STA环境

> 给整个静态时序分析指定正确、精确的约束 -> 指导静态时序分析工具对设计进行全面、严谨、正确的检查

STA主要针对同步电路，对异步电路无能为力

1. 定义时钟（Specifying Clocks）
    - Clock Source

      整个Design的port & Design中某cell的pin
    - Period
    - Duty Cycle
    - Edge Times：Times of the rising edge and the falling edge

      -waveform选项只需要设置一个周期中的上升沿和下降沿的时间即可

```
# SDC 控制语句
create_clock -name SYSCLK -period 20 -waveform {0 5} [get_ports SCLK]
```

2. 时钟不确定性（Clocks Uncertainty）

> 对于一个时钟周期中的时序不确定性，时钟边沿可能会在一定的时间范围中出现
>
> 使用***set_clock_uncertainty***命令指定
>
> 此时，时序分析过程中悲观度会有所增加，分析会更加严格，同时也会增加电路的稳健性

```
# 使用 set_clock_uncertainty 指令分别对 Tsetup 和 Thold 做出指定
set_clock_uncertainty -setup 0.2 [get_clocks CLK_CONFIG]
set_clock_uncertainty -hold 0.05 [get_clocks CLK_CONFIG]
```

Clocks-Clock Uncertainty的产生原因：

1. Clock Skew（时钟偏差）
2. Clock Delay（时钟延迟）
3. Clock Jitter（时钟抖动）

3. 时钟延迟（Clocks-Clock Latency）

> 时钟网络中存在一定的延迟，主要分为两部分：
>
> 1. Source Latency
>
>   从时钟源点到定义的时钟节点造成的延时
> 2. Network Latency
>
>   从定义的时钟节点到驱动的触发器的时钟网络造成的延时
>
> - 一旦时钟树确定（Place & Route后），network latency 可以被忽略，而source latency则会保留，使用 ***set_propagated_clock***得到时钟延迟
> - 在PR（Place & Route）前（DC，Design Compile 过程中），时钟的latency只能是估计值

```
# 指定时钟延迟
set_clock_latency 0.8 [get_clocks CLK_CONFIG]
set_clock_latency 1.9 -source [get_clocks SYS_CLK]

# 指定时钟延迟的范围
set_clock_latency 0.851 -source -min [get_clocks CFG_CLK] # 设定最小时钟延迟
set_clock_latency 1.322 -source -max [get_clocks CFG_CLK] # 设定最大时钟延迟
```

4. 生成时钟（Generated Clocks）

> Master Clock通常会通过分频产生一系列新的时钟，即Generated Clock
>
> - Generated Clock和Source Clock在处理时会有什么区别

```
# Source Clock
create_clock -name CLK_P 10 [get_pins UPLL0/CLKOUT]
# Generated Clock
create_generated_clock -name CLK_P_DIV2 -source UPLL0/CLKOUT -devide_by 2 [get_pins
UFF0/Q]
################ 通过 -source 选项指定源时钟
################ 通过 -devide_by 选项指定与源时钟的倍数关系
```

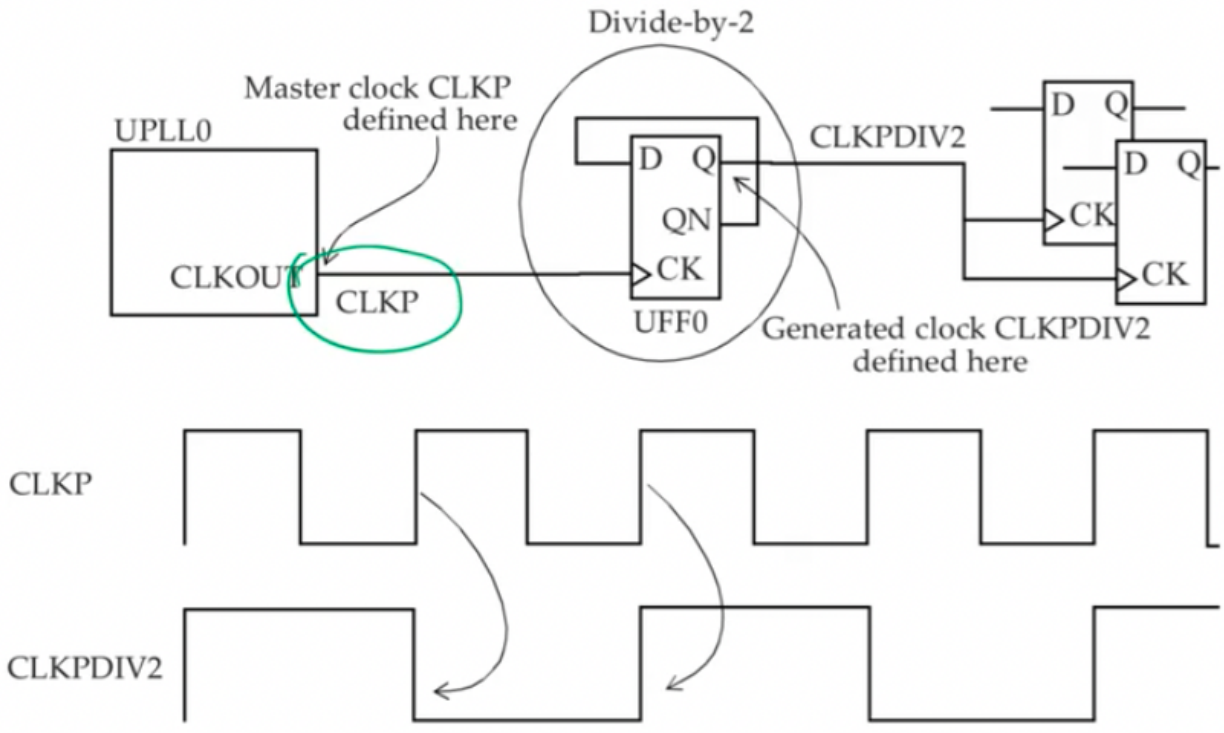


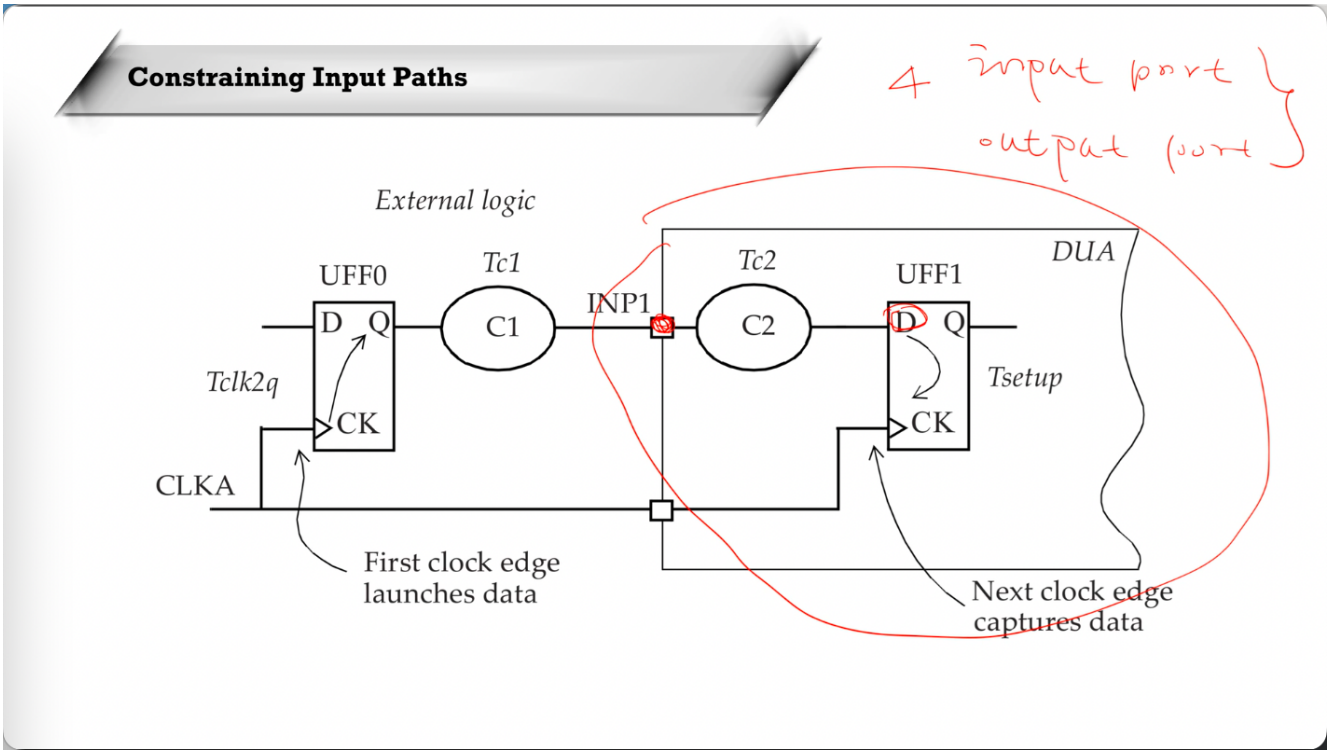
Figure 7-10 *Generated clock at output of divider.*

> - 对于一个新时钟，将其定义为master clock（使用create_clock指令）也是可以的，但是可能会出现一些问题 - ***Master Clock和其Generated Clock之间存在时钟继承性***，使用create_generated_clock命令时系统会当作同步时钟系统，而定义为master clock时，系统认为两个时钟独立，系统当作异步时钟对待 静态时序分析工具不会去分析，可能会造成时序问题

> - 使用逻辑电路经过运算产生的新时钟，一般建议定义为新的master clock，因为与源时钟继承性较弱

5. 约束输入与输出路径（Constraining Input Paths & Output Path）

   时序路径中并不完全是 ***reg-to-reg*** 的，对于起点是 ***input port*** 或终点是 ***output port*** 的时序路径，需要使用 ***input delay*** 和 ***output delay*** 进行约束



Port-to-reg的时序路径，从port到reg的路径中并没有时钟信号（Reg-to-port的时序路径与之相同）

```
# Input Delay Constrain
set_input_delay -clock CLKA -max [expr Tclk2q + Tc1] [get_ports INP1]
# 引脚虽然含有延时，但需要指定其处于哪一个时钟域 - 使用 -clock 选项进行指定
```

引脚虽然含有延时，但需要指定其处于哪一个时钟域 - 使用 -clock 选项进行指定

- 设计时也可以针对建立时间与保持时间分别进行约束

```
# 使用-max和-min选项分别对建立时间与保持时间给出严格约束
```

> Output Delay与之类似
>
> ***时序路径起点为CLK，终点为Port***

```
set_output_delay -clock CLKQ -max [expr Tc2 + Tsetup] [get_ports OUTB]
# 也可以使用-max和-min分别对其建立时间和保持时间进行约束
```

6. 时序路径组（Timing Path Groups）

> 通常来说 ***PT*** 工具判断时钟分组的方法是 ***看时序路径的终点是属于哪一个时钟***

7. 扩展属性模式（Modeling of External Attributes）

> 对于 ***input delay*** 和 ***output delay*** 还存在额外的属性
>
> - ***input delay*** （三种指定其一即可）
>
> 对于 ***input path*** 中，需要考虑 ***前一级cell产生的delay***，否则会按照 0 进行分析，与实际情况存在偏差
>
> - ***set_drive***
>
> ```
> # Delay_to_first_gate = (drive * load_on_net) + interconnect_delay
> set_drive 100 UCLK
> # 表示指定在input port路径阻抗为100，可以提现前一级电路的驱动能力，值越小代表其驱动强度越大，0表示无限大
> # 实际上并不是每一个IO的drive都是理想的，为了贴近实际值，使用set_drive命令进行指定
>
> # 也可以使用上升和下降沿的时间进行指定
> set_drive -rise 3 [all_inputs]
> set_drive -fall 2 [all_inputs]
> ```
>
> - ***set_driving_cell***
>
> > 相对于 ***set_drive*** 命令更简单的方法
> >
> > 从库中取一个cell的模型，通过 ***set_driving_cell*** 使用其驱动强度进行指定
>
> ```
> # 使用 库（slow）中的 单元（INV3）进行驱动强度指定
> set_driving_cell -lib_cell INV3 -library slow [get_ports INPB]
>
> # 另外两种不同的脚本写法
> set_driving_cell -lib_cell INV2 -library tech13g [all_inputs]
> set_driving_cell -lib_cell BUFFD4 -library tech90gwc [get_ports {testmode[3]}]
> ```
>
> - ***set_input_transition***
>
> > 直接指定input的传输时间
>
> ```
> set_input_transition 0.85 [get_ports INPC]
> set_input_transition 0.6 [all_inputs]
> set_input_transition 0.25 [get_ports SD_DIN*]
> ```
>
> - ***output delay***
>
> > 对于 Output Path，关注的更多是其负载电容

> 若不对其进行约束，则认为其输出负载为0，与实际情况存在较大偏差

- ***set_load***

```
# 指定输出port OUTX上的负载电容为5pF
set_load 5 [get_ports OUTX]

# 使用 -pin_load 指定从 pin 到 port的负载
set_load -pin_load 0.007 [get_ports {shift_write[31]}]

# 使用 -wire_load 指定 net 连接到 port的负载
# 若在 set_load 指令中没有使用 -pin_load 或 -wire_load 特殊指定是是 pin-to-port的负载还是 net-connecting-to-port的负载，则默认为 -pin_load
```

- 在 ***set_load*** 命令中引用库进行约束

```
set_load [get_attribute [get_lib_lins tech_lib/NAND2/A] pin_capacitance] [all_outputs]
```

8. 设计规则检查（DRC，Design Rule Check）
   - ***set_max_transition***

```
# 对 IOBANK 设置 600pf 的限制
set_max_transition 0.6 IOBANK
```

   - ***set_max_capacitance***

```
# 对当前 design 中的所有 net 设置最大 capacitance 为 0.5pf
set_max_capacitance 0.5 [current_design]
```

   - ***set_max_fanout***

> Design 的最大扇出约束

   - ***set_max_area***

> Design 的最大面积约束

9. 虚拟时钟（Virtual Clock）

> 时钟与pin或design没有关系，主要用来作为静态时序分析的参考
>
> ***虚拟时钟是与电路中的节点没有关系的***

```
# 由于虚拟时钟与电路中的节点没有关系，只是作为参考
# 在定义虚拟时钟时，不需要使用 get_ports 命令指定端口
create_clock -name VIRTUAL_CLK_SAD -period 10 -waveform {0 5}
```

> ***使用虚拟时钟定义与其相关的时序路径***

```
set_input_delay -clock VIRTUAL_CLK_SAD -max [get_ports ROW_IN]
```

10. 改进时序分析（Refining the Timing Analysis）

> 精确设置会简化时序分析
>
> ***若设置的有问题，则会使时序分析更加复杂***

- ***set_case_analysis***

  > 举例说明：静态时序分析时，DFT其实是不工作的
  >
  > reg使用选择器选择接入时钟为SYS_CLK或Scan_CLK，选择信号为Test

  ```
  # 将 Test 信号接入0，指定 reg接入的时钟为 SYS_CLK
  set_case_analysis 0 Test

  # 分别对信号指定其实际接入值，得到想要的情况
  set_case_analysis 0 UCORE/UMUX0/CLK_SEL[0]
  set_case_analysis 1 UCORE/UMUX0/CLK_SEL[1]
  ```

- ***set_disable_timing*** - 指定该位置没有 Timing Arc

  > 举例说明：时钟控制的二选一数据选择器，实际上不是一条时序路径
  >
  > 本质上是一个组合逻辑电路

  ```
  # S 为数据选择器的地址输入端，Z为数据输出端，指定在 cell UMUX0中，S到Z不是一条时序路径
  set_disable_timing -from S -to Z [get_cells UMUX0
  ```

- ***set_false_path***

  > 跨时钟域的异步电路，不进行分析

  ```
  # 从 USBCLK 到 MEMCLK 的时序路径都不进行分析
  set_false_path -from [get_clocks USBCLK] -to [get_clocks MEMCLK]
  ```

- ***set_multicycle_path***

  > 一个周期无法满足***数据路径中组合逻辑的延时***，告诉PT工具 ***在指定周期对其进行检查***

  ```
  # 指定在第三个时钟周期检查从 UFF0/Q 到 UFF1/D 的数据时序
  set_multicycle_path 3 -setup -from [get_pins UFF0/Q] -to [get_pins UFF1/D]
  ```