data check hold time</b>	<b>1.00</b>	1.29	
data required time		1.29	
-----			
data required time		1.29	
data arrival time		-10.17	
-----			
slack (MET)		8.88	



data to data check 可以很方便的用于需要明确两对应信号之间到达时间的自定义模块。一个常见的情况就是，数据信号被使能信号控制，那么我们需要确保数据信号到达前使能信号是稳定的。

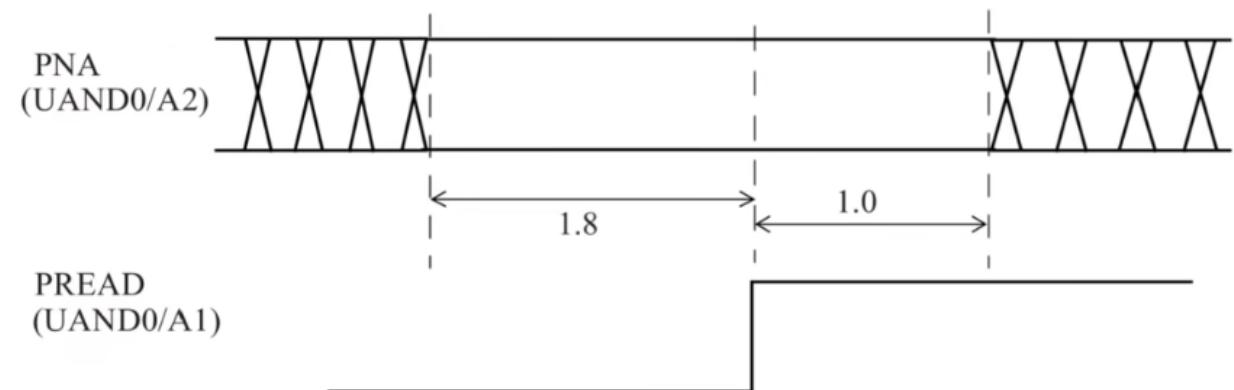
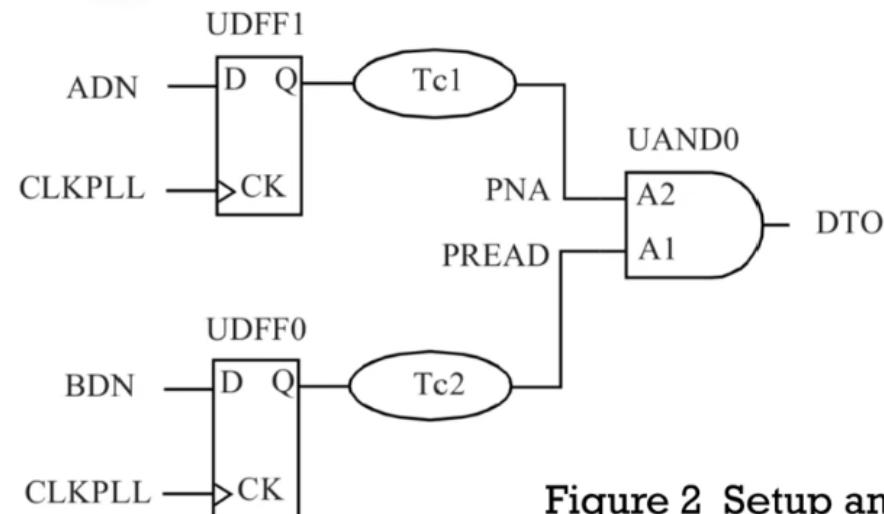


Figure 2 Setup and hold timing checks between PNA and PREAD

以上面的电路为例，DTO容易出现毛刺，此时可以对其进行约束，保证PREAD 信号到达的时候，PNA稳定



data to data check也可以用来约束数据不变化的时间：我们通过在上升沿进行setup check和在下降沿进行hold check，可以有效定义一个数据保持（no-change）窗口。

```
set_data_check -rise_from D2 -to D1 -setup 1.2
```

```
set_data_check -fall_from D2 -to D1 -hold 0.8
```

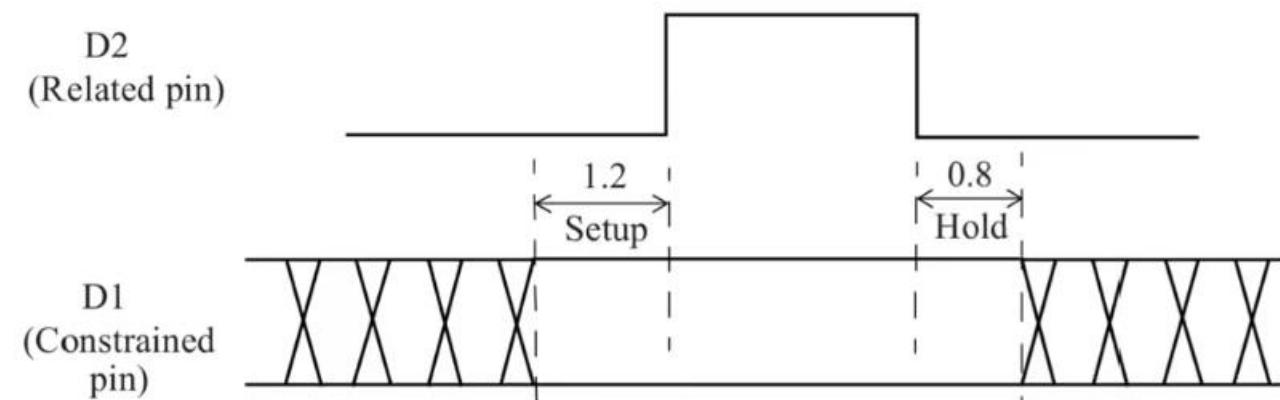


Figure 3 A no-change data check achieved using setup and hold data checks



整个芯片上的延时分布并不均匀，可能由于不同的工艺和环境导致的。

为了整个芯片的鲁棒性，可以考虑 on-chip Variation 将快的路径和慢的路径进行等比例的放大和缩小，让时序检查更为严苛，更符合实际情况，满足鲁棒性的要求

```
set_timing_derate -early 0.8 #将快的路径缩小20%
set_timing_derate -late 1.1 #将慢的路径延长10%
```

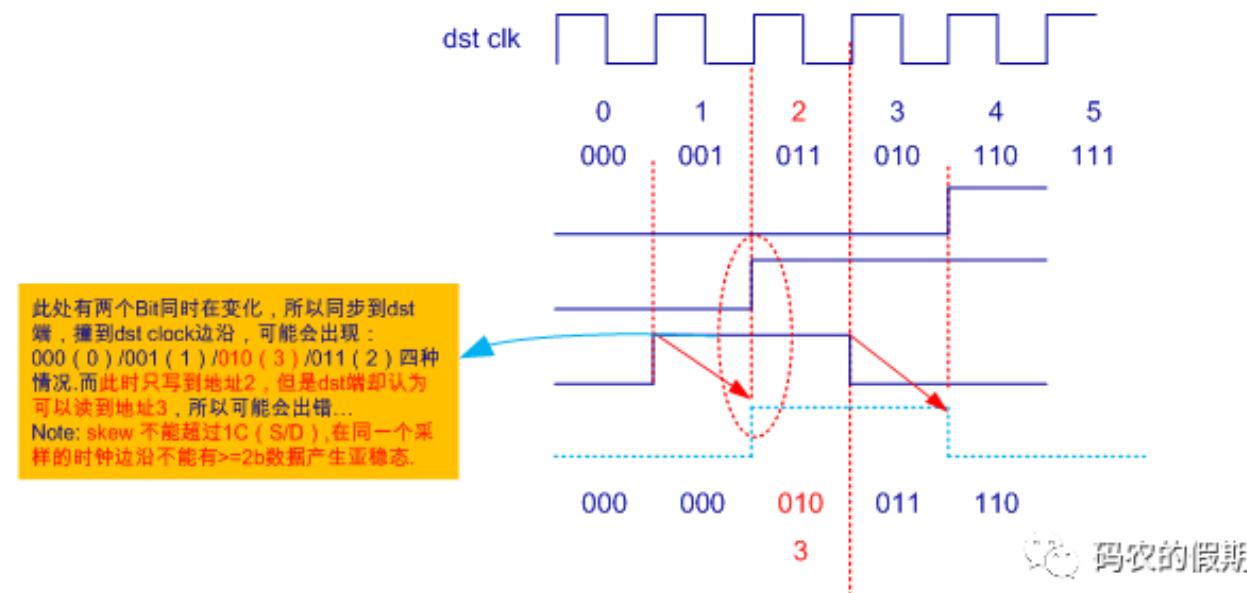
注意考虑OCV的时候要移除公共路径上的悲观度（CPP），因为公共路径上同时乘上了一个比例，但是实际上经过公共路径的延迟应该是一致的。

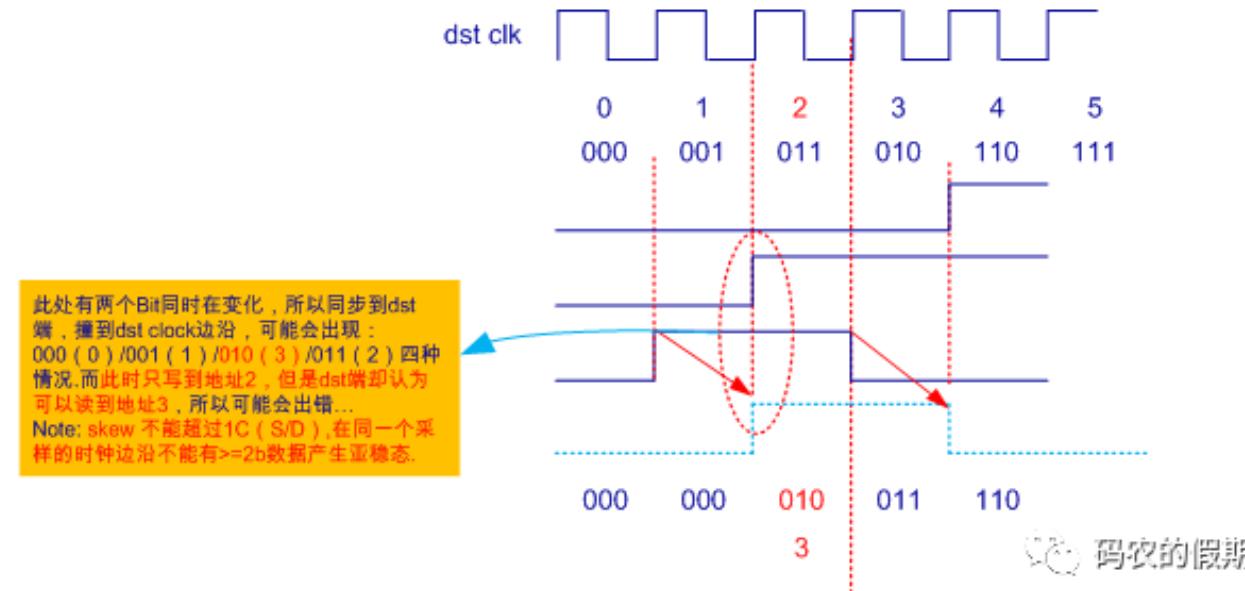


异步FIFO中存在异步路径但是不能完全按照异步路径进行处理 (set\_false\_path),单纯按照异步路径进行处理还会存在一些问题。

## 异步fifo中的多比特跨时钟域问题

异步fifo中采用格雷码进行多比特跨时钟的操作，因为格雷码的编码方式保证了每次只有一个bit会进行翻转。但是通过格雷码编码后的多bit地址信号，由于从S-Clk到D-Clk传递时，是**多条纯异步的path**（芯片中不同时钟域的逻辑电路一般不会布局到一起），那么由于**多条路径的延时不一样**，多bit信号经过这些path到达目的时钟端被采样时，就可能出现**多bit信号同时变化并撞在目的时钟边沿**的情况出现。如下图。





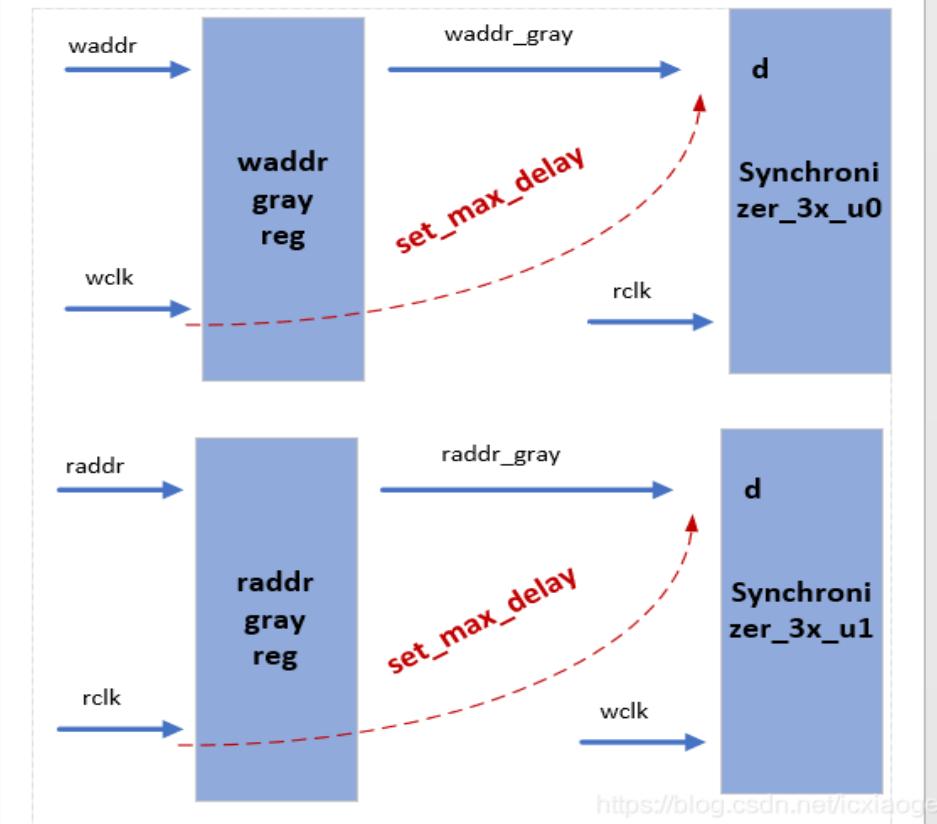
蓝色虚线波形为bit[2]延迟一个周期到达的结果，可以理解为bit[2]这根地址线的线延迟比其他的多一个cycle。而多bit信号同时撞在clock边沿，就会出现都产生亚稳态的问题。如果多个信号出现亚稳态，那么，目的时钟采到的Gray Code地址就没法预计是什么值，因此也就会出现可能源端只写了3个数据，而目的端认为写了5个数据类似的情况，那么必然会导致空满信号出错，从而也就会导致读取的数据出错的情况。

解决问题的方法其实很简单，就是对布局布线的路径进行约束，保证多bit之间的skew不会太大，从而保证多bit信号在目的时钟进行采样不会有两位以上的bit同时进行变化



## 通过 skew 进行约束

约束如图所示：约束从格雷码寄存器的时钟端口---->到3级同步器的输入端口的最大延时。写地址waddr和读地址raddr格雷码同步都需要设置set\_max\_delay。此处set\_max\_delay是为了保证源端信号到达目的端被采样时的格雷码唯一bit跳变特性。延时可设置为读写时钟中最快时钟周期的一半（最大可以为一个cycle），也可以设置成源端时钟的一半，或者设置成源端时钟的倍数且bit间的skew明显小于一个源端时钟周期。保证在采样沿的时候格雷码的多bit信号已经稳定。



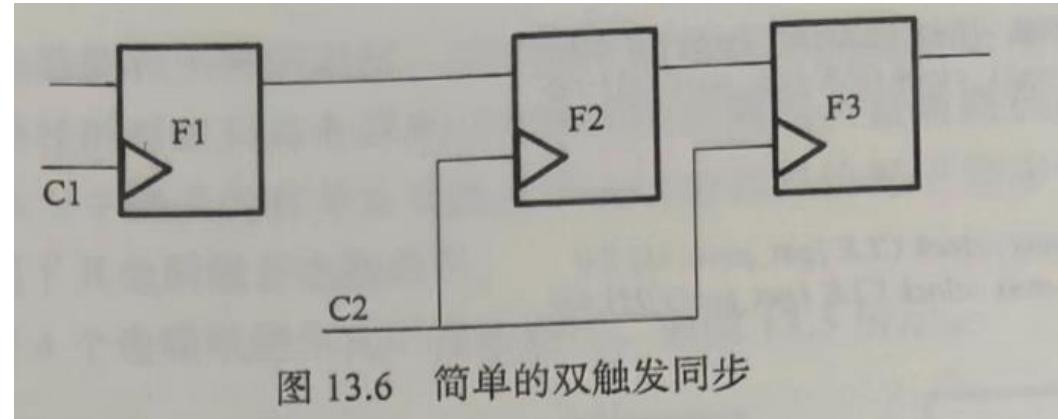
max delay的约束方式，同样也存在SDC优先级的问题，默认情况下是不起作用的（默认是false\_path），如果要想让max delay起作用，我们需要**在约束asynchronous group时加上"-allow\_paths"**的option。这样就会让timing分析工具，在异步clock之间还会分析这些max delay的timing约束。但是同时也会引入了一些本来不需要做timing check的path，需要designer在去人工排除。



## 通过 `max_delay` 进行约束

上面的约束方式，都会由于SDC优先级的限制，需要或多或少的额外修改原本的约束或者设计，并且也会引入一些不需要的问题。另外，上面的两种方式，其实都是一种过约的方式。从我们最开始的分析可以知道，此处多bit Gray Code，只要在目的时钟采样边沿不会有超过两bit同时变化即可，换句话说，这其实是我们**只要保证这多bit的格雷码地址信号的之间的skew在一定范围内**（最大一个周期之内，一般会比较小），就可以避免这个问题发生。而针对这些path，**每bit信号的传输延时其实是没有需求的。**

以3bit gray code信号为例，要想保证不超过2bit信号同时在目的时钟采样边沿变化，必须要保证他们之间的skew不能超过1个cycle的源时钟。但是在真正约束的时候，由于还要考虑library cell的setup时间以及多留一些margin等因素，可以考虑约束这些path的skew不能超过0.8个源时钟cycle或者更紧一些，这些就要by design具体分析下再决定了。



- 不应该定时从F1到F2的路径，这可以通过 set\_clock\_groups 或 set\_false\_path 来实现。或者，用户可能会放置一个set\_multicycle\_path。
- 由于C2定义的时钟周期原因，从F2到F3的路径受到时序检查。通常，没有逻辑摆放在 F2和 F3之间。布局和布线工具可能会将这些触发器放置得很远或者布一根很长的连线，因为他们可以看见对此路径有效的完整时钟周期。如果从F2到F3的路径很长，这样减少了同步的实效性[意味着 MTBF(平均故障间隔时间) 不会增加]。设计人员通常希望从F2到F3的延迟非常小，即远小于允许的时钟周期。所以，他们将使用set\_max\_delay 来约束这个路径，例如：

```
set_max_delay-from F2-to F3 <value>
```



## 平均无故障时间(MTBF)的公式

$$MTBF = \frac{\exp(t_r / \tau)}{T_0 \cdot f_{in} \cdot f_{clock}}$$

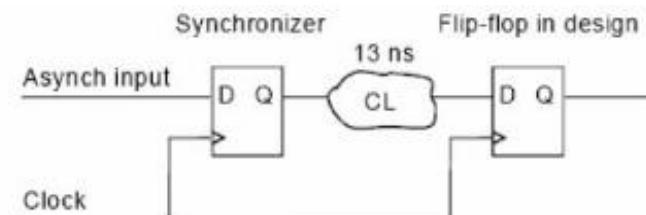
- $t_r$ : 不引起synchronizer failure的前提下，亚稳态可持续的最长时间 (MetaStability Resolution Time)；
- $\tau$  和  $T_0$ : 与触发器电气特性有关的常数；
- $f_{in}$ : 异步输入信号的频率；
- $f_{clock}$ : 起同步作用的触发器时钟频率。[blog.csdn.net/weixin\\_39015789](https://blog.csdn.net/weixin_39015789)

$\tau$	1 ns
$T_0$	$5 \cdot 10^5$ s
$t_{su}$	2 ns

Resolution time:  $t_r = \frac{1}{f_{clock}} - t_{CL} - t_{su} = (100 - 13 - 2) \text{ ns} = 85 \text{ ns}$

$$MTBF = \frac{\exp(85)}{5 \cdot 10^5 \cdot 10 \cdot 10^6 \cdot 3 \cdot 10^3} = 5.5 \cdot 10^{20} \text{ s} = 1.74^{13} \text{ years}$$

[https://blog.csdn.net/weixin\\_39015789](https://blog.csdn.net/weixin_39015789)



当  $T_{CL}$  减少的时候留给MTBF稳定的时间就会增多，有利于减少亚稳态。如果不进行约束的话布局布线工具的范围可能达到一个周期，MTBF反而会减少。



谢谢！