特殊时序检查

多周期时序路径（Multi-Cycle Paths）

> 数据路径中组合逻辑造成的延时大于一个周期

对于多周期时序路径，需要进行多周期时序约束，以组合逻辑需要占用三个周期为例：

```
# 创建时钟
create_clock -name CLKM -period 10 [get_ports CLKM]

# 对3周期路径进行建立时间约束
set_multicycle_path 3 -setup -from [get_pins UFF0/Q] -to [get_pins UFF1/D]
# 对3周期路径进行保持时间约束
set_multicycle_path 2 -hold -from [get_pins UFF0/Q] -to [get_pins UFF1/D]
# 2 -hold 表示的不是两个时钟周期后检查hold，而是在默认hold检查的前2个周期进行hold检查，即在当前边沿检查hold
```

在使用PT工具进行建立时间检查时，3个时钟周期的时间会体现在Capture路径上

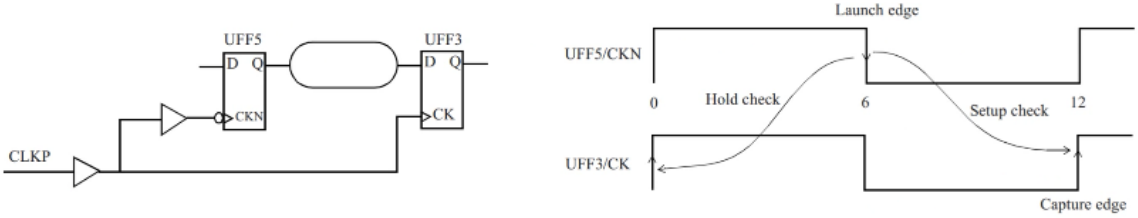
默认情况下，时序分析工具会在检查setup的前一个周期进行hold检查，但对于multicycle路径过于严苛，所以使用2 -hold选项指明，在默认检查hold的前2个周期进行hold检查

- ***三个选项***

```
-path_multiplier     # 默认值 setup 时为1，hold 时为0
-setup | -hold       # 表明多周期路径设置是对于setup（max_delay）还是hold(min_delay)进行，setup时默认移动 Capture_clk，hold时默认移动 Launch_clk
-start | -end        # 手动指定移动的边沿，-start 表示强制移动 Launch Clock，-end 表示强制移动的是 Capture Clock
```

半周期时序路径（Half-Cycle Paths）

> 若设计中的DFF既有使用下降沿采样（active clock edge is falling edge），也有使用上升沿采样（active clock edge is rising edge）（*同时存在*），则设计中存在半周期路径（Half-Cycle Paths）



上图中，进行建立时间检查时，Launch路径时间点为6，Capture时间点为12——Setup紧（T/2）

进行保持时间检查是，Launch路径时间点为6，Capture时间点为0——Hold松

好像没得约束，直接做时序检查

伪路径（False Paths）

> 设计中会存在不可能发生的时序路径——伪路径
>
> 降低静态时序分析的计算空间
>
> ***通常情况下，伪路径存在于跨时钟域的路径中***
>
> ***另外，一些特殊的路径也可被看成伪路径***

- 当伪路径被定义为穿过一个cell中的pin，则这个pin上穿过的所有路径都会被时序分析忽略
- 定义伪路径的目的在于减少时序分析的计算空间
- ***过多的伪路径定义会降低静态时序分析的效率***

> 约束方式

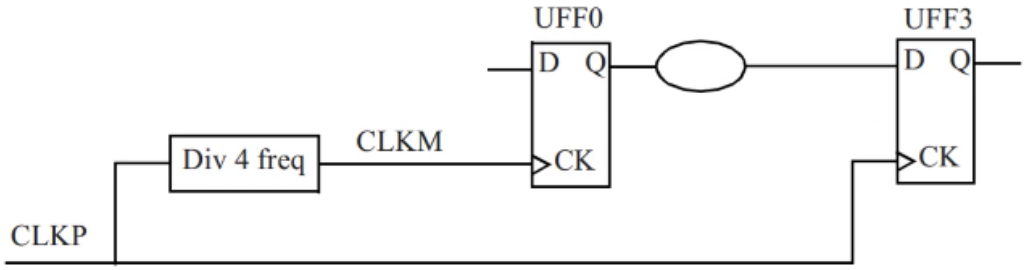
```
# 从时钟域 SCAN_CLK 到 时钟域 CORE_CLK 的所有路径都作为伪路径处理
set_false_path -from [get_clocks SCAN_CLK] -to [get_clocks CORE_CLK]
# 从 Pin(UMUX0/S) 穿过的所有路径都作为伪路径处理
set_false_path -through [get_pins UMUX0/S]
# 到达 Port(TEST_REG*)的所有路径都作为伪路径处理
set_false_path -to [get_ports TEST_REG*]
# 穿过 UINV/Z 和 UAND0/Z 的所有路径都作为伪路径处理
set_false_path -through UINV/Z -through UAND0/Z

#####################################################################
set_false_path -from [get_pins {regA_*}/CP] -to [get_pins {regB_*}/D]
# 这种方式定义的伪路径在静态时序分析时计算的数据量很大，运行缓慢，在定义时直接指定时钟
```

> ***应该尽量少的使用 -through 这个 option ， 其会使时序分析变的复杂***
>
> ***应该着重关注一些特殊路径到底是不是伪路径，应该确保每一条真实路径（Real Paths）都被分析到***

跨时钟域传输

> 设计中存在多时钟域，但是由于时钟之间是分频关系，所以***其实属于同步时钟域，这种情况可以进行分析，不属于异步时钟域，如果是真正的异步时钟域不需要进行分析***

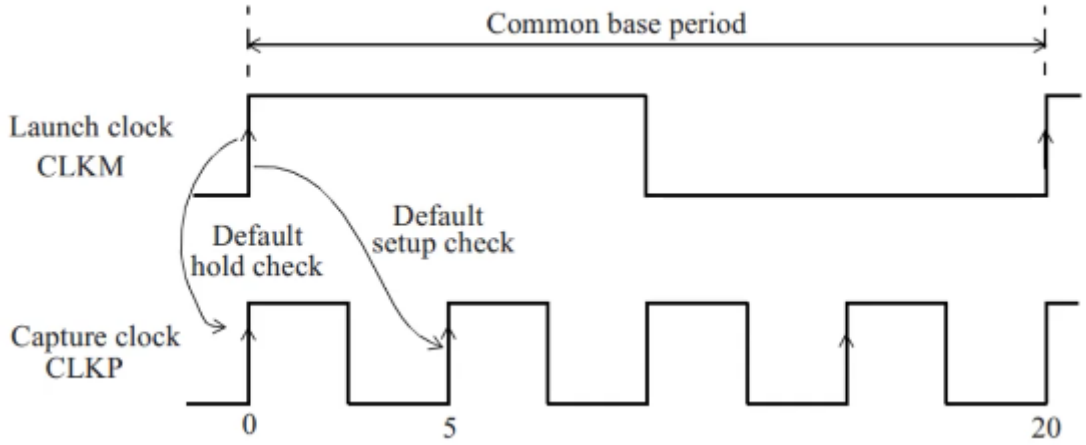
1. **慢时钟域到快时钟域（Slow to Fast Clock Domains）**



> 源时钟为CLKP，数据从其4分频时钟域传输至原始时钟域

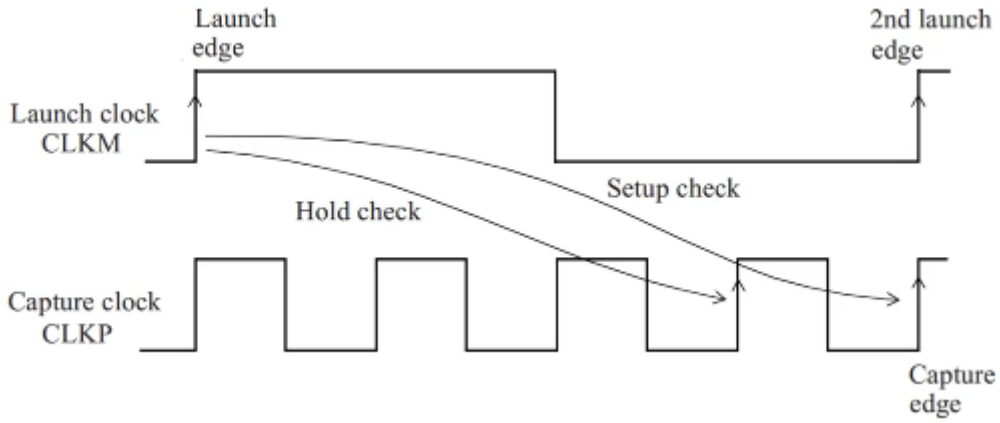
默认情况下，进行建立时间检查时使用最受约束的设置边缘关系，即快时钟域的下一个Capture边沿。

```
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]
```



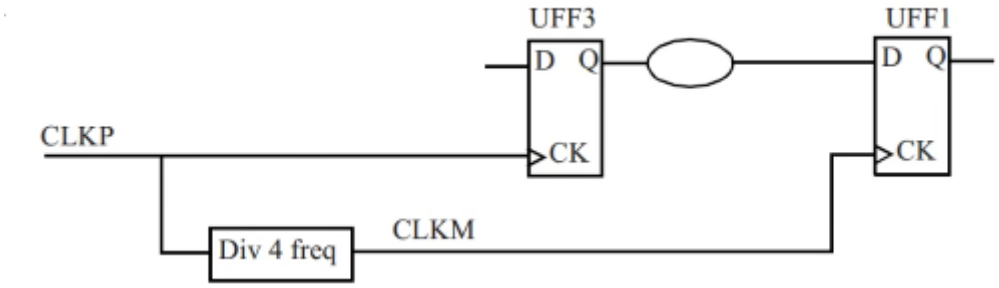
其实不用这么严苛，由于Launch网络的时钟慢，数据更新更缓慢，可以使用多周期路径进行约束

```
set_multicycle_path 4 -setup -from [get_clocks CLKM] -to [get_clocks CLKP] -
end
set_multicycle_path 3 -hold -from [get_clocks CLKM] -to [get_clock CLKP] -
end
# -end 选项 表示强制移动的为 end clock 即 capture clock
```



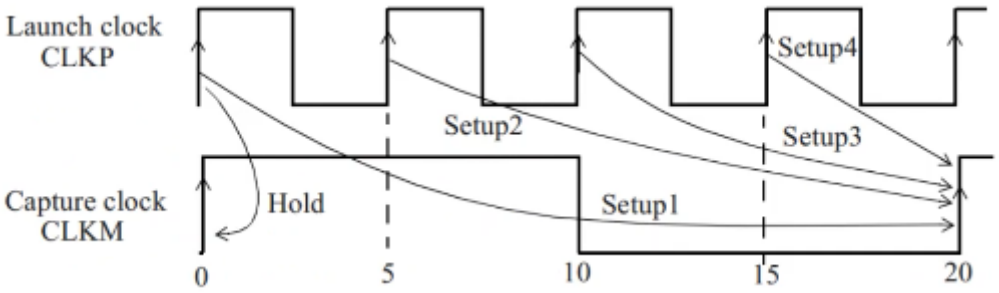
> ***对于slow -to-fast 跨时钟域传输的时序检查，使用多周期路径进行约束，建立时间检查使用N Cycle，保持时间检查使用N-1 Cycle***

2. **快时钟域到慢时钟域（Fast to Slow Clock Domains）**



默认情况下建立时间和保持时间检查的约束

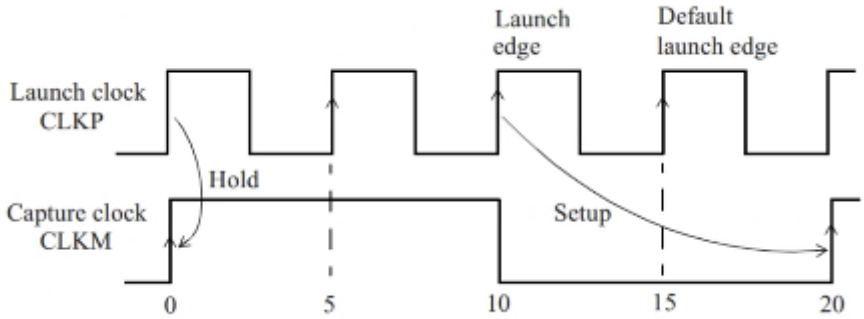
```
create_clock -name CLKM -period 20 -waveform {0 10} [get_ports CLKM]
create_clock -name CLKP -period 5 -waveform {0 2.5} [get_ports CLKP]
```



此时 时序检查不合理，可以考虑使用多周期路径进行约束

```
set_multicycle_path 2 -setup -from [get_clocks CLKP] -to [get_clocks CLKM] -
start
set_multicycle_path 1 -hold -from [get_clocks CLKP] -to [get_clocks CLKM] -
start
# -start 选项 表示强制移动的为 start clock 即 launch clock
```

> ***此时使用 -start 选项表示对起始的快时钟进行约束，建立时间检查时将其延后 2 个时钟周期，保持时间检查使用第 0 个边沿***



整数倍时钟

存在整数倍关系的时钟时，STA工具取最大的时钟

非整数倍时钟

存在非整数倍关系的时钟时，STA工具取其最小公倍数进行分析