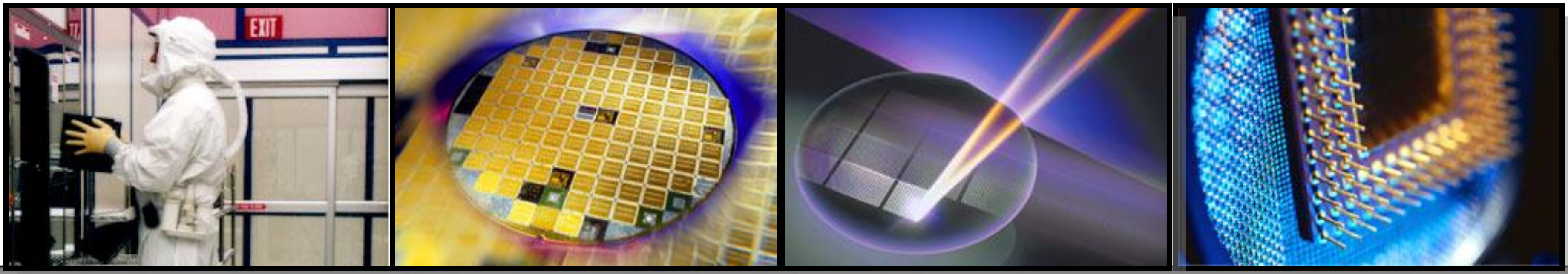
# FPGA时序约束方法



# 课程安排

- 时序约束的目的
- 时序约束的内容
- Xilinx FPGA时序约束方法
- Altera FPGA时序约束方法
- 时序约束的原则

# 课程安排

- **时序约束的目的**
- 时序约束的内容
- Xilinx FPGA时序约束方法
- Altera FPGA时序约束方法
- 时序约束的原则

# 时序约束

- 规范设计的时序行为，表达设计者期望满足的时序条件，指导综合、布局布线
  - –过紧的时序约束会延长编译时间
  - –不合理的约束可能会使工具停止工作
  - –利用时序分析报告来判断约束是否可行
- 在设计实现后，查看布局布线后静态时序报告判断是否达到预定的性能目标
  - –如果约束未满足，利用时序报告确定原因

# 为何要进行时序约束？

- 设计工具不能自动实现获得最佳速度的布局和布线方式，因此需要用户设定性能目标，让工具去实现

- 用户设定的性能目标由时序约束体现
  - –时序约束提高设计性能的途径是将逻辑尽可能放的近，从而使用尽可能短的布线资源

# 什么情况需要做时序约束

- 当设计仅有一个时钟信号，且频率低于50MHz，逻辑电路简单（7级以下），不需要对设计进行时序约束。
- 当设计超过50MHz，或者设计较为复杂时，需要进行时序约束。

# 约束的基本作用

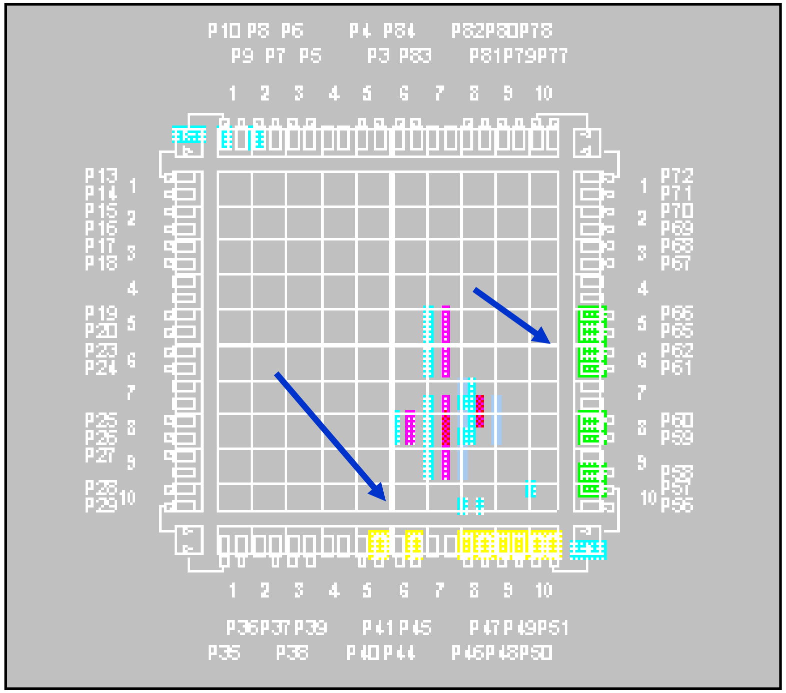
- 提高设计的工作频率
  - 通过附加约束可以控制逻辑的综合、映射、布局和布线，以减小逻辑和布线延时，从而提高工作频率。
- 获得正确的时序分析报告
  - FPGA设计平台包含静态时序分析工具，可以获得映射或布局布线后的时序分析报告，从而对设计的性能做出评估。
  - 静态时序分析工具以约束作为判断时序是否满足设计要求的标准。
- 指定FPGA引脚位置与电气标准
  - FPGA的可编程特性使电路板设计加工和FPGA设计可以同时进行，而不必等FPGA引脚位置完全确定，从而节省了系统开发时间。
  - 通过约束还可以指定I/O引脚所支持的接口标准和其他电气特性。

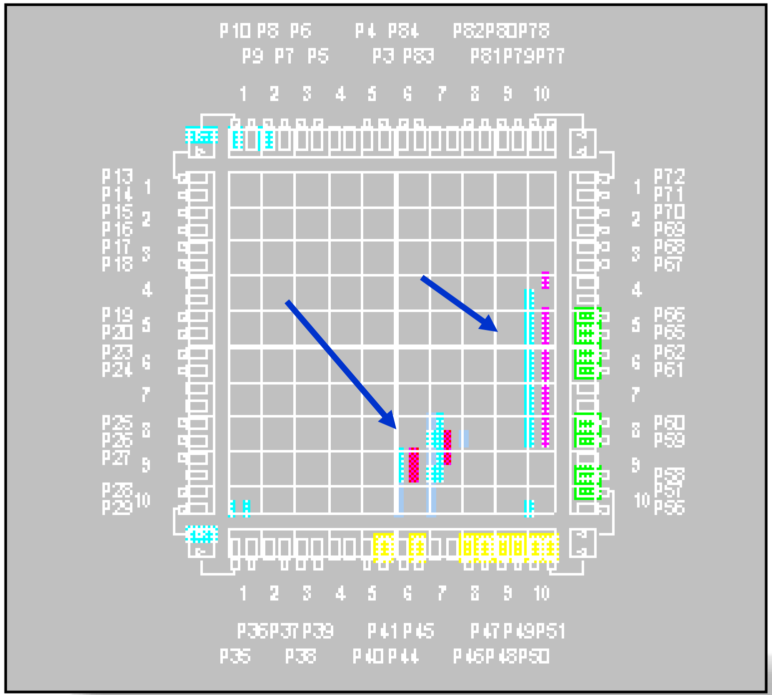
# 时序约束对FPGA设计影响

- EDA工具不会试图寻找达到最快速度的Place & Route 结果
  - 施加时序约束后，implementation工具才会尝试满足性能期望
- 你对设计性能的期望是通过设计时序约束传递给EDA工具的
  - 让相关逻辑尽量靠近，从而减小布线延迟。通过这个方法，时序约束试图满足你的性能要求

# 时序约束的影响

- Without global timing constraints
  - Logic tends to be grouped to improve internal timing at the expense of I/O timing
- With global timing constraints
  - All timing paths are evaluated
  - I/O paths are improved (CLBs are place closer to I/O pins)





# 用时序约束定义时序的目标

- 时序约束定义时序目标
  - Over-constrain需要额外的布局布线时间
  - 尝试尽量使用时序约束，即使在时序要求在中等情况下
- 非现实 的时序约束将会使工具停下来
  - 综合工具的timing report 和Post-Map Static Timing Report包含性能估计
  - 都告诉了约束是否符合现实
- 在工具完成流程后，需要审核 Post-Place & Route Static Timing Report to 来确定目标是否满足
  - 如果时序不满足, 根据Timing Report找到原因

# 课程安排

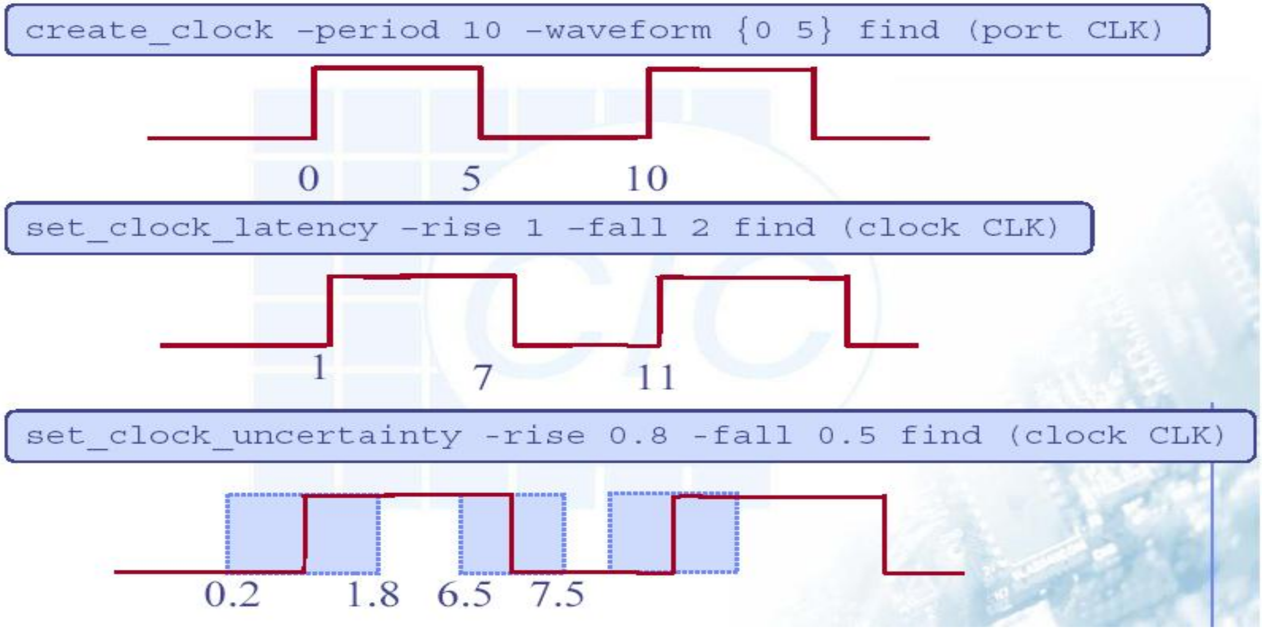
- 时序约束的目的
- **时序约束的内容**
- Xilinx FPGA时序约束方法
- Altera FPGA时序约束方法
- 时序约束的原则

# 时序约束的内容

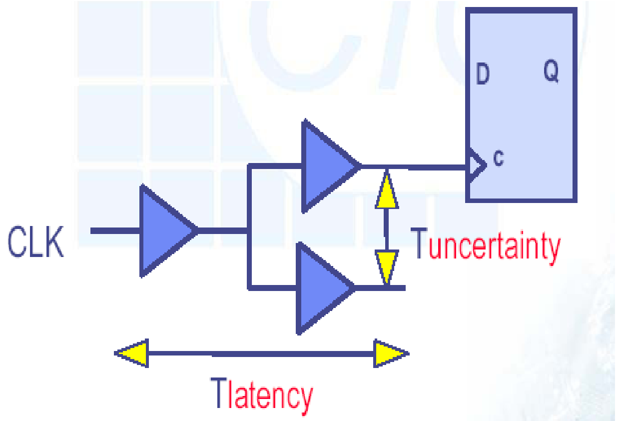
- 时钟定义：包含所有的时钟
- 输入路径延迟
- 输出路径延迟
- 多周期路径
- 异步电路中的虚假路径

# 时钟定义

- 时钟周期



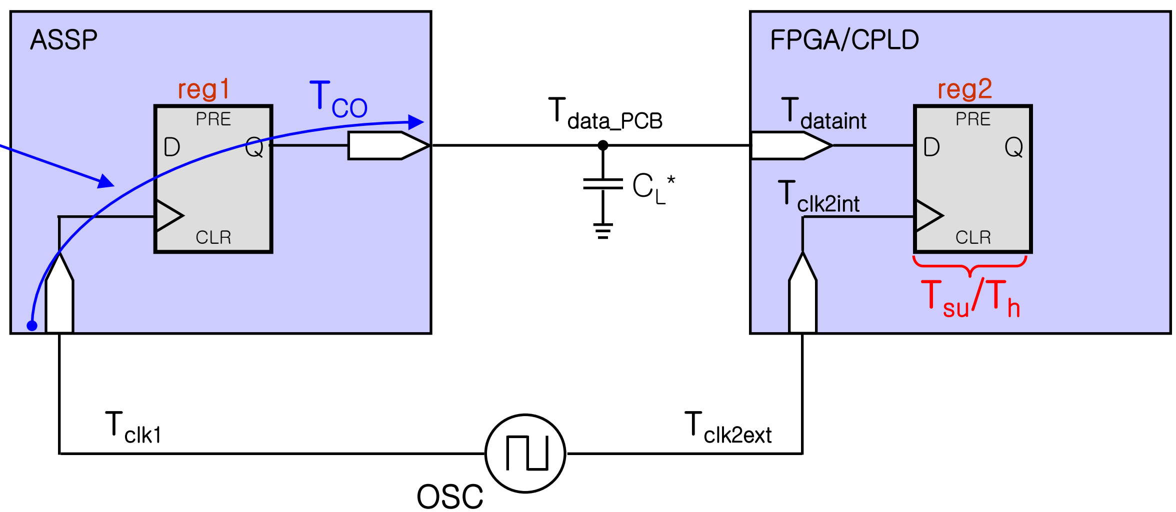

- Clock latency & uncertainty



# 输入路径延迟

- Need to specify timing relationship from ASSP to FPGA to guar antee setup/hold in FPGA

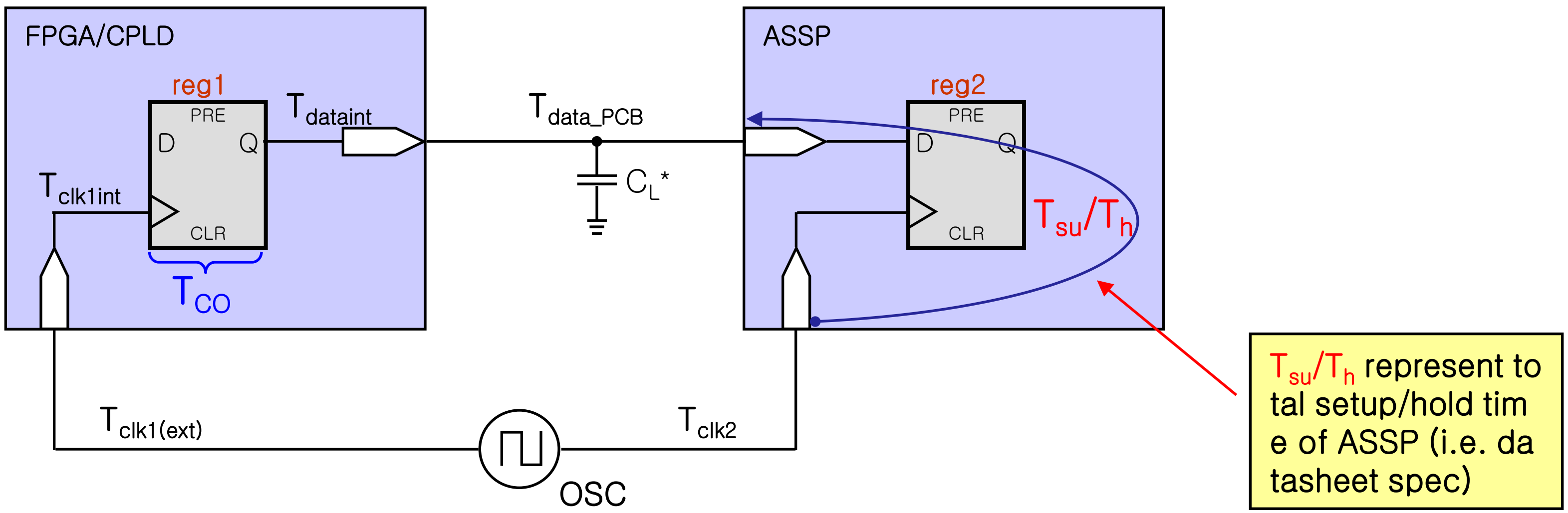
**$T_{co}$ represents total clock-to-output time of ASSP (i.e. datasheet spec)**




** Represents delay due to capacitive loading*

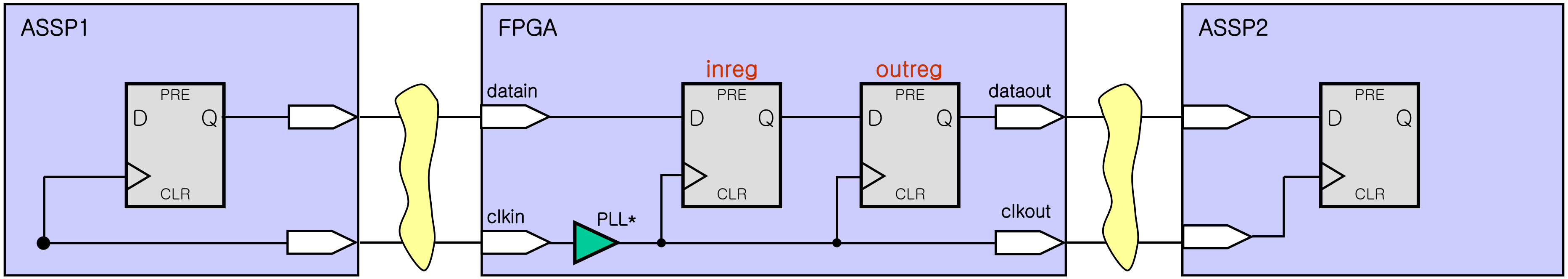
# 输出路径延迟

- Need to specify timing relationship from FPGA to ASSP to guar antee clock-to-output times in FPGA




* *Represents delay due to capacitive loading*

# Source-Synchronous接口

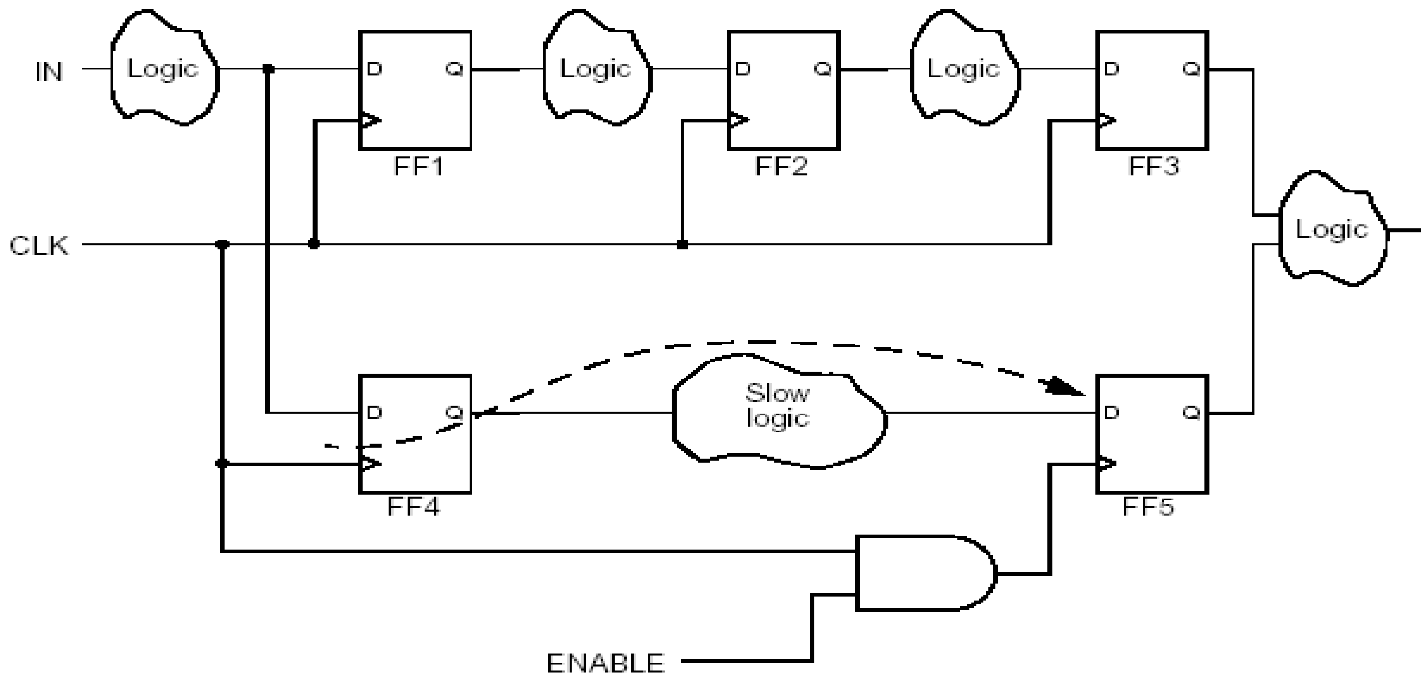



- Both data & clock transmitted by host device with designated ph ase relationship (e.g. edge or center-aligned)
  - No clock tree skew included in calculation
  - Target device uses transmitted clock to sample incoming data

- Data & clock routed identically to maintain phase relationship at destination device
  - Board delay not included in external delay calculations
    - Clock trace delay (data required time) & Data trace delay (data arrival time) are equal and offset
  - Enables higher interface speeds (compared to using system clock)

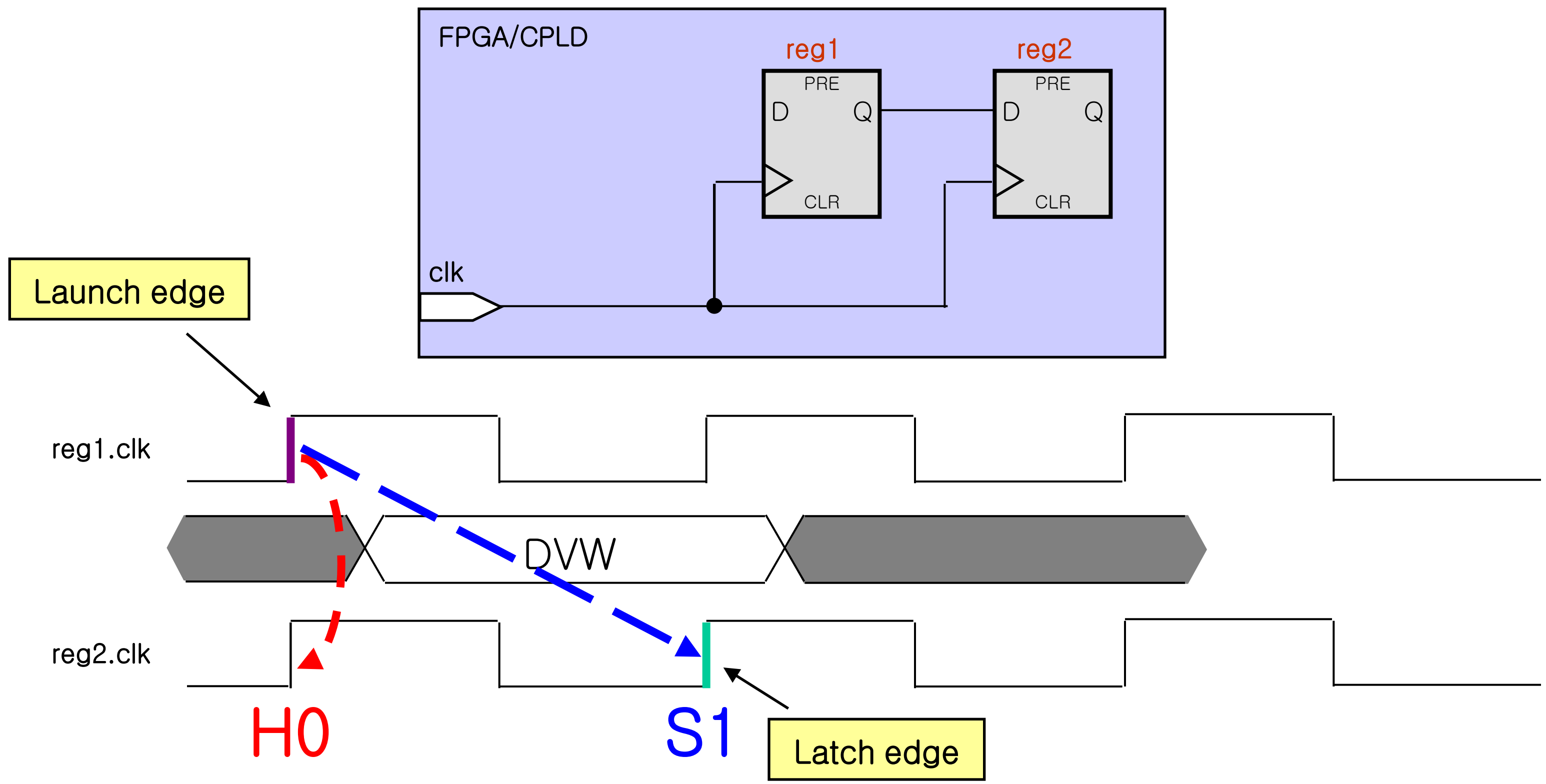
# 多周期Multi-cycle 路径

- 多周期路径是指从launch到latch数据超过一个时钟周期



# Multicycle (1)

Standard single-cycle register transfer




Multicycle Setup = 1 (Default)

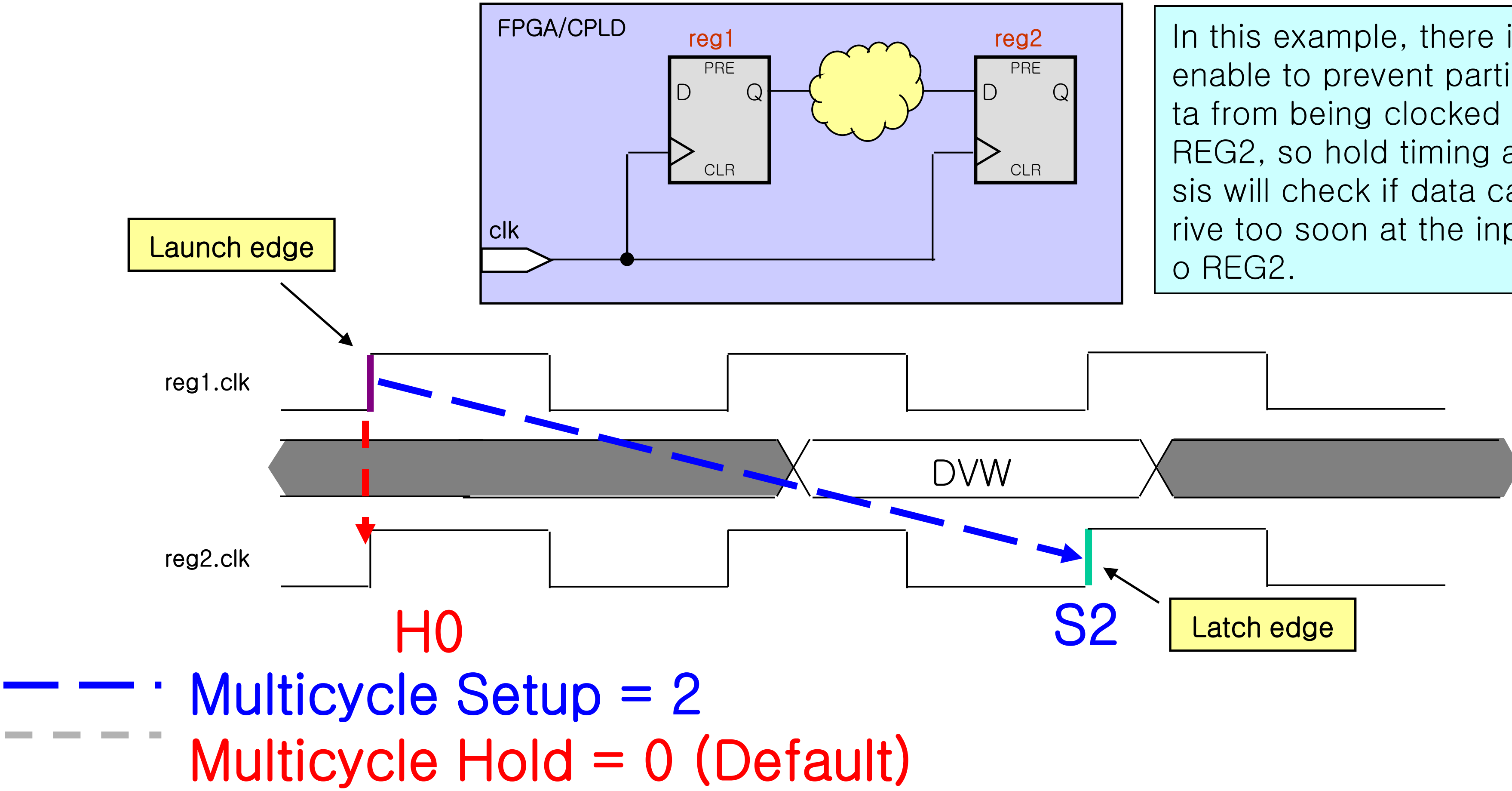
Multicycle Hold = 0 (Default)*


18

# Multicycle (2)

Change to a *two cycle setup*; *single cycle hold* transfer

In this example, there is no enable to prevent partial data from being clocked into REG2, so hold timing analysis will check if data can arrive too soon at the input to REG2.

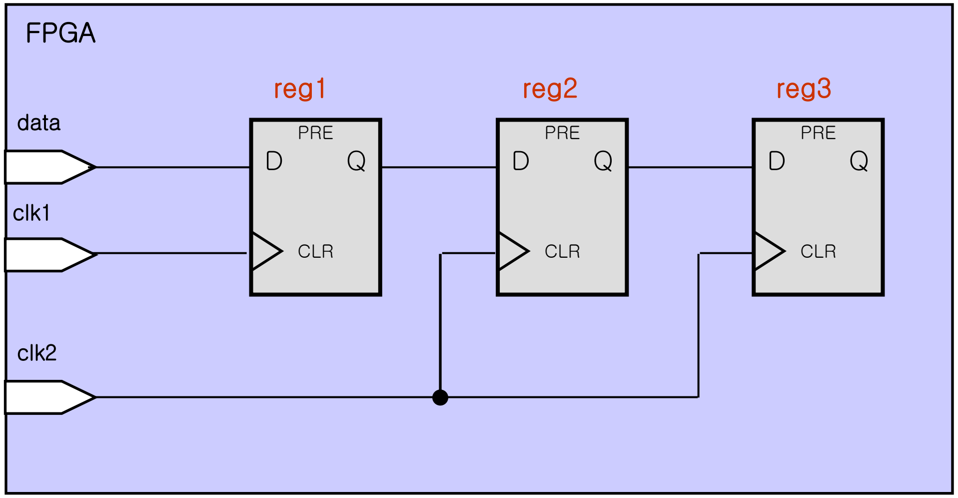




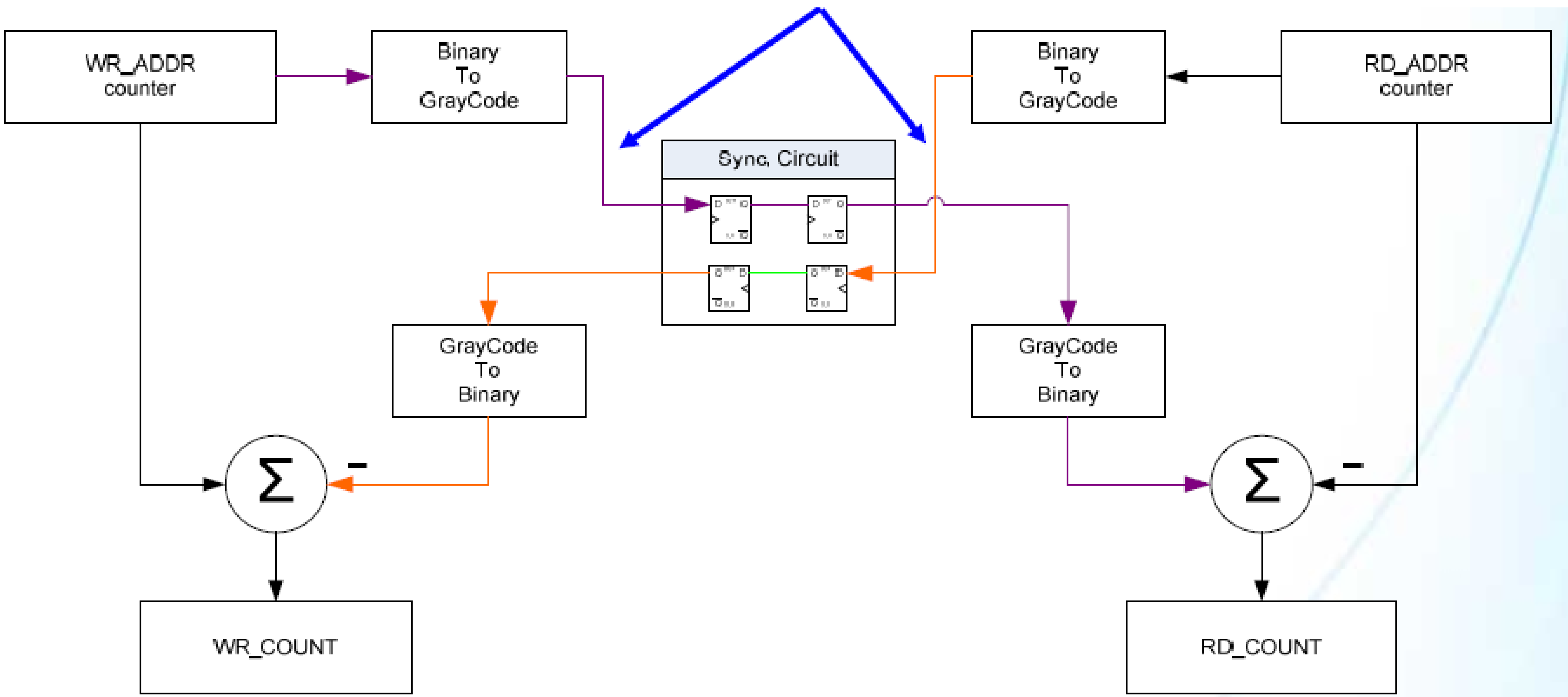
19

# 虛假路徑

- A false path  is the path which is never sensitized/cared due to the logic configuration, expected data sequence, or operating mode.
- It will not be checked in STA



# 异步FIFO中的虚假路径



# 课程安排

- 时序约束的目的
- 时序约束的内容
- **Xilinx FPGA时序约束方法**
- Altera FPGA时序约束方法
- 时序约束的原则

# Xilinx FPGA时序约束方法（1）

- 时序约束覆盖的基本路径包括：
  - 输入路径（Input paths）
  - 同步元件路径（Synchronous element to synchronous element paths）、
  - 输出路径（Output paths）
  - 特殊路径（Path specific exceptions）
- 最有效的办法是从全局约束开始，然后根据需要做特定路径约束。FPGA实现工具在时序约束的驱动下，完成映射、布局和布线，最终实现时序目标。
- 约束编辑器提供了一个统一的界面管理设计的所有时序约束，并且提供图形化的界面简化输入过程。

# Xilinx FPGA时序约束方法（2）

- 全局时序约束条件为设计中的所有组合路径设置时序要求，全局约束条件覆盖整个设 计。设计需要的基本时序约束主要包括：
  - 每个时钟的全局周期约束（Global Period）
  - 全局输入偏移约束（Global OFFSET IN)
  - 全局输出偏移约束（Global OFFSET OUT）
- 为了提高约束准确性，还可以使用特殊路径约束
  - 多周期路径（Multi-cycle Path）
  - TIG（Timing Ignore）
- 在定义时序例外路径时，推荐按照实际的需求设置时序约束值。过约束会造成布局和布线时间过长以及增加资源占用等问题，甚至反而降低系统性能。

# 建立Timing Constraints

- 建立Timing constraints需要两步

  **Step 1:** 按照path endpoints进行分组

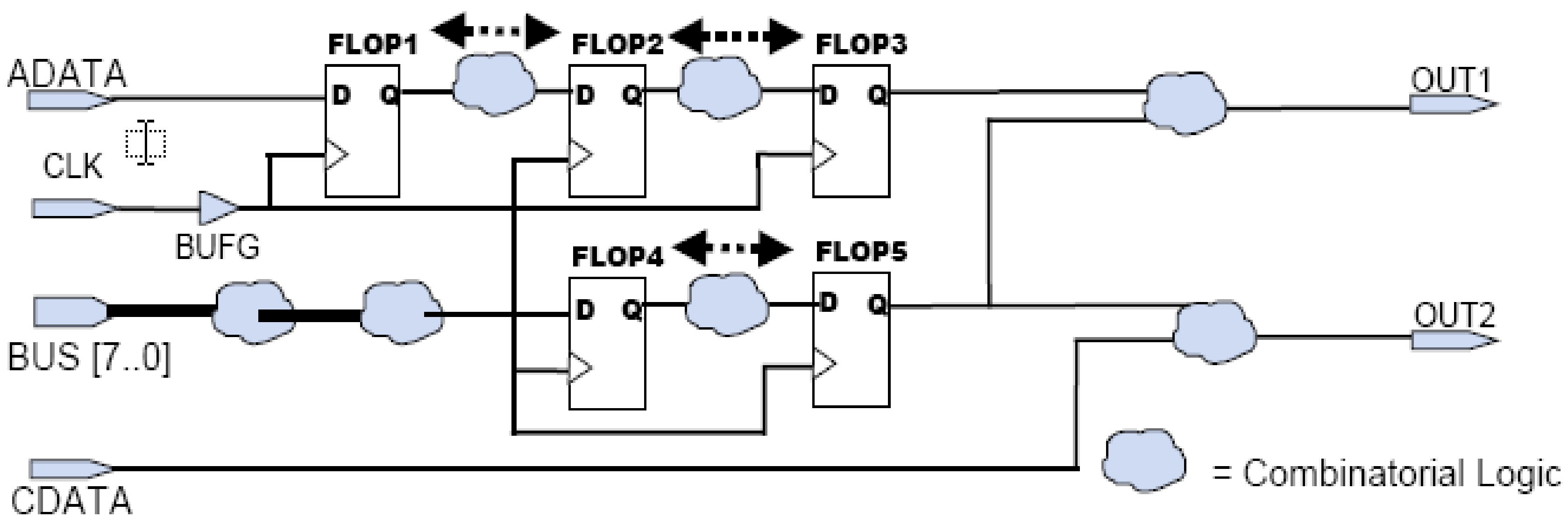
  Synchronous element I/O pads

  **Step 2:** 在组与组之间指定时序要求

- Global timing constraints use a default grouping of path endpoints which makes it easy to constrain your design

# 周期约束

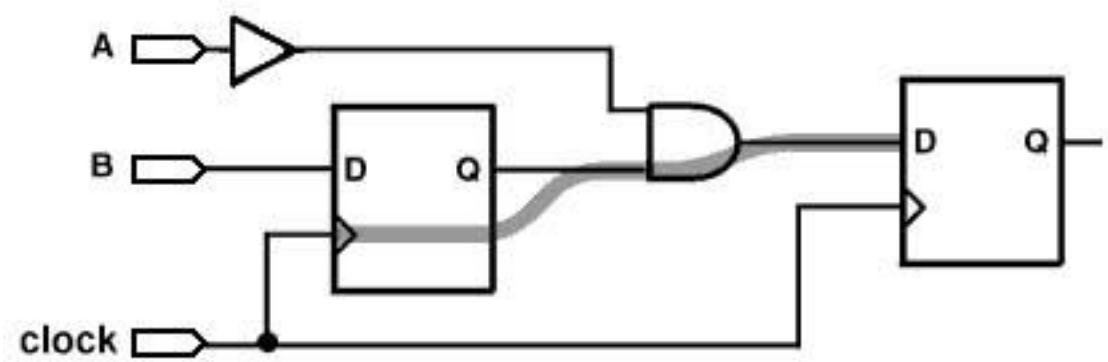
- 周期(PERIOD)指参考网络为时钟的同步元件间的路径，包括：flip-flop、latch、synchronous RAM、DSP48等。
- 周期约束不会优化以下路径：
  - 从输入管脚到输出管脚之间的路径纯组合逻辑
  - 从输入管脚到同步元件之间的路径
  - 从同步元件到输出管脚的路径

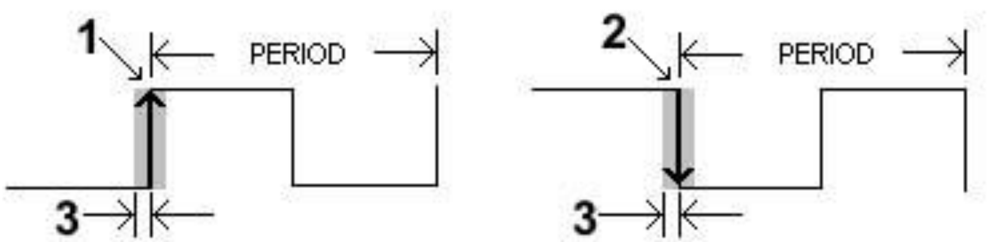



周期约束路径示意图

# 周期约束：精确的时间信息

- 源flip-flop和目的flip-flop之间的clock skew
- 时钟负沿动作的同步元件
- 占空比不等
- 输入时钟抖动(jitter)

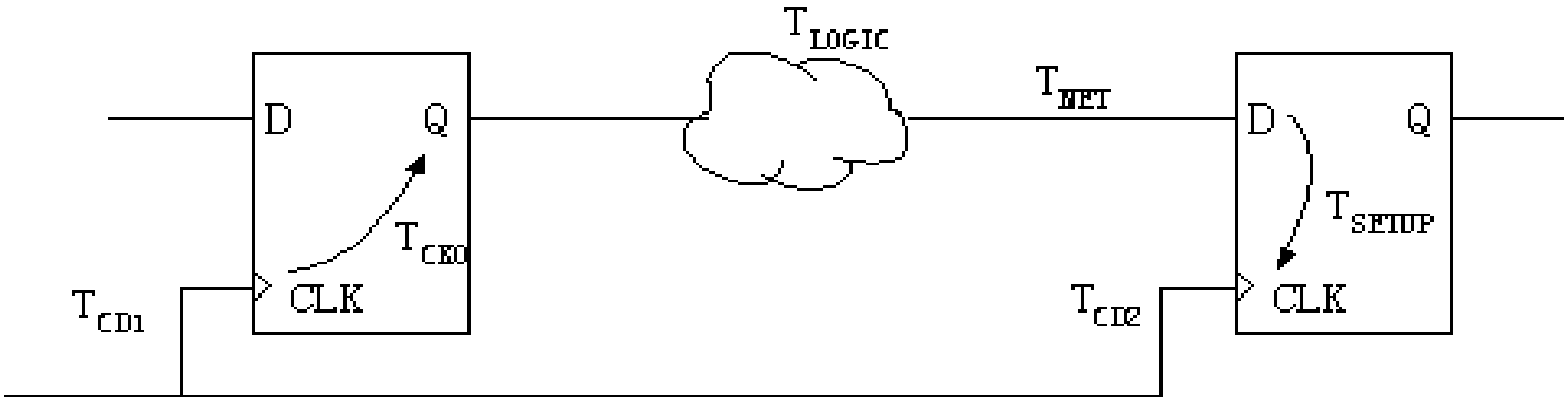


# 周期约束

- 周期约束是一个基本时序和综合约束，它附加在时钟网线上，时序分析工具根据周期约束检查与同步时序约束端口相连接的所有路径延迟是否满足要求。
- 周期是时序中最简单也是最重要的含义,后面要讲到的其它时序约束都是建立在周期约束的基础上的，很多其它时序公式，可以用周期公式推导。
- 在附加周期约束之前，首先要对电路的时钟周期有一定的估计，不能盲目上。约束过松，性能达不到要求，需要有少量余量；约束过紧，会大大增加布局布线时间，甚至效果相反。

# 周期约束




- 周期约束的计算
  - 设计内部电路所能达到的**最高运行频率**取决于同步元件本身的建立保持时间，以及同步元件之间的逻辑和布线延迟。
  - 时钟的最小周期为：

**Tperiod= Tcko +Tlogic +Tnet +Tsetup-Tclk_skew**

**Tclk_skew =Tcd2-Tcd1**

其中Tcko为时钟输出时间，Tlogic为同步元件之间的组合逻辑延迟，Tnet为网线延迟。Tsetup为同步元件的建立时间，Tclk_skew为时钟信号偏斜。

# 周期约束

- 设计中的每个时钟都应该定义周期约束。定义周期约束的优选方式是使用TIMESPEC周期约束语句。

- TIMESPEC周期约束语句能够定义衍生时钟关系，如“DLL/DCM/PLL/BUFR/PMCD”时钟变换元件输出时钟的周期约束。衍生时钟的TIMESPEC周期约束需要根据其源时钟的TIMESPEC周期约束来定义。衍生时钟是相关的。

# 周期约束实例1

- 附加周期约束的一个例子：

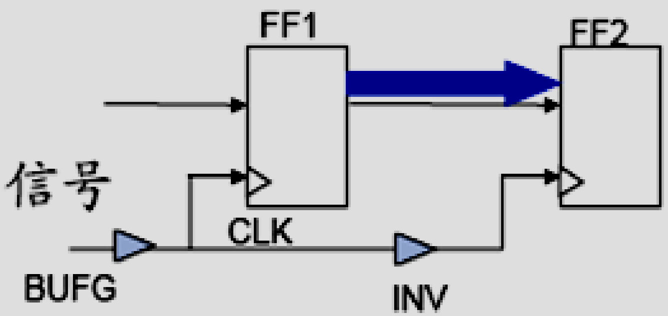
  **TIMESPEC SYS_CLK PERIOD=10ns HIGH 4ns**

  这个约束将被附加到SYS_CLK所驱动的所有同步元件上。

- PERIOD约束自动处理寄存器时钟端的反相问题，如果相邻同步元件时钟相位相反，那么它们间的延迟将默认限制为PERIOD约束值的一半。

假定：

- CLK上施加的是占空比为百分之五十的信号
- 周期约束为10 ns
- 因为FF2将要用CLK的下降沿触发，所以在这两个触发器之间实际的约束变为10 ns - 5 ns = 5 ns




反相时钟周期约束问题的例子

# 周期约束实例2

- TS_Period_1 和TS_Period_2是相关时钟域，TS_Period_2周期约束值是TS_Period_1周期约束值的2倍，则可以定义时序约束为：

  - TIMESPEC TS_Period_1 = PERIOD "clk1_in_grp" 20 ns HIGH 50%;
  - TIMESPEC TS_Period_2 = PERIOD "clk2_in_grp" TS_Period_1 * 2;

# DCM元件的输出时钟约束

- DCM输入端口上的约束将自动被转换为输出端口的周期约束。新产生的周 期约束覆盖所有与时钟变换模块相关的路径。

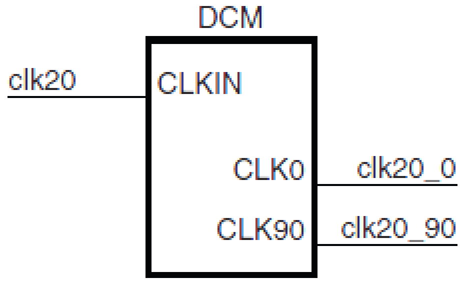
  TIMESPEC “TS_clk20” = PERIOD “clk20_grp” 20 ns HIGH 50 %

- CLK0输出端口上自动产生的新周期约束为：

TS_clk20_0=PERIOD clk20_0 TS_clk20*1.000000 HIGH 50.000000%

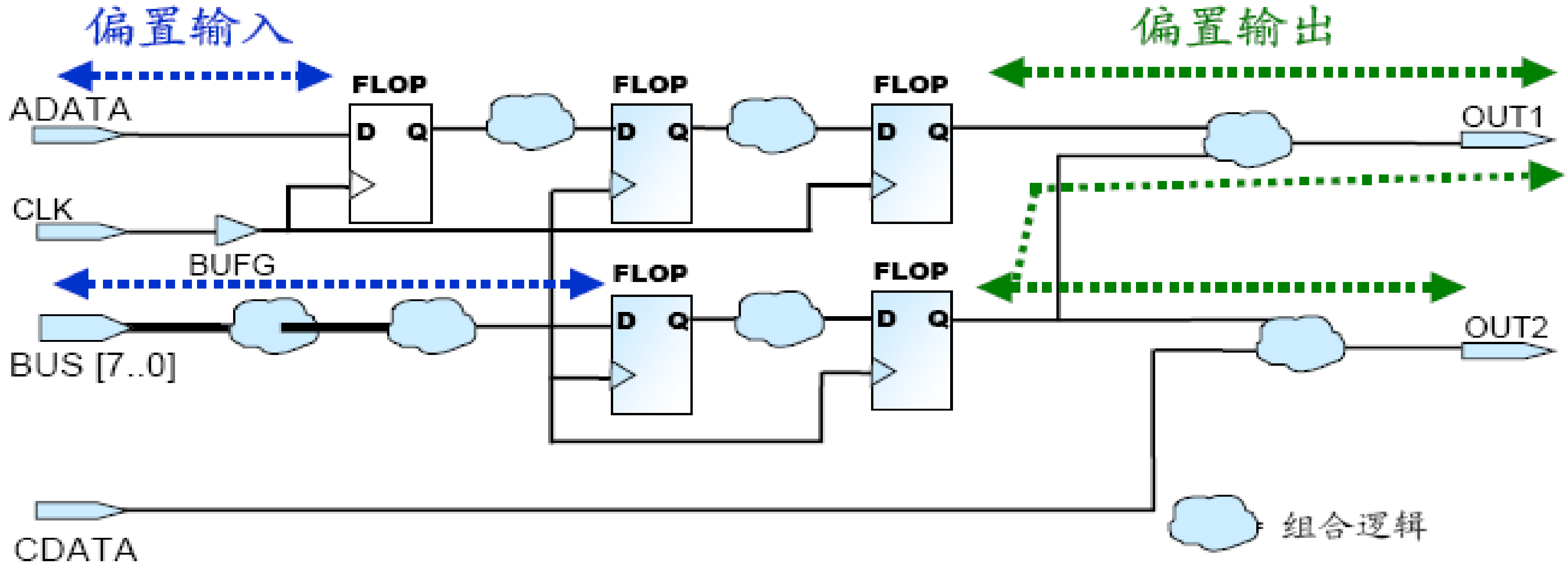
- CLK90输出端口上自动产生的新周期约束为：

TS_clk20_90=PERIOD clk20_90 TS_clk20*1.000000 PHASE + 5.000 000 nS HIGH 50.000000%



# 偏移约束OFFSET IN/OUT

偏移约束指数据和时钟之间的约束，偏移约束规定了外部时钟和数据输入输出引脚之间的时序关系，只用于与PAD相连的信号，不能用于内部信号。




偏移约束示意图

# 偏移约束

- 偏移约束优化以下时延路径
  - 从输入管脚到同步元件偏置输入(OFFSET IN)
  - 从同步元件到输出管脚偏置输出(OFFSET OUT)

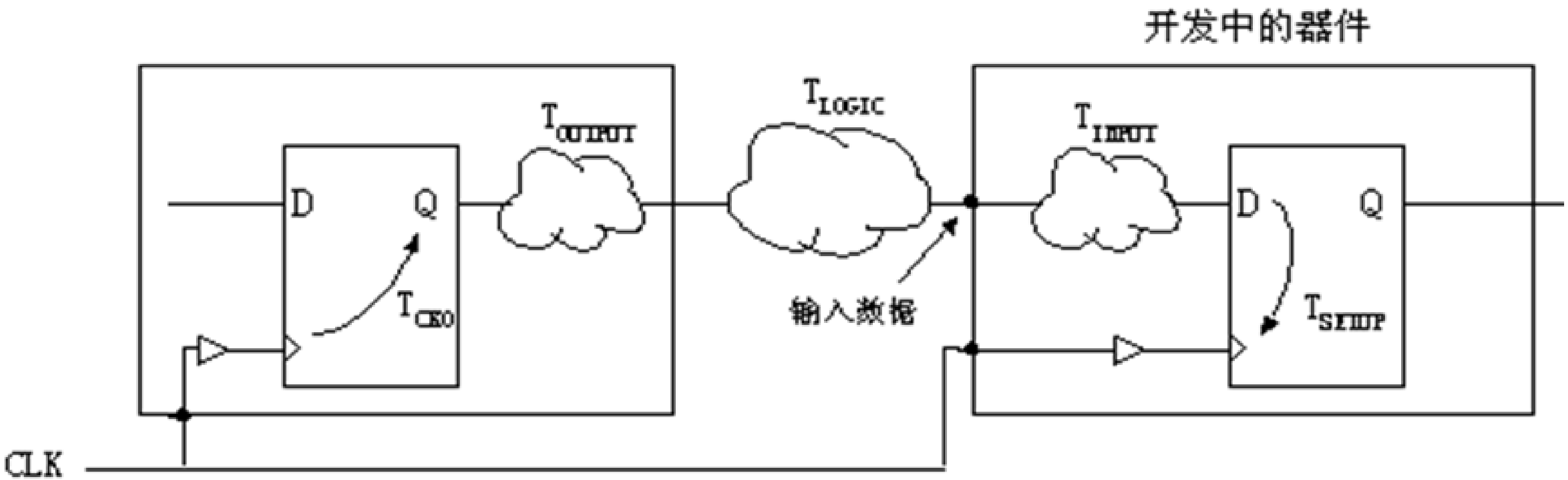
- 为了确保芯片数据采样可靠和下级芯片之间正确的交换数据，需要约束外部时钟和数据输入输出引脚之间的时序关系。偏移约束的内容的时刻，从而保证与下一级电路的时序关系。告诉综合器、布线器输入数据到达的时刻，或者输出数据稳定。

# 偏移约束 OFFSET IN

- ***OFFSET_IN_BEFORE***
  - 说明了输入数据比有效时钟沿提前多长时间准备好，用于约束芯片内部与输入引脚之间的组合逻辑。于是芯片内部与输入引脚的组合逻辑延迟就不能大于该时间（上限, 最大值），否则将发生采样错误。

- ***OFFSET_IN_AFTER***
  - 指出输入数据在有效时钟沿之后多长时间到达芯片的输入引脚，也可以得到芯片内部延迟的上限。

# 偏移约束OFFSET_IN

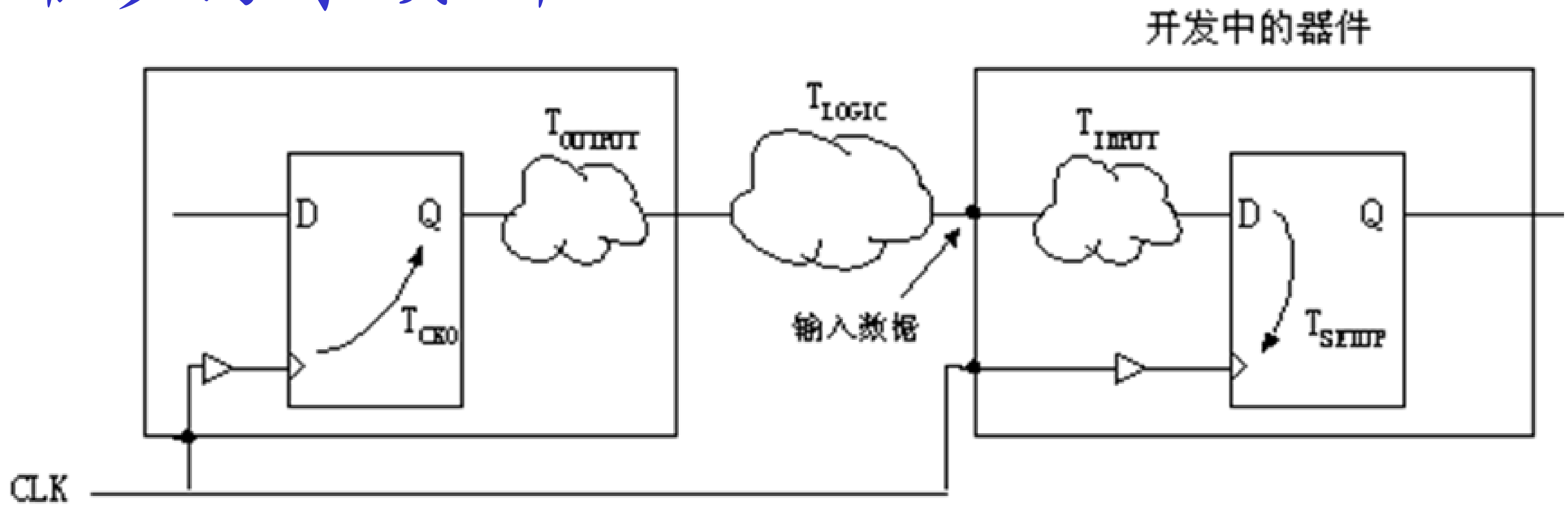



- 输入到达时间计算时序描述
  - 信号从上一级电路输出到达芯片输入引脚的延时计算公式：
    ***Tarrive=Tcko+Toutput+Tlogic***
  - 信号将在有效时钟沿之后的*Tarrive*时刻到达，约束设置为：
    **NET SIGNAL_IN OFFSET=IN *Tarrive* AFTER CLK**
  - 到达时间和时钟周期的*关系满足：*
    ***Tarrive +Tinput+Tsetup<Tperiod***

因此**NET SIGNAL_IN OFFSET=IN *Tdelay* BEFORE CLK**

***Tdelay<Tperiod-Tarrive***

# 偏移约束实例



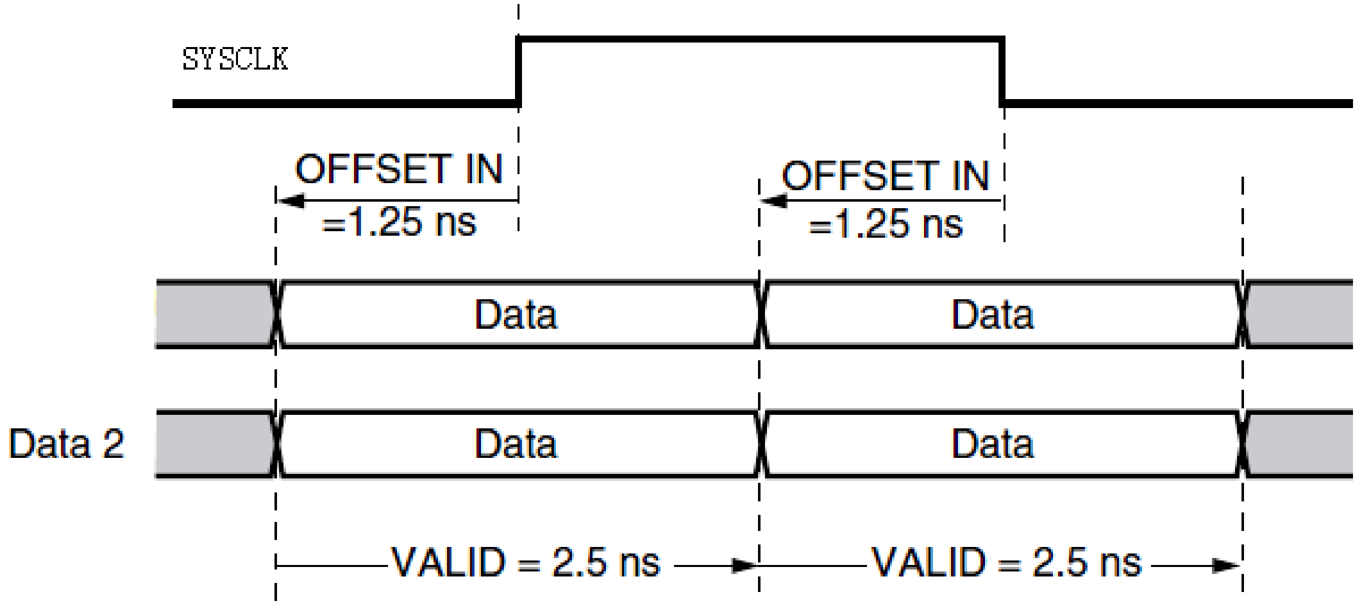

- 假设*Tperiod*=20ns，*Tcko*＝1ns，*Toutput*＝3ns，*Tlogic*＝8ns，请给出偏移约束。

- ***Tarrival=Tcko+Toutput+Tlogic = 12ns***,
  - 使用OFFSET_IN_AFTER进行偏移约束为：

    ***NET DATA_IN OFFSET=IN 12ns AFTER CLK***
  - 也可以使用OFFSET_IN_BEFORE进行偏移约束，它们是等价的：

    ***NET DATA_IN OFFSET=IN 8ns BEFORE CLK***

# DDR数据




NET "SYSCLk" TNM_NET = "SYSClk";

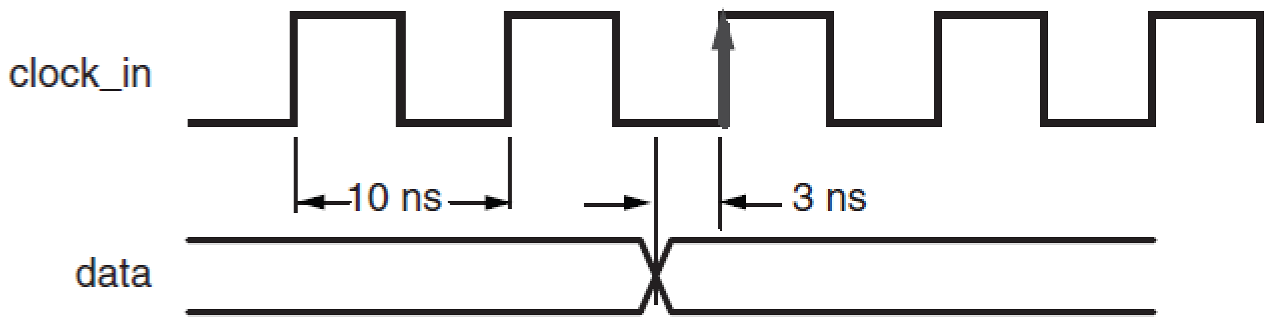
TIMESPEC "TS_SYSClk" = PERIOD "SYSClk" 5 ns HIGH 50%;

OFFSET = IN 1.25 ns VALID 2.5 ns BEFORE "SYSClk" RISING;

OFFSET = IN 1.25 ns VALID 2.5 ns BEFORE "SYSClk" FALLING;

# OFFSET_IN_BEFORE分析(1)

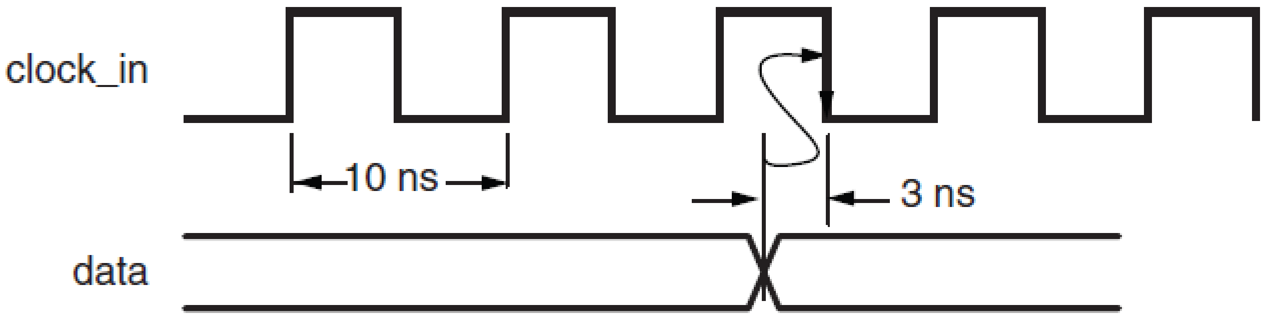
- TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
- OFFSET = IN 3 ns BEFORE clock;




```
Slack:                  -0.191ns (requirement - (data path - clock path
- clock arrival + uncertainty))
  Source:                 reset (PAD)
  Destination:            my_oddrA_ODDR_inst/FF0 (FF)
  Destination Clock:      clock0_ddr_bufg rising at 0.000ns
  Requirement:            3.000ns
  Data Path Delay:        2.784ns (Levels of Logic = 1)
  Clock Path Delay:       -0.168ns (Levels of Logic = 3)
  Clock Uncertainty:      0.239ns
...
```

# OFFSET_IN_BEFORE分析(2)

- TIMESPEC TS_clock = PERIOD clock 10 ns HIGH 50%;
- OFFSET = IN 3 ns BEFORE clock RISING;
- OFFSET = IN 3 ns BEFORE clock FALLING;

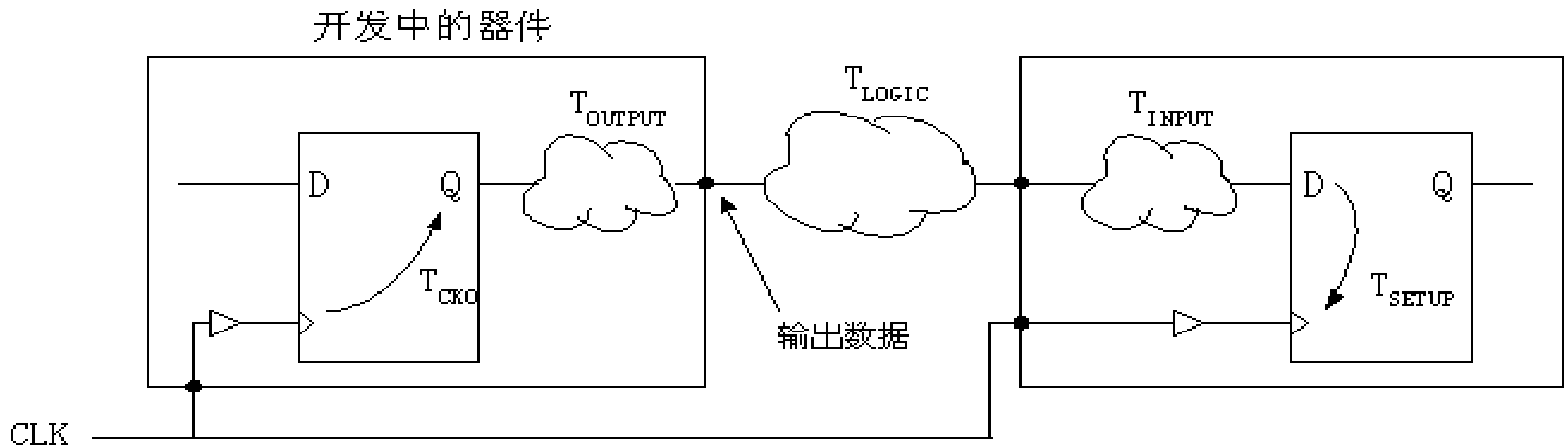



```
Slack:                  0.231ns (requirement - (data path - clock path
- clock arrival + uncertainty))
  Source:                DataD<9> (PAD)
  Destination:           TmpAa_1 (FF)
  Destination Clock:     clock0_ddr_bufg falling at 0.000ns
  Requirement:           3.000ns
  Data Path Delay:       2.492ns (Levels of Logic = 2)
  Clock Path Delay:      -0.038ns (Levels of Logic = 3)
  Clock Uncertainty:     0.239ns
…
```

# 偏移约束OFFSET_OUT

- *OFFSET_OUT_BEFORE*
  - 指出下一级芯片的输入数据应该在**有效时钟沿之前多长时间**准备好。
  - 从下一级的输入端的延迟可以计算出当前设计输出的数据必须在何时稳定下来，根据这个数据对设计输出端的逻辑布线进行约束，以满足下一级的建立时间要求，保证下一级采样数据稳定。
- *OFFSET_OUT_AFTER*
  - 规定本级输出数据必须在有效时钟沿之后多长时间(上限，最大值)稳定下来，芯片内部的输出延迟必须小于这个值。

# 偏移约束OFFSET_OUT




- 计算要求的输出稳定时间
  - 定义：*Tstable= Tlogic+Tinput +Tsetup*
  - 只要当前设计输出端的数据比时钟上升沿提前*Tstable*时间稳定下来，下一级就可以正确采样数据。
  - 实现工具将会努力使输出端的延迟*满足以下关系:*

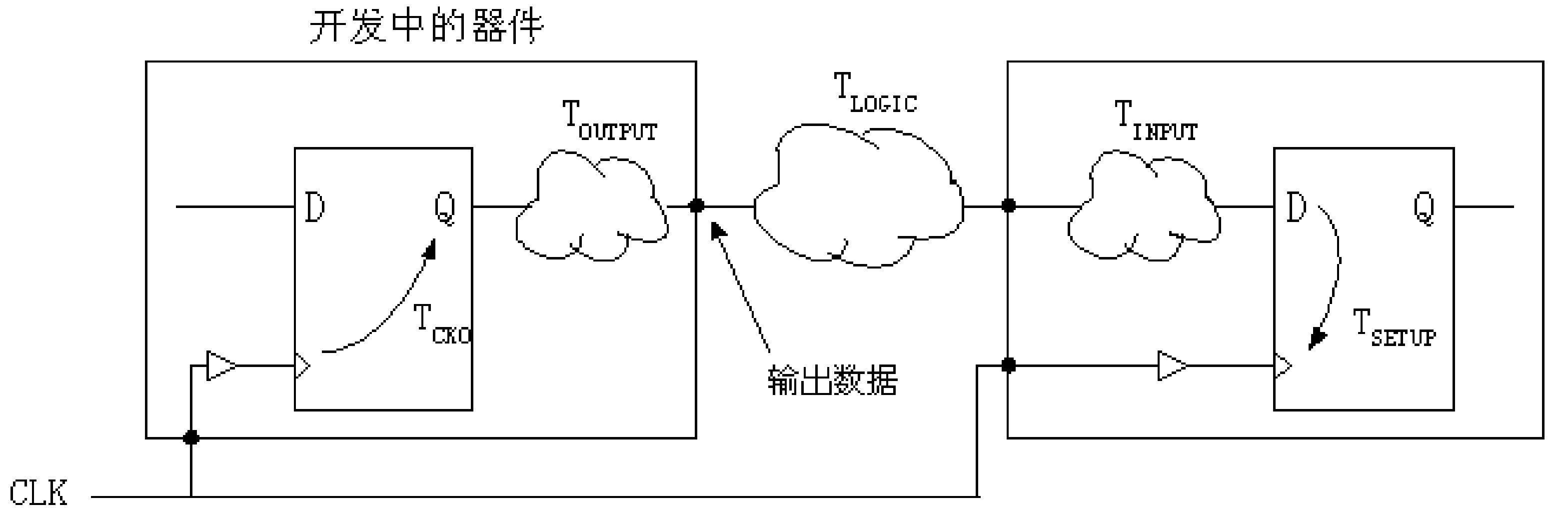
*Tcko +Toutput+Tstable<Tperiod*

NET SIGNAL_OUT OFFSET=OUT Tstable BEFORE CLK

NET SIGNAL_OUT OFFSET=OUT Tdelay AFTER CLK

其中Tdelay<Tperiod-Tstable

# 偏移约束OFFSET_OUT实例




- 例子：
  设时钟周期为20ns，后级输入逻辑延时*Tinput*为4ns、建立时间*Tsetup*为1ns，中间逻辑*Tlogic*的延时为8ns，请给出设计的输出偏移约束。
- 答案：
  - OFFSET_OUT_BEFORE 偏移约束为：
    *NET DATA_OUT OFFSET=OUT 13ns BEFORE CLK*
  - OFFSET_OUT_AFTER约束：
    *NET DATA_OUT FFSET=OUT 7ns AFTER CLK*

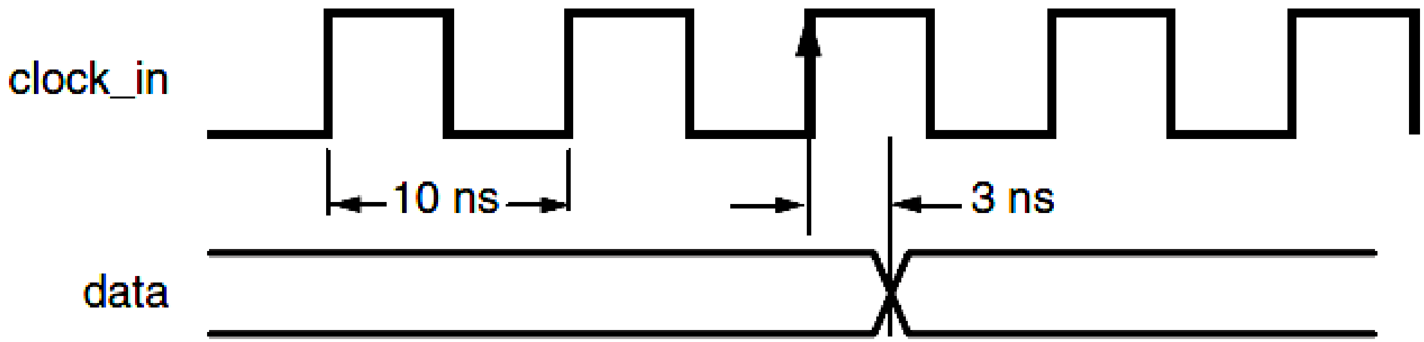
# OFFSET Constraints Reporting

- Timing constraint reporting is handled by the Timing Analyzer
- The Offset In/Out constraints take into account the clock delay and jitter

```
Timing constraint: OFFSET = OUT 15 ns AFTER COMP "clk_pin" "RISING";
 8 paths analyzed, 8 endpoints analyzed, 0 failing endpoints
 0 timing errors detected.
 Minimum allowable offset is  8.449ns.
--------------------------------------------------------------------------------
Slack (slowest paths):  6.551ns (requirement - (clock arrival + clock path + data path + uncertainty))
  Source:               led_ctl_i0/led_o_1 (FF)
  Destination:          led_pins<1> (PAD)
  Source Clock:         clk_rx rising at 0.000ns
  Requirement:          15.000ns
  Data Path Delay:      3.816ns (Levels of Logic = 1)
  Clock Path Delay:     4.633ns (Levels of Logic = 2)
  Clock Uncertainty:    0.000ns
```

# OFFSET_OUT_AFTER分析

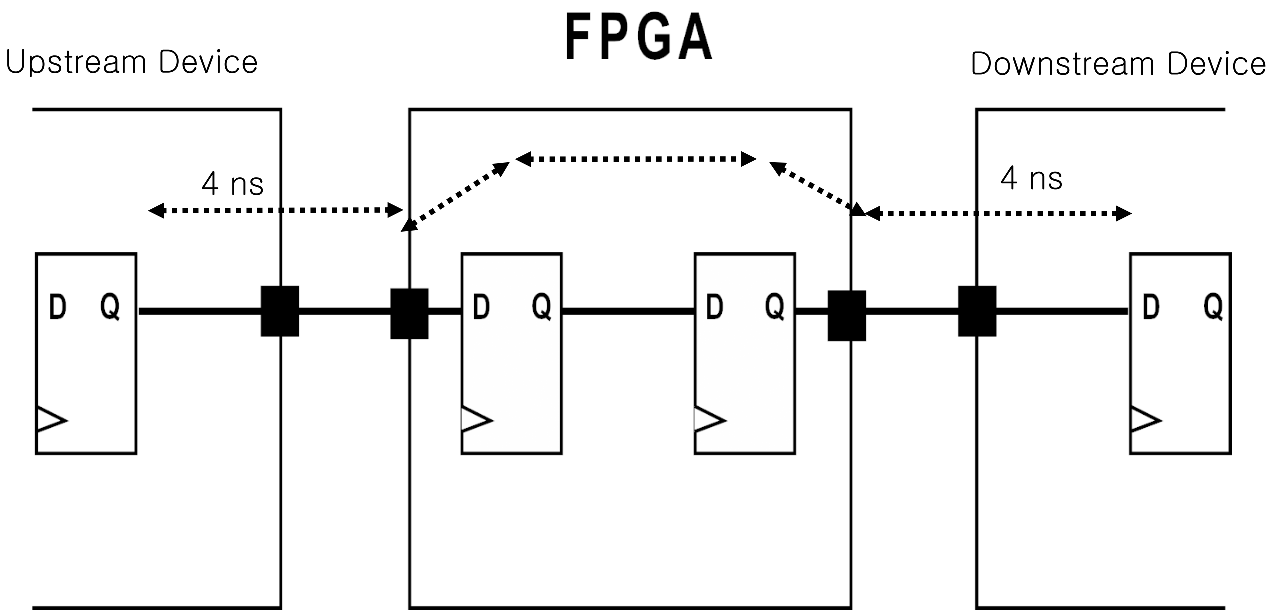
- TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
- OFFSET = OUT 3 ns AFTER clock;




```
Slack:                  -0.865ns (requirement - (clock arrival + clock
path + data path + uncertainty))
  Source:               OutD_7 (FF)
  Destination:          OutD<7> (PAD)
  Source Clock:         clock3_std_bufg rising at 0.000ns
  Requirement:          3.000ns
  Data Path Delay:      3.405ns (Levels of Logic = 1)
  Clock Path Delay:     0.280ns (Levels of Logic = 3)
  Clock Uncertainty:    0.180ns
…
```

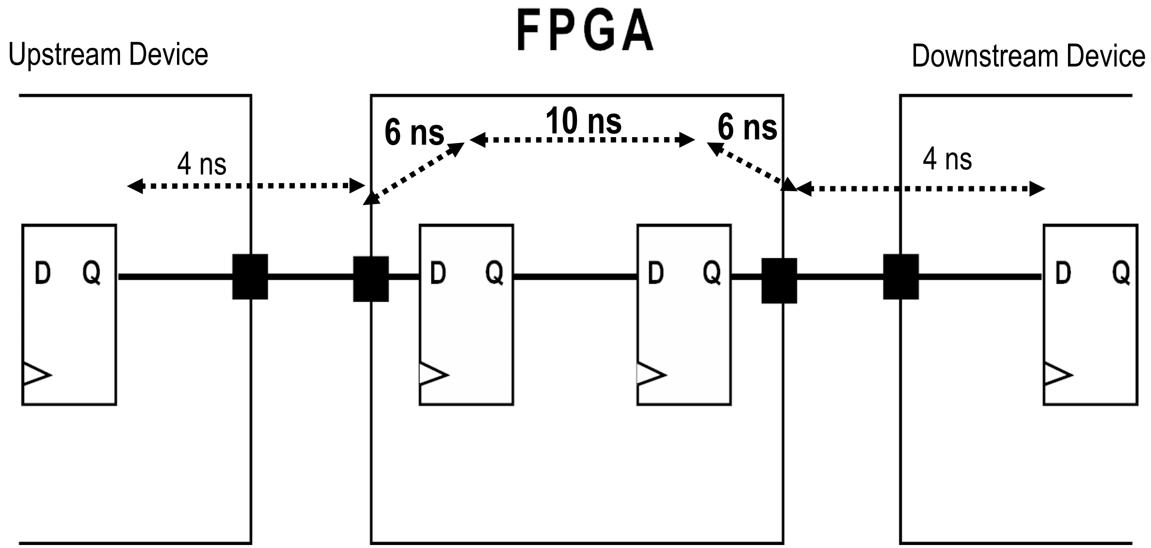
# 偏移约束—思考

- Given the system diagram below, what values would you put in the Constraints Editor so that the system will run at **100 MHz**?（Assume no clock skew between devices）



# Answer

Given the system diagram below, what values would you put in the Constraints Editor so that the system will run at 100 MHz?



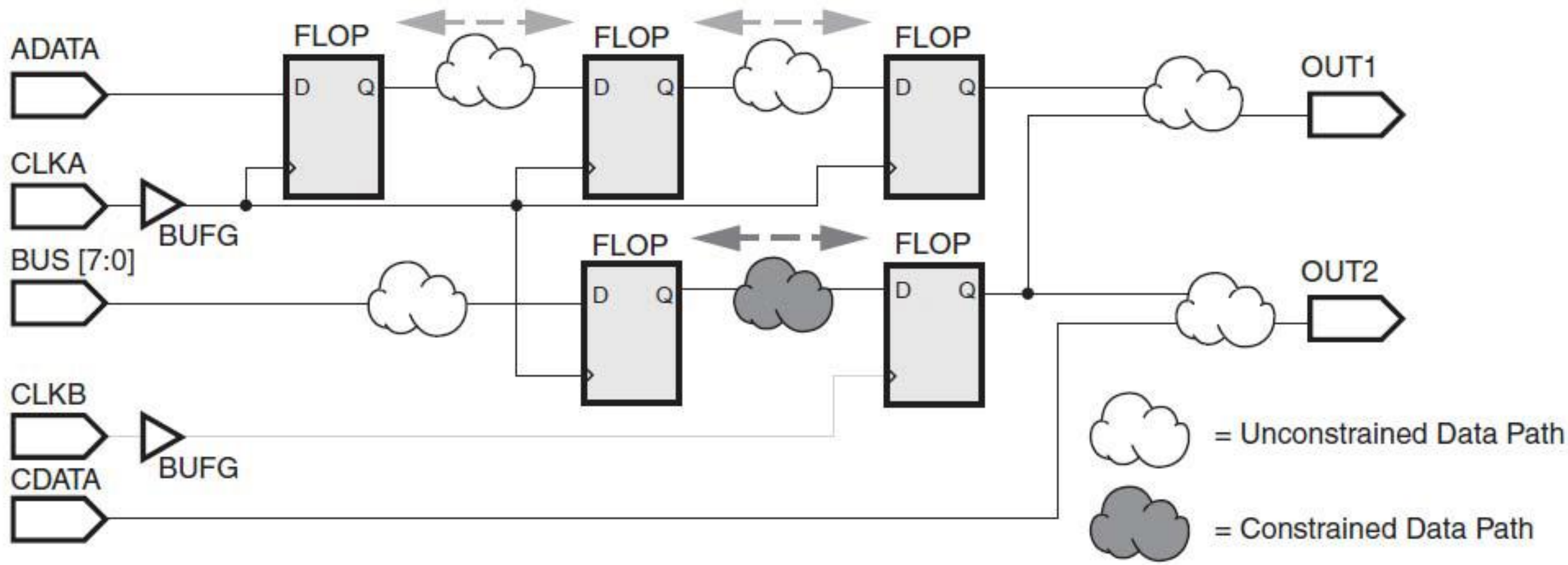

- Answer: PERIOD = 10 ns , OFFSET IN BEFORE = 6 ns, and OFFSET OUT AFTER= 6 ns

# 专用约束

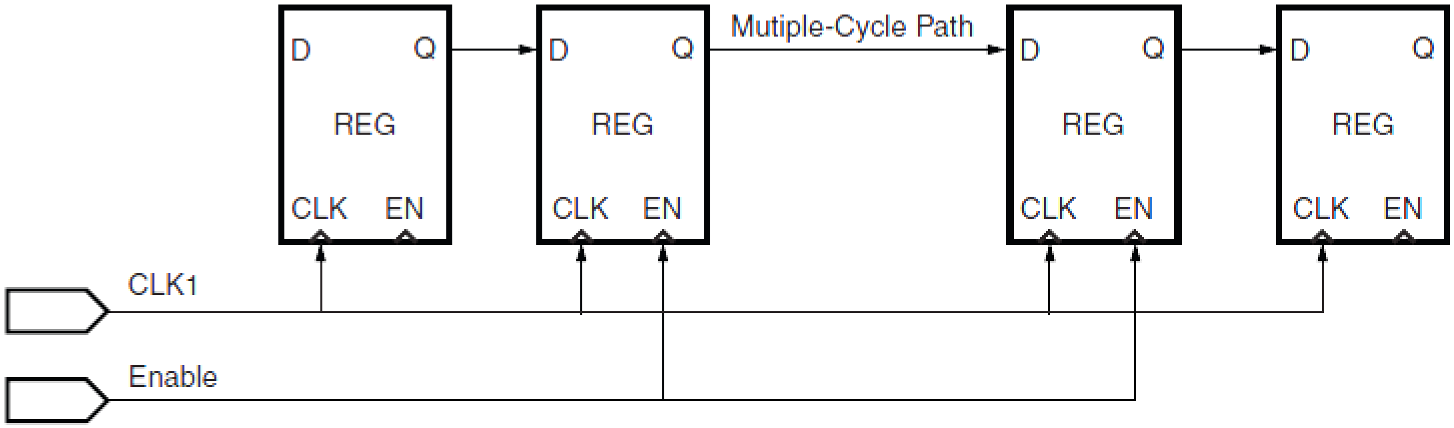
- 多周期路径
- 不同时钟域之间的路径

# 多周期路径约束

- 多周期（Multi-cycle）路径指路径延迟允许跨越多个时钟周期，多周期约束通常使用FROM:TO约束语句完成。
- FROM:TO约束也能覆盖不同时钟域之间间多周期路径。



# 多周期路径约束

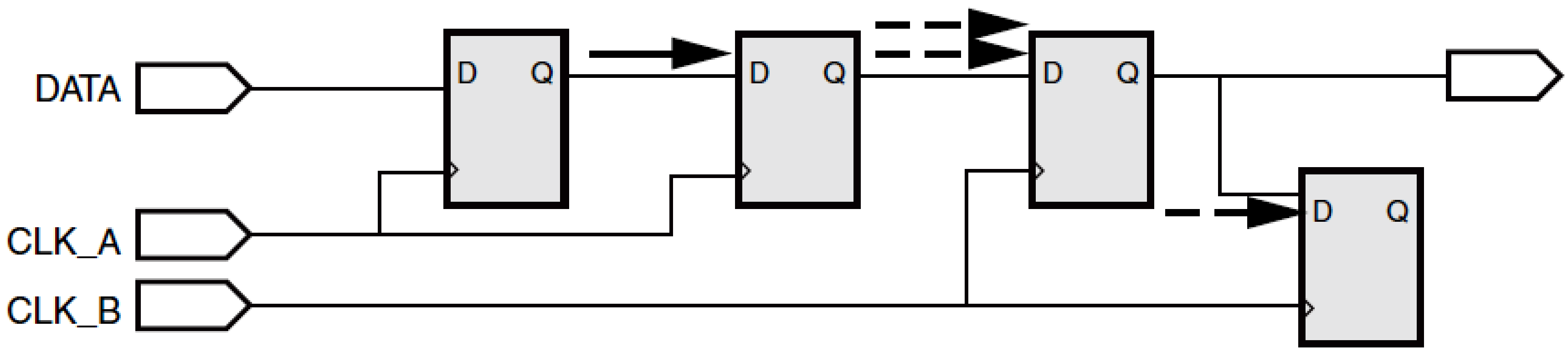



- 在两个时序分组之间定义多周期路径约束的语法格式为：
- TIMESPEC "TSid" = FROM "MC_GRP" TO "MC_GRP" <value>;
- 图中，“MC_GRP”时序分组是一个被“Enable”时钟使能信号驱动的寄存器组，时钟使能信号以参考时钟的一半频率变化
- 约束如下：

```
NET "CLK1" TNM_NET = "CLK1";
TIMESPEC "TS_CLK1" = PERIOD "CLK1" 5 ns HIGH 50%;
NET "Enable" TNM_NET = FFS "MC_GRP";
TIMESPEC TS_Example = FROM "MC_GRP" TO "MC_GRP" TS_CLK1*2;
```

# FROM:TO约束

- 时序分析报告包含不相关时钟域之间的跨时钟域路径。
- 如果需要对这些路径进行时序分析，就必须为这两个时钟域建立相关关系，然后使用多周期路径约束或者FROM:TO约束定义时序要求。

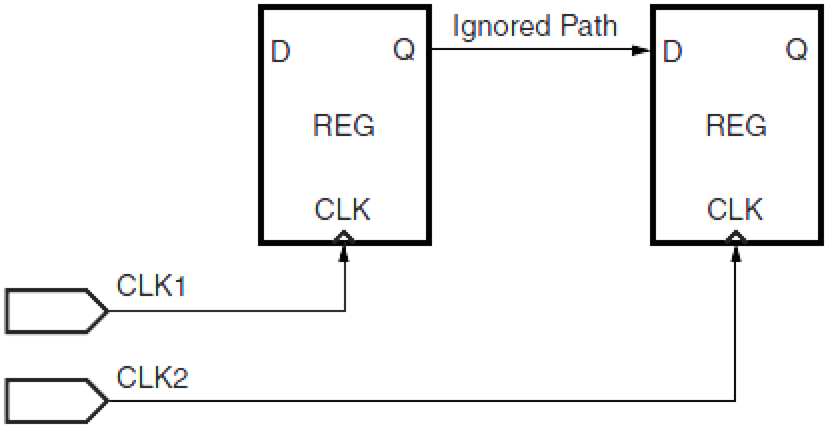



如果两个时钟在时序定义中确实是不相关的，但是在两个时钟域之间又存在数据路径需要约束，则可以建立 FROM:TO约束定义其时间延迟。

TIMESPEC TS_clkA_to_clkB = FROM CLK_A TO CLK_B 8 ns;

# 虚假路径约束

- 设计中不会影响时序性能的路径， 可以从时序约束中排除。
- 减少实现流程时间， 甚至提高时序性能的目的。

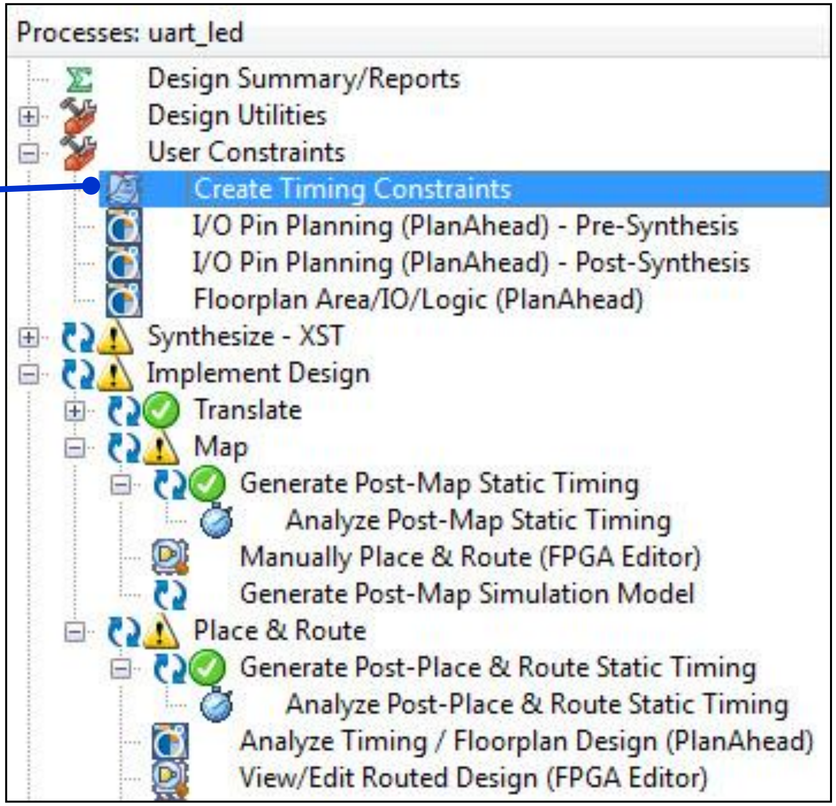



- 假路径约束步骤：指定一组寄存器为源时序分组；指定另外一组寄存器为目标时序分组；
- 使用带有“TIG”关键字的FROM-TO约束移除两个时序分组之间的路径。在时序分析过程中，时序分析器会自动移除这些路径。
- 在时序分组之间定义时序忽略（TIG）的语法是：TIMESPEC "TSid" = FROM "SRC_GRP" TO "DST_GRP" TIG;

```
NET "CLK1" TNM_NET = FFS "GRP_1";
NET "CLK2" TNM_NET = FFS "GRP_2";
TIMESPEC TS_Example = FROM "GRP_1" TO "GRP_2" TIG;
```

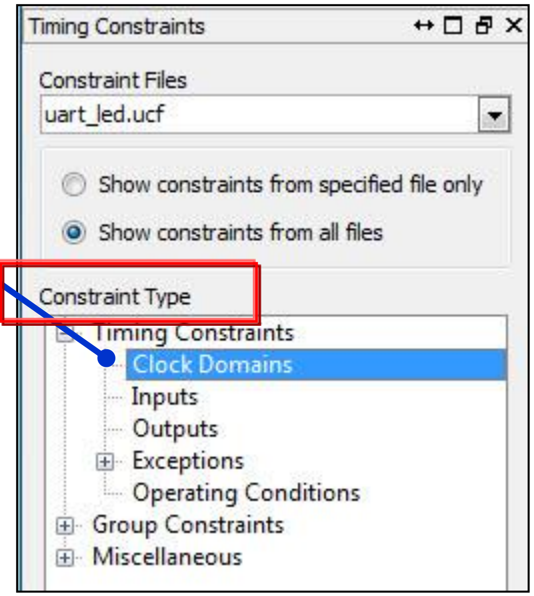
# 启动Constraints Editor

- Expand **User Constraints** in t he Processes window
- Double-click **Create Timing Constraints**



# 进行周期约束

- PERIOD constraints can be entered by clicking **Clock Domains**
- Constraints can be deleted by right-clicking the constraint
- Right-click here and select **Create Constraint** to make a PERIOD constraint

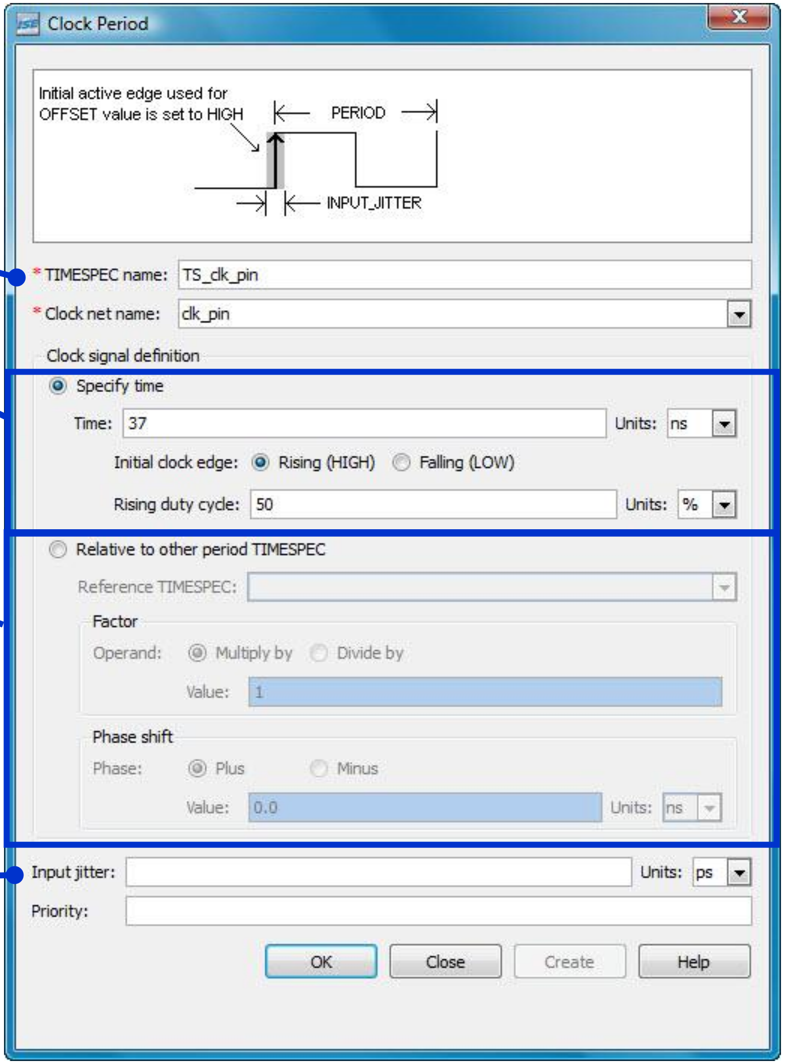



Create Timing Constraints for Clock Domains (PERIOD)

| | TIMESPEC Name * | Clock Time Name | Clock Net * | Reference TIMESPEC | Period | Duty Cycle | Factor | Edge | Phase Shift | Input Jitter |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | TS_clk_pin | clk_pin | clk_pin | | 37 ns | 50 % | | HIGH | | |
| 2 | | | | | | | | | | |

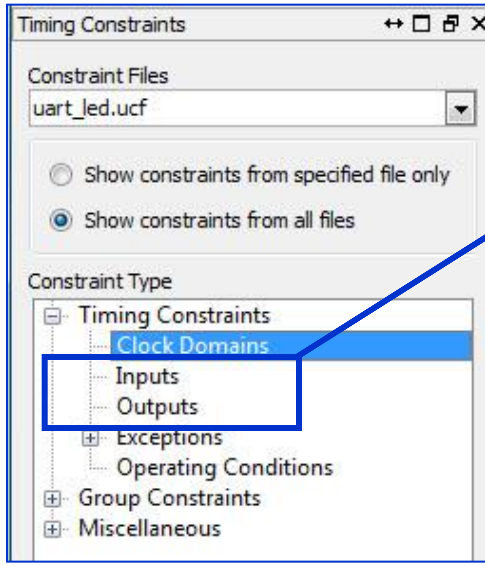
# 周期约束选项

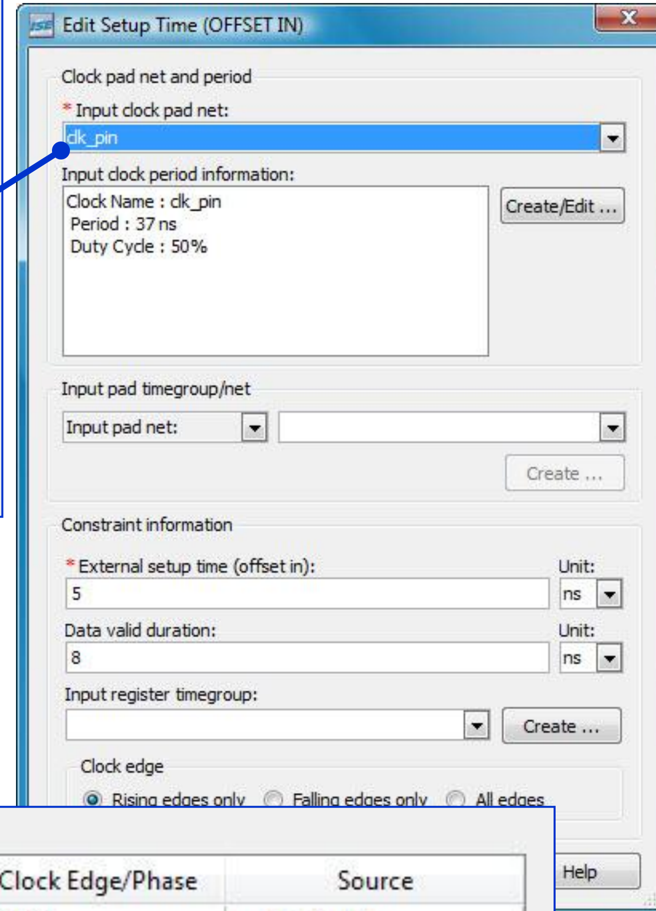
- TIMESPEC name
- Specific constraint value
  - Active clock edge
  - Duty cycle
- Relative to other PERIOD TIMESPEC
  - Useful for designs with multiple clock signals
  - Can define both frequency and phase relationships
- Input jitter

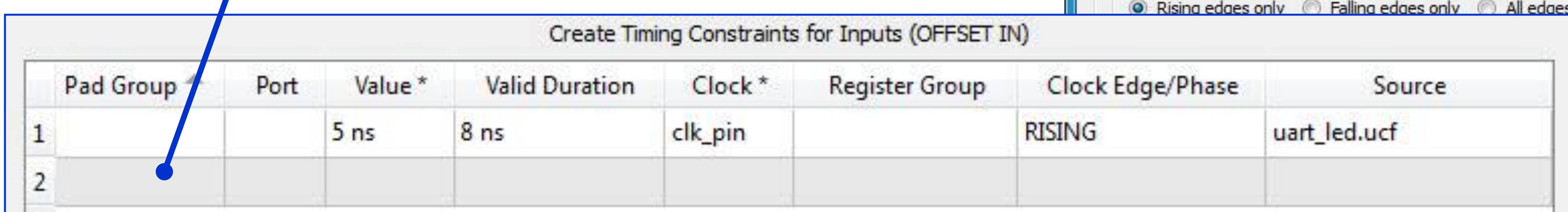


# 进行OFFSET约束

- Global OFFSET IN and OFFSET OUT constraints can be made from Inputs or Outputs
- Right-click here and select **Create Constraint** to make an OFFSET constraint





Create Timing Constraints for Inputs (OFFSET IN)

| | Pad Group * | Port | Value * | Valid Duration | Clock * | Register Group | Clock Edge/Phase | Source |
|---|---|---|---|---|---|---|---|---|
| 1 | | | 5 ns | 8 ns | clk_pin | | RISING | uart_led.ucf |
| 2 | | | | | | | | |

# UCF文件示例

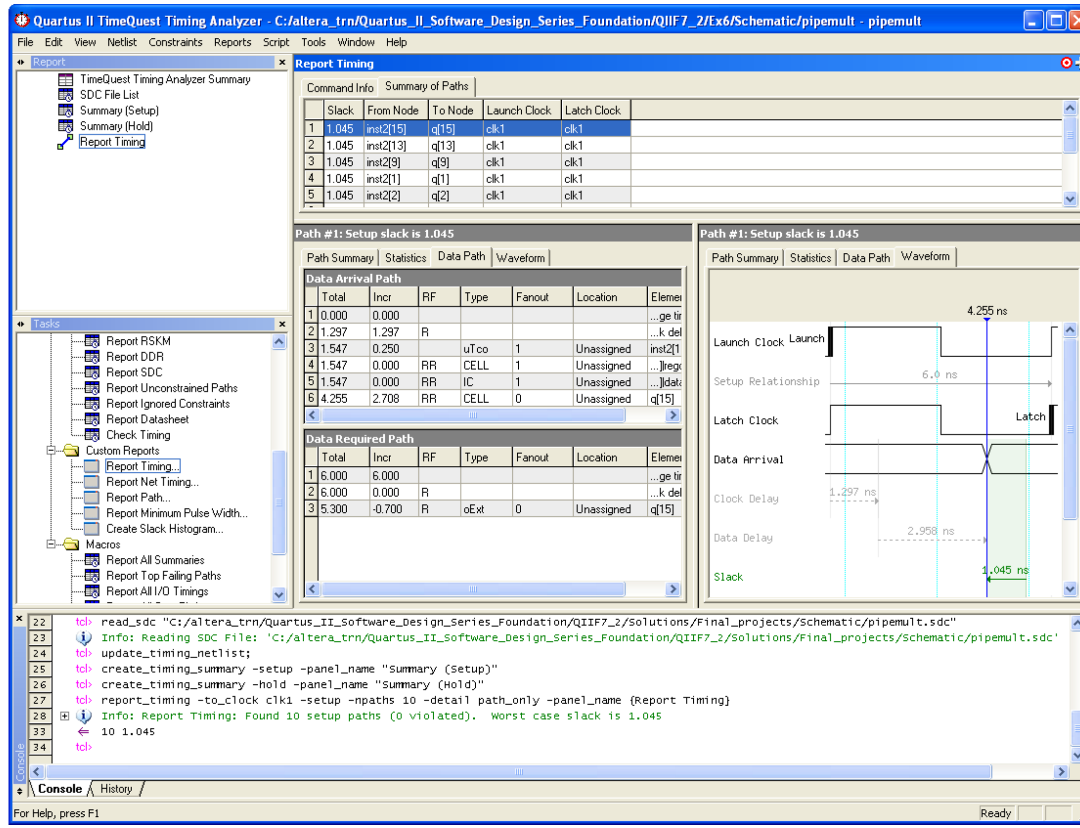
```
# Define CLK
NET "CLKP_I" TNM_NET = TNM_Clk;
TIMESPEC "TS_Clk" = PERIOD "TNM_Clk" 10.0 ns PHASE 0.000 ns HIGH 50%;

#OFFSETS
NET "DAT*_I" OFFSET = IN 1 ns VALID 2.7 ns BEFORE "CLKP_I" RISING;
NET "DAT*_I" OFFSET = IN 1ns VALID 2.7 ns BEFORE "CLKP_I" FALLING;

NET "DAT*_O" OFFSET = OUT 15.0 ns AFTER "CLKP_I" RISING;
NET "DAT*_O" OFFSET = OUT 15.0 ns AFTER "CLKP_I" FALLING;

NET "DATA1_I" TIG;
NET "DATA2_O" TIG;
```

# 课程安排

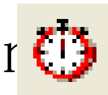
- 时序约束的目的
- 时序约束的内容
- Xilinx FPGA时序约束方法
- **Altera FPGA时序约束方法**
- 时序约束的原则

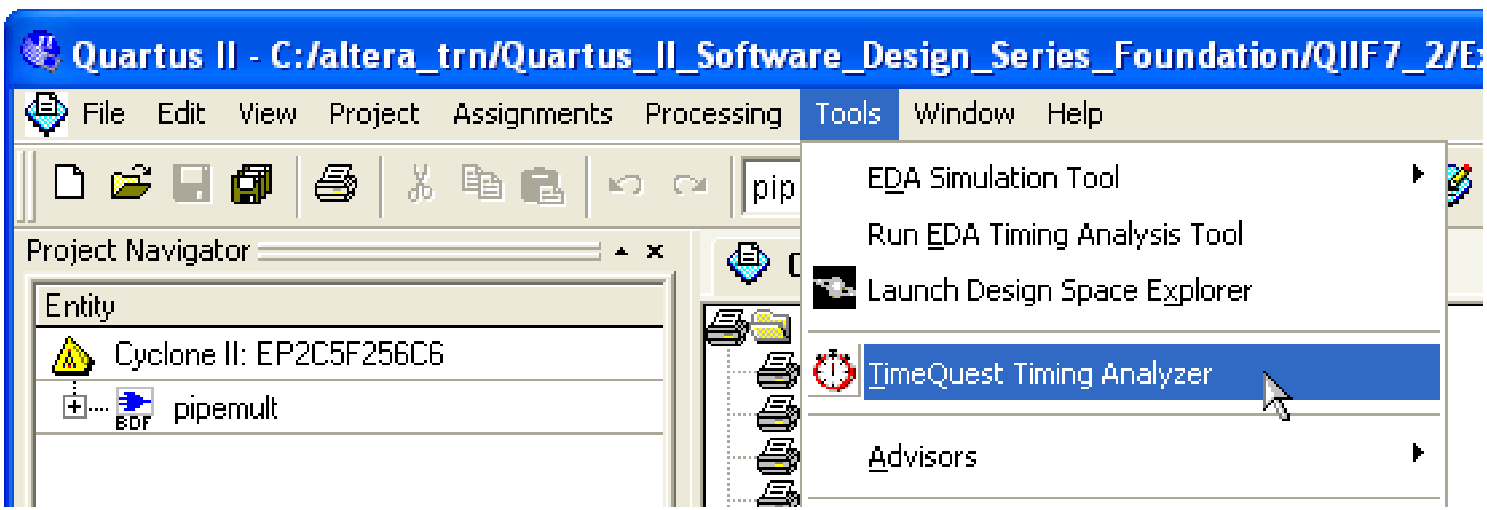
# TimeQuest Timing Analyzer

- Timing engine in Quartus II software
- Provides timing analysis solution for all levels of experience
- Features
  - Synopsys Design Constraints (SDC) support
    - Standardized constraint methodology
  - Easy-to-use interface
    - Constraint entry
    - Standard reporting
  - Scripting emphasis
    - Presentation focuses on using GUI



# Opening the TimeQuest Interface

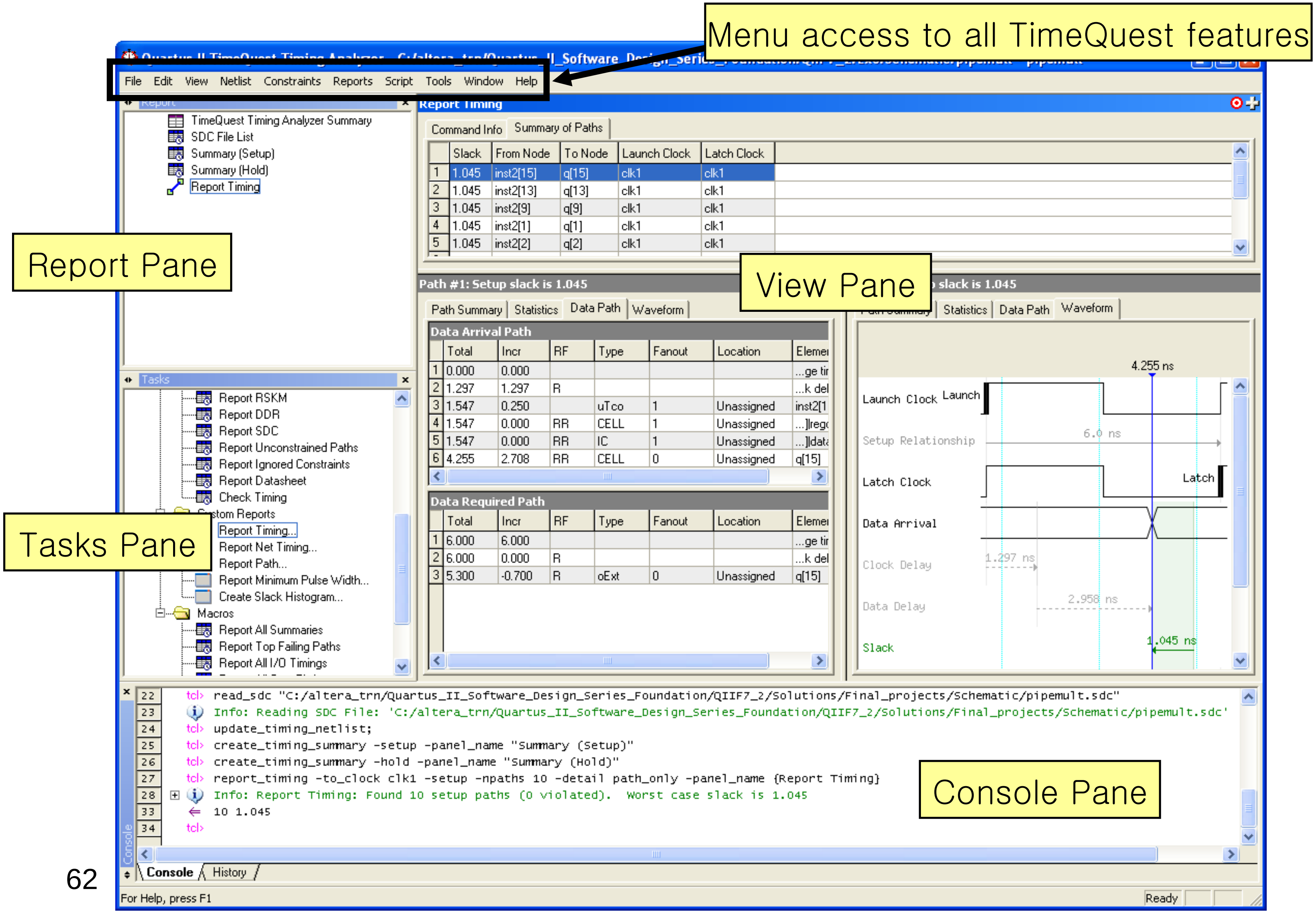
- Toolbar button 
- **Tools** menu
- Tasks window
- Stand-alone mode
  - **quartus_staw**
- Command line






61

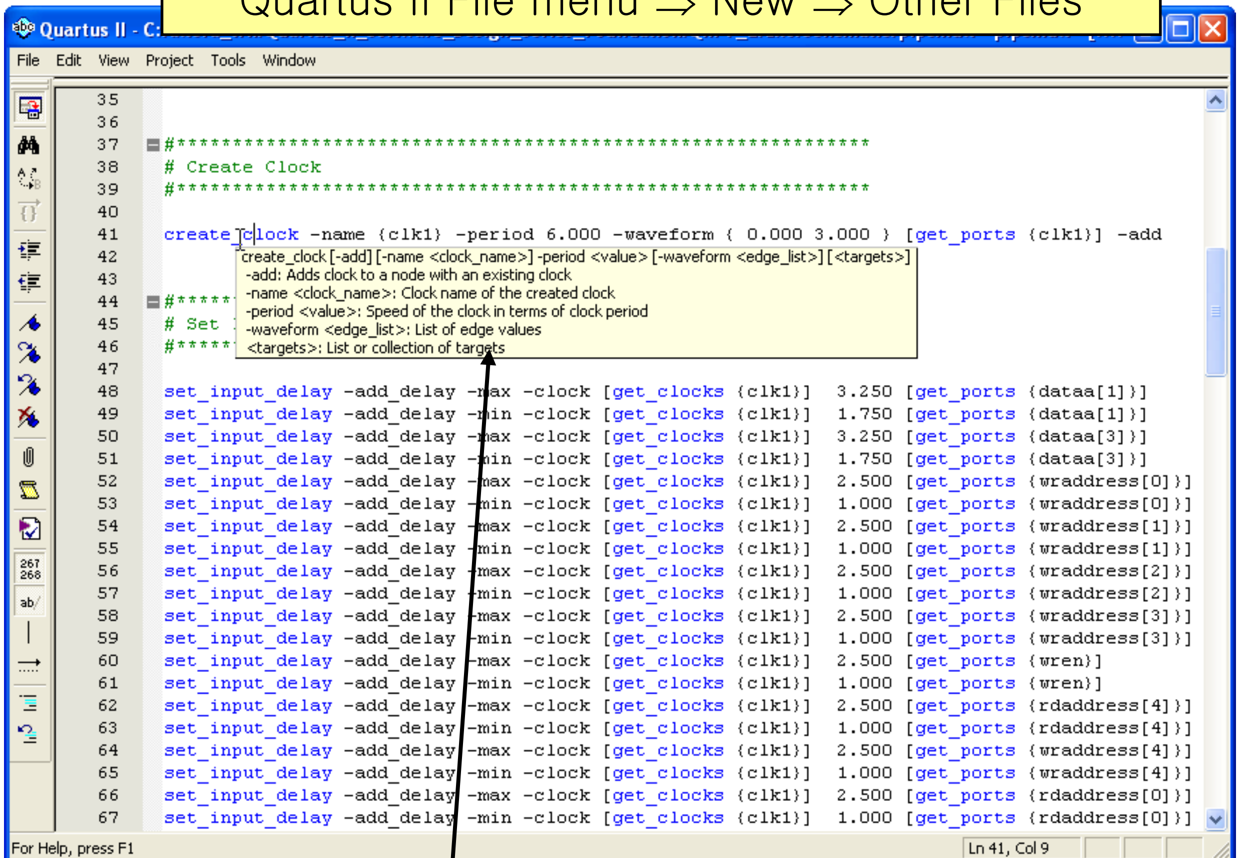
# TimeQuest GUI



# SDC File Editor = Quartus II Text Editor

- Use Quartus II editor to create and/or edit SDC
- SDC editing unique features (for .sdc files)
  - Access to GUI dialog boxes for constraint entry (**Edit ⇒ Insert Constraint**)
  - Syntax coloring
  - Tooltip syntax help

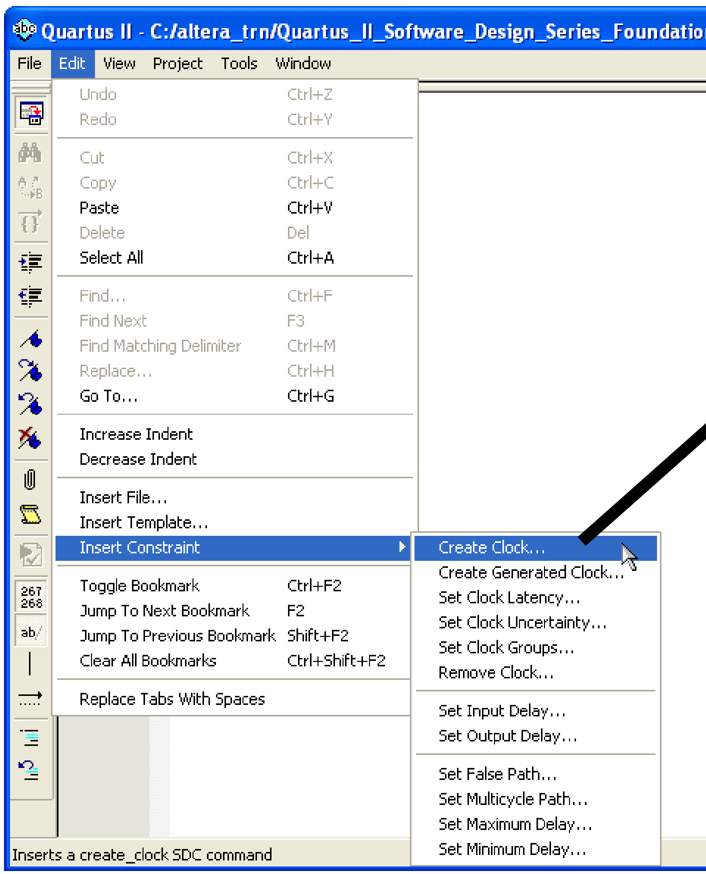
TimeQuest File menu ⇒ New/Open SDC File
Quartus II File menu ⇒ New ⇒ Other Files

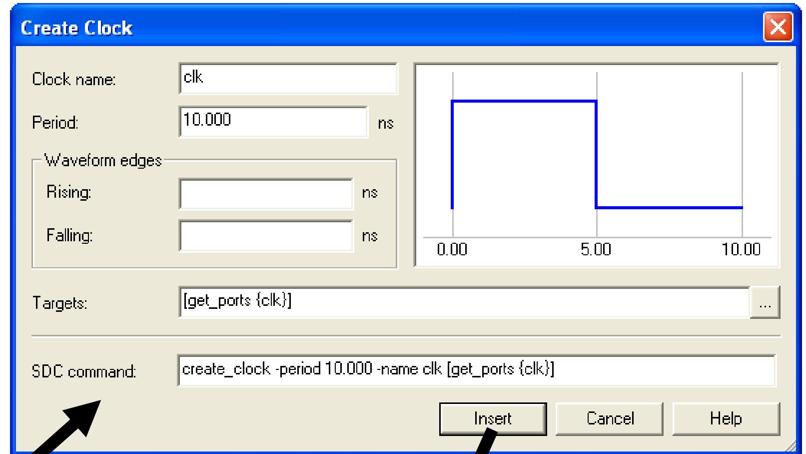


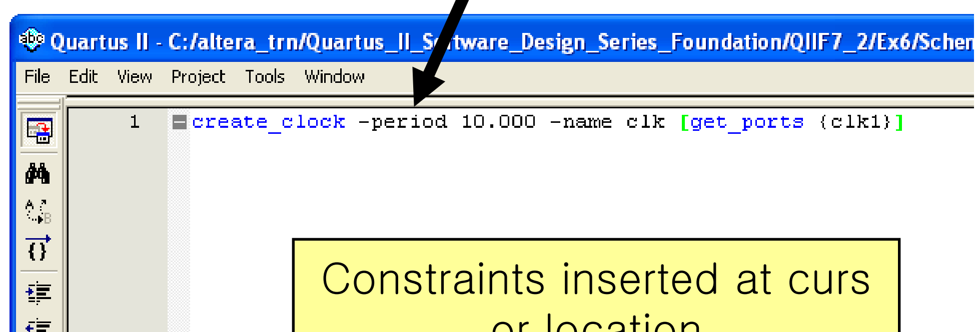

Place cursor over command to see tooltip

# SDC File Editor (cont.)

Construct an SDC file using TimeQuest graphical constraint creation tools



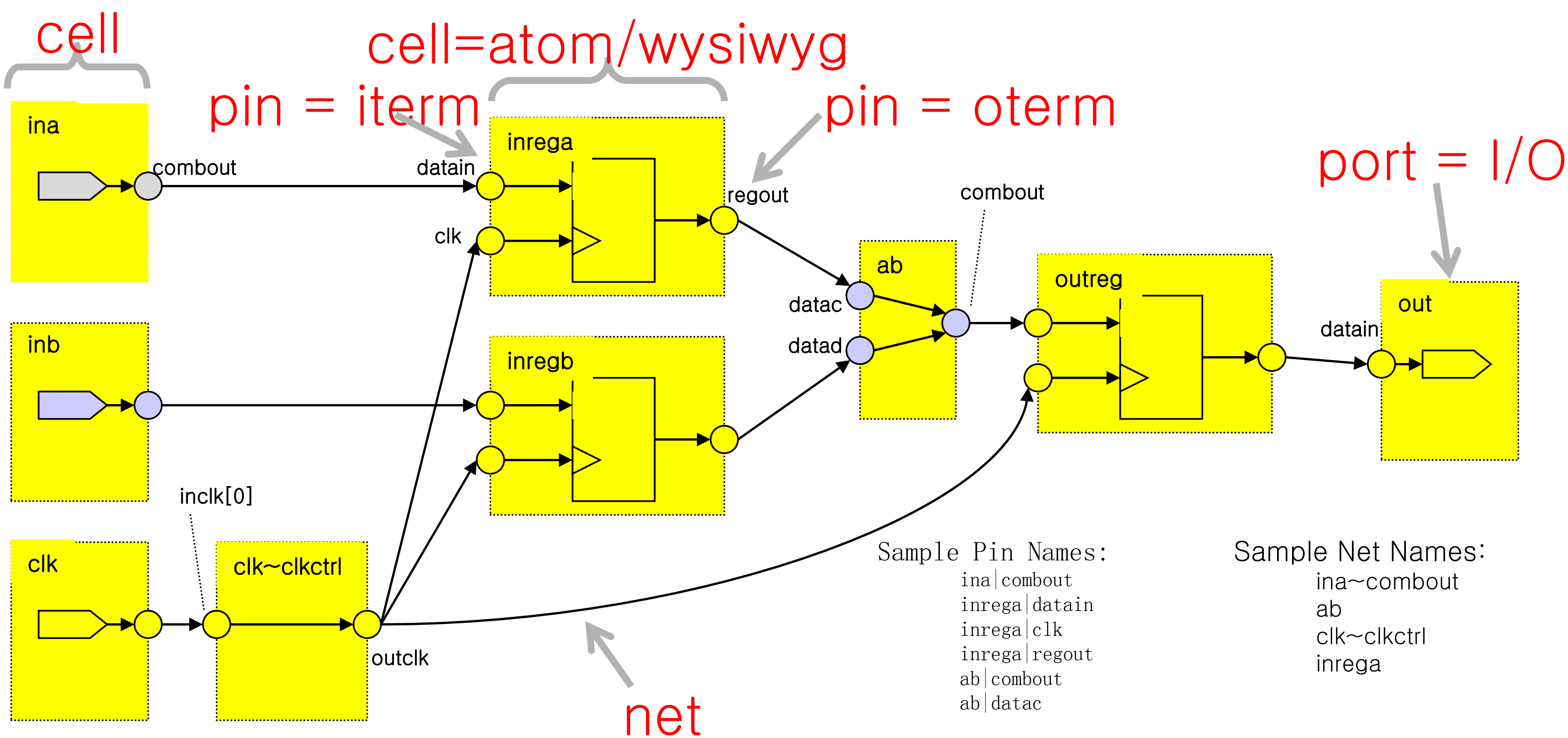




Constraints inserted at cursor location

# SDC Netlist Terminology

| Term | Definition |
|---|---|
| Cell | Device building blocks (e.g. look-up tables, registers, embedded multipliers, memory blocks, I/O elements, PLLs, etc.) |
| Pin | Input or outputs of cells |
| Net | Connections between pins |
| Port | Top-level inputs and outputs (e.g. device pins) |

# SDC Netlist Example




- Paths defined in constraints by targeted endpoints (pins or ports)

# SDC Timing Constraints

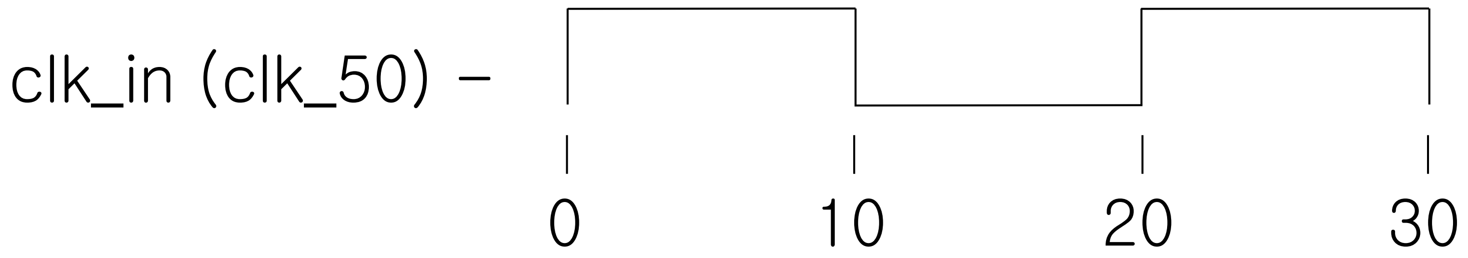
- Clocks ⟵
- I/O
- False paths
- Multicycle paths

# Clock Constraints

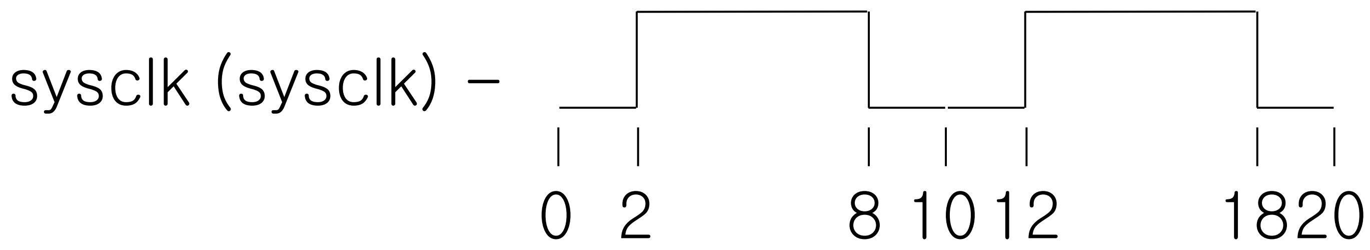
- Create clock
- Create generated clock
- PLL clocks
- Automatic clock detection & creation
- Default constraints
- Clock latency
- Clock uncertainty

# create_clock Examples

```
create_clock -period 20.0 -name clk_50 [get_ports clk_in]
```




```
create_clock -period 10.0 -waveform {2.0 8.0} [get_ports sysclk]
```

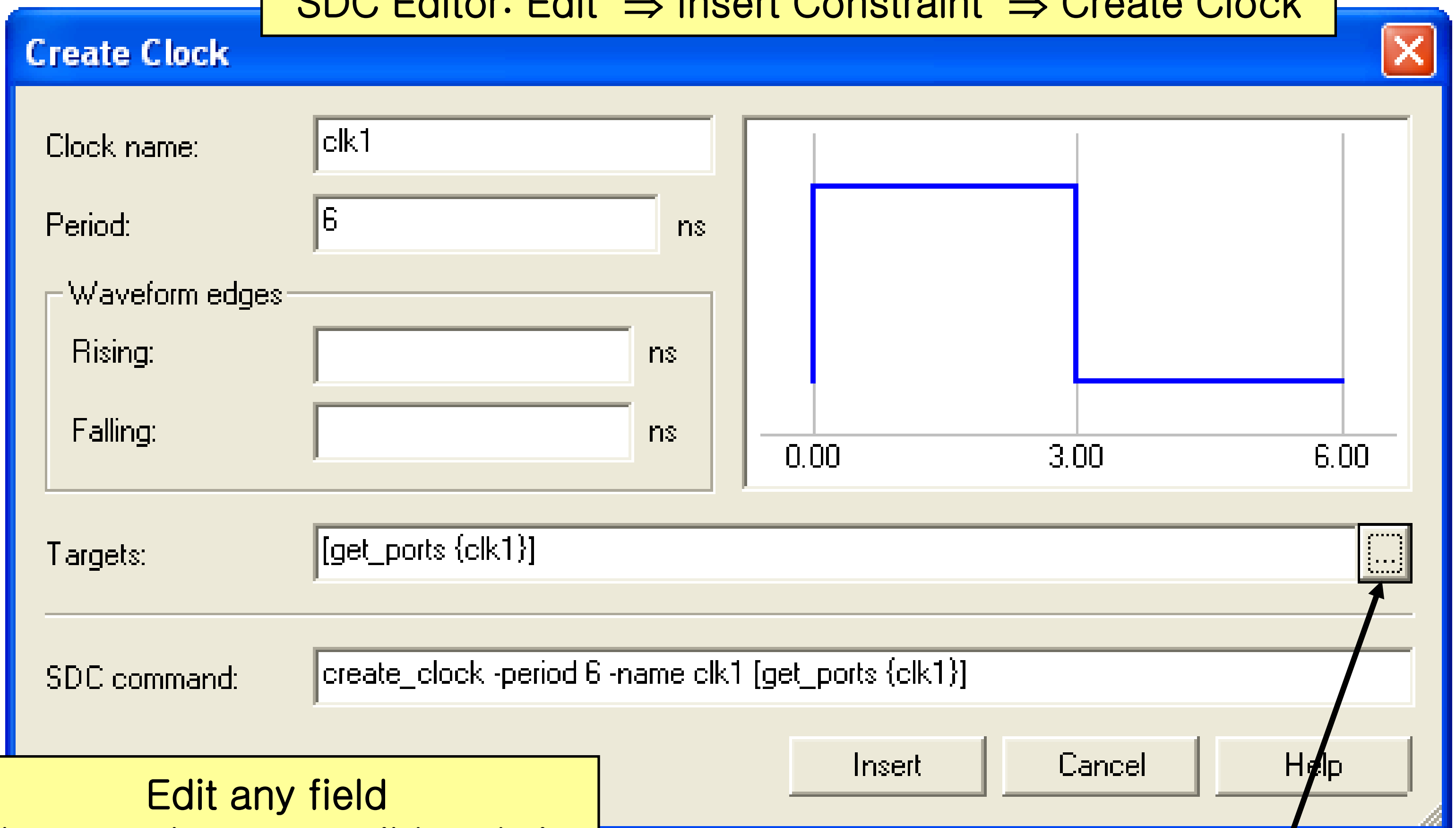





69

# Create Clock using GUI

TimeQuest main: Constraints ⇒ Create Clock
SDC Editor: Edit ⇒ Insert Constraint ⇒ Create Clock




Edit any field
(change values; use wildcards in targets or command)

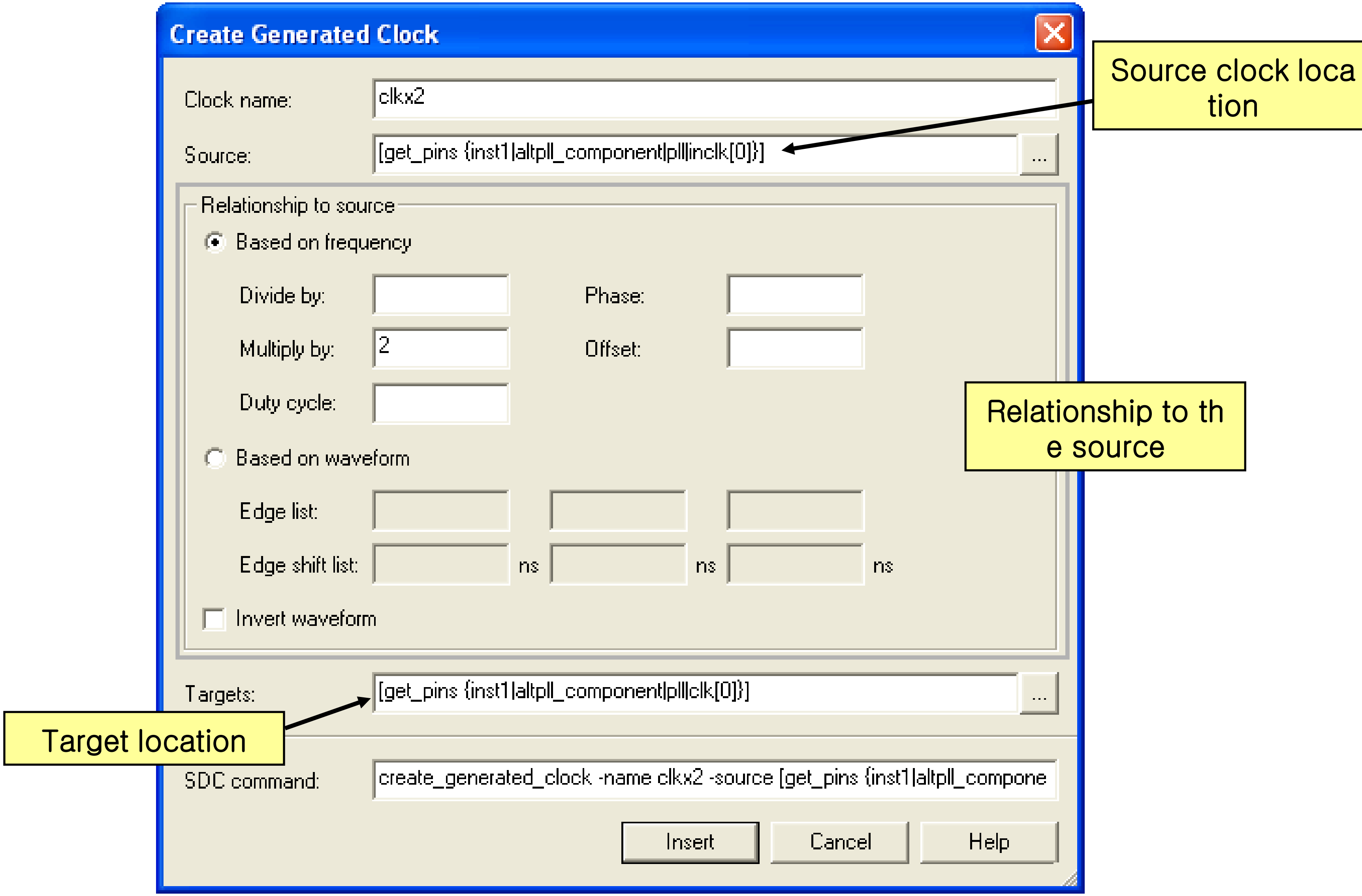
Name Finder ( next slide)

# Creating a Generated Clock

- Command:  create_generated_clock
- Options

  [-name <*clock_name*>]
  -source <*master_pin*>
  [-master_clock <*clock_name*>]
  [-divide_by <*factor*>]
  [-multiply_by <*factor*>]
  [-duty_cycle <*percent*>]
  [-invert]
  [-phase <*degrees*>]
  [-edges <*edge_list*>]
  [-edge_shift <*shift_list*>]
  [<*targets*>]
  [-add]

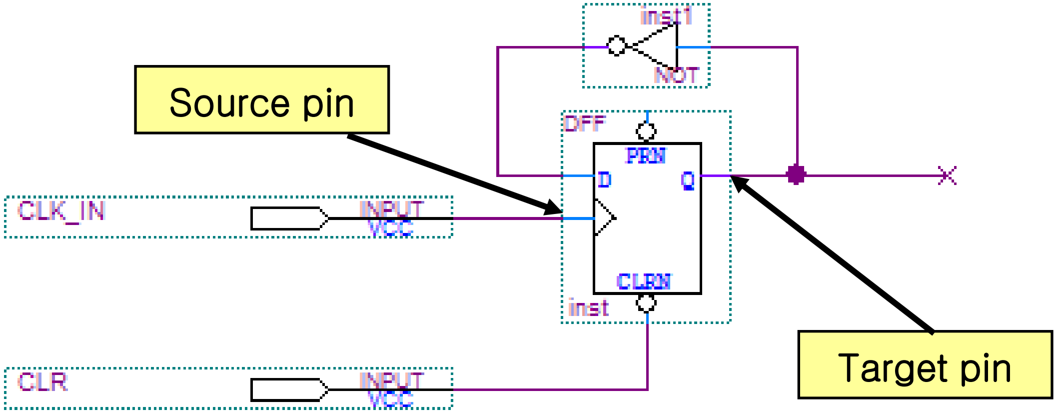
# Create Generated Clock using GUI





72

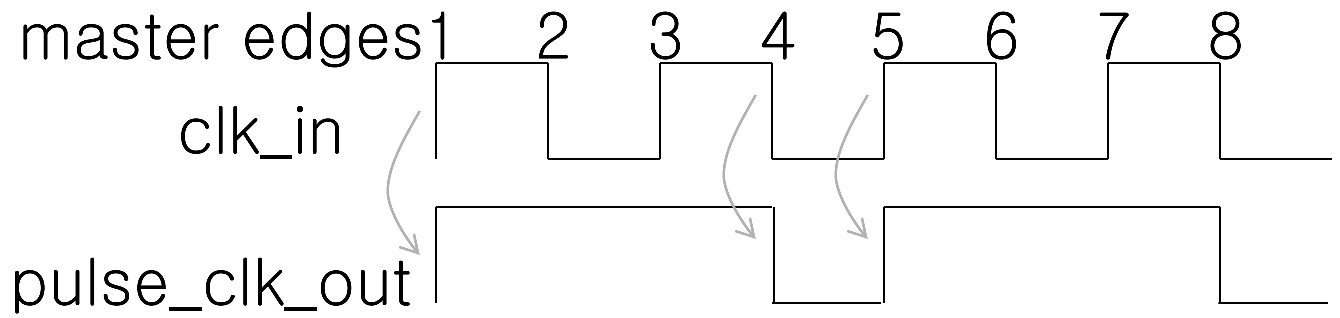
# Generated Clock Example 1




create_clock –period 10 [get_ports clk_in]

create_generated_clock –name clk_div \
–source [get_pins inst|clk] \
-divide_by 2 \
[get_pins inst|regout]

# Generated Clock Example 2




create_clock –period 10 [get_ports clk_in]

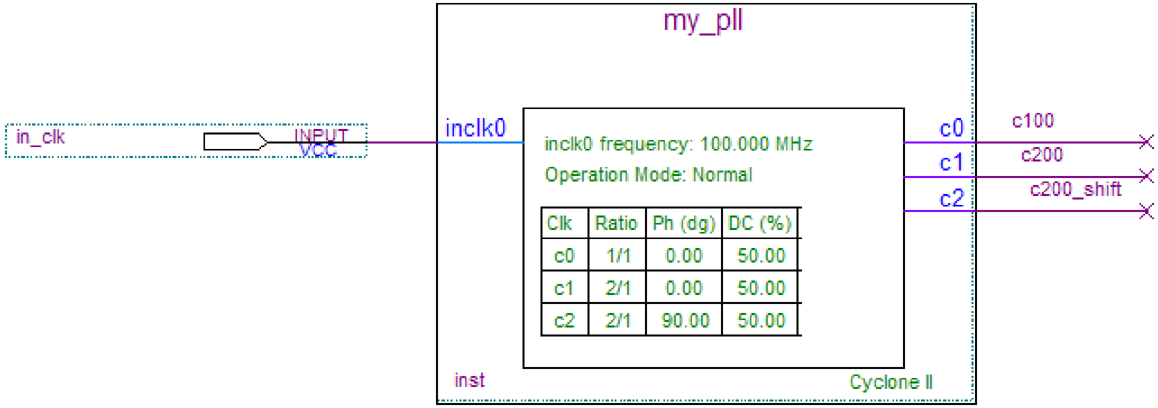
create_generated_clock –name pulse_clk_out -source clk_in \
–edges {1 4 5}
[get_pins pulse_logic|out]


74

# PLL Clocks (Altera SDC Extension)

- Command:  derive_pll_clocks
  - [-use_tan_name]: names clock after design net name from Classic timing analyzer settings instead of the default PLL output SDC pin name
  - [-create_base_clocks]: generates create_clock constraint(s) for PLL input clocks
- Create generated clocks on all PLL outputs
  - Based on input clock & PLL settings
- Automatically updates generated clocks on PLL outputs as changes made to PLL design
- *Not in GUI; must be entered in SDC manually*

# derive_pll_clocks Example



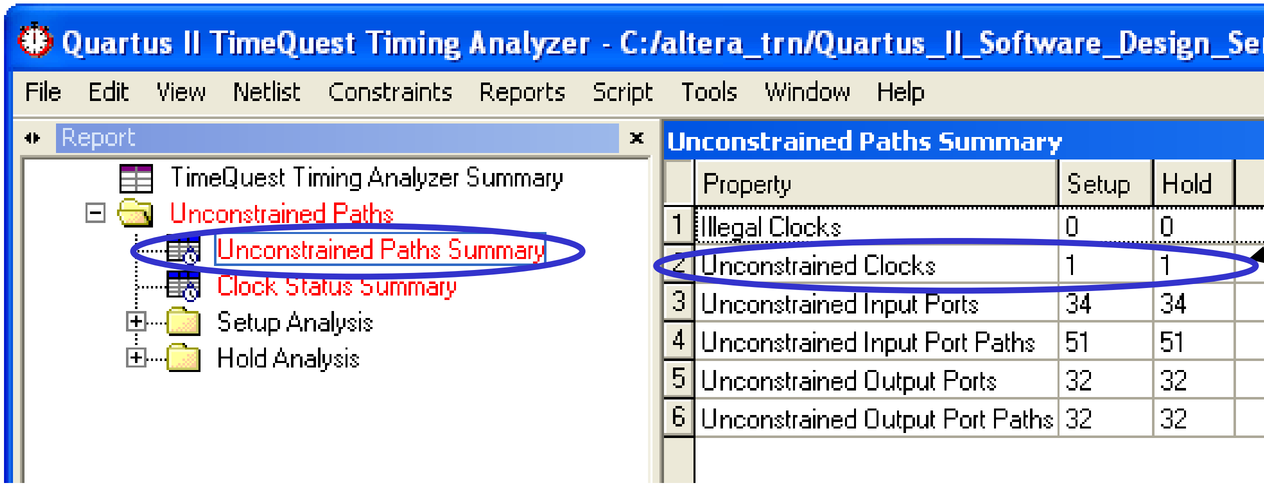

Using generated clock commands

```
create_clock -period 10.0 [get_ports in_clk]
create_generated_clock -name c100 \
    -source [get_pins {inst|altpll_component|pll|inclk[0]}] \
    -divide_by 1 \
    [get_pins {inst|altpll_component|pll|clk[0]}]
create_generated_clock -name c200 \
    -source [get_pins {inst|altpll_component|pll|inclk[0]}] \
    -multiply_by 2 \
    [get_pins {inst|altpll_component|pll|clk[1]}]
create_generated_clock -name c200_shift \
    -source [get_pins {inst|altpll_component|pll|inclk[0]}] \
    -multiply_by 2 \
    -phase 90 \
    [get_pins {inst|altpll_component|pll|clk[2]}]
```

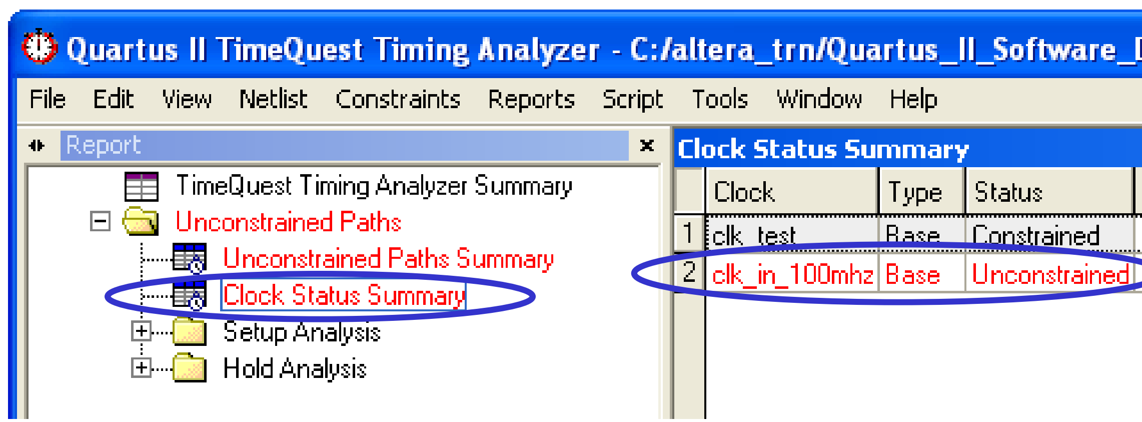
Using derive pll command

```
create_clock -period 10.0 \
    [get_ports in_clk]
derive_pll_clocks


#  Note the clock names for
#    the generated clocks
#    will be the names of
#    the PLL output pins
```

# 未约束路径报告




*指出有哪些nodes没有被约束*





77

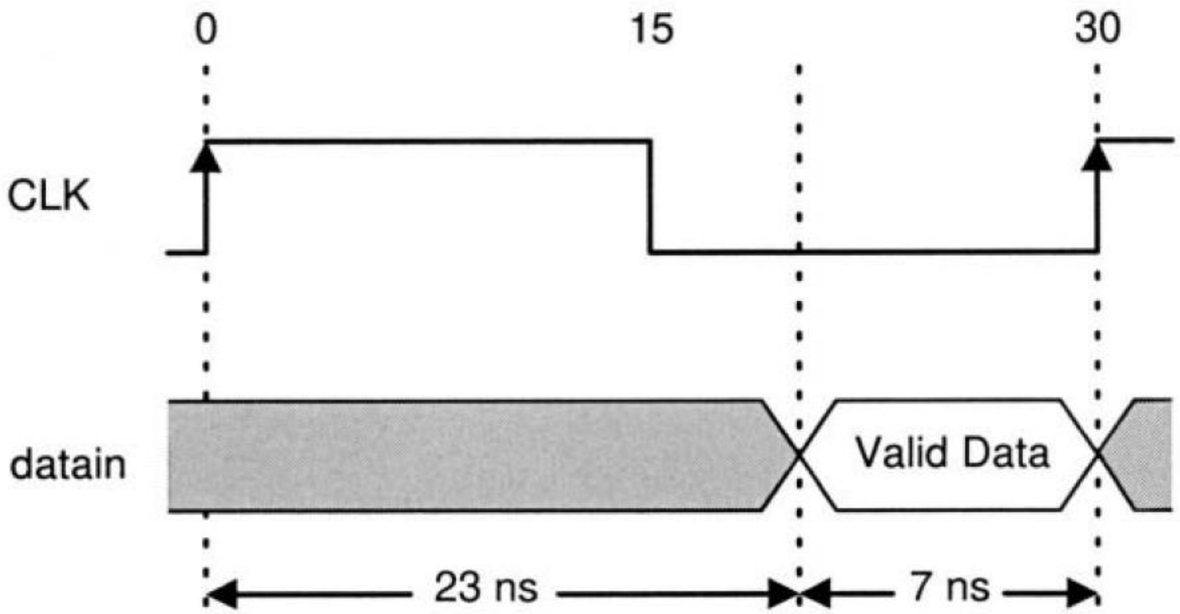
# SDC Timing Constraints

- Clocks
- I/O
- False paths
- Multicycle paths

# Input Delay

specifies the input arrival time of a signal in relation to the clock. It is used at the input ports, to specify the time it takes for the data to be stable after the clock edge.

set_input_delay –max 23.0 –clock CLK {datain}






79

# `set_input_delay` Command

- Constrains input pins by specifying *external* device timing parameters
- Options

  -clock <*clock_name*>

  [-clock_fall]

  [-rise | -fall]

  [-max | -min]

  [-add_delay]

  [-reference_pin <*target*>]

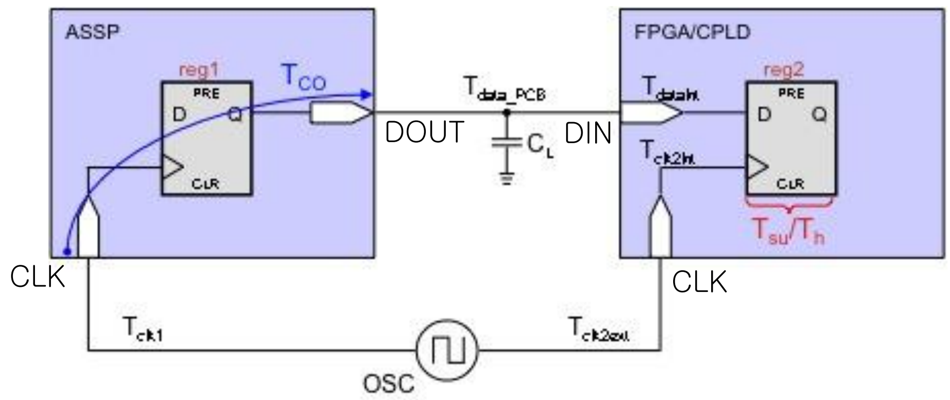
  [-source_latency_included]

  <*delay value*>

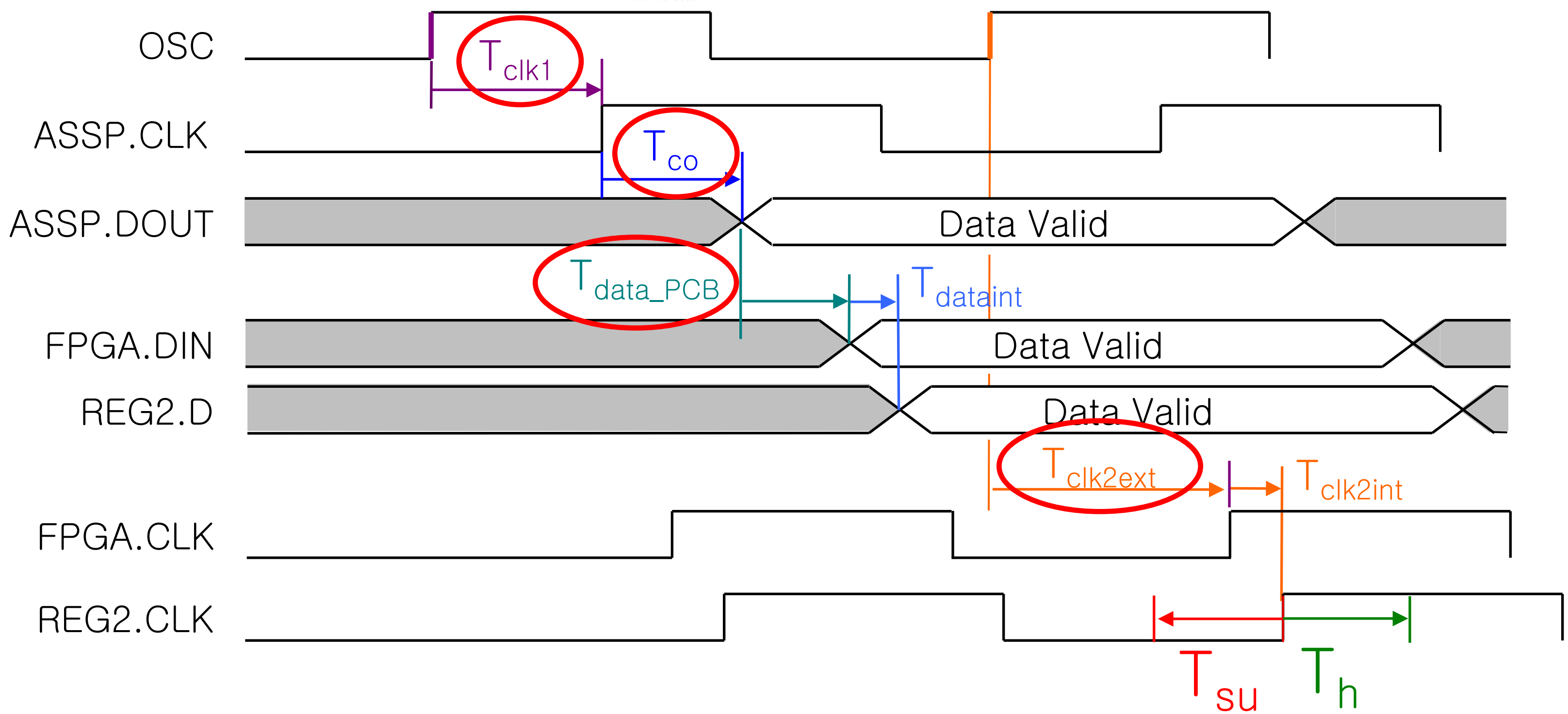
  <*targets*>


80

# 同步输入



ASSP
reg1
PRE
D
Q
CLR
$T_{co}$
DOUT
$T_{data\_PCB}$
$C_L$
DIN
FPGA/CPLD
$T_{dataint}$
reg2
$T_{clk2int}$
$T_{su}/T_h$
CLK
CLK
$T_{clk1}$
$T_{clk2ext}$
OSC

OSC
$T_{clk1}$
ASSP.CLK
$T_{co}$
ASSP.DOUT
Data Valid
$T_{data\_PCB}$
$T_{dataint}$
FPGA.DIN
Data Valid
REG2.D
Data Valid
$T_{clk2ext}$
$T_{clk2int}$
FPGA.CLK
REG2.CLK
$T_{su}$
$T_h$

# 同步输入约束

- Use set_input_delay (-max option) command to constrain input setup time (maximum time to arrive and still meet $T_{su}$)

| | |
|---|---|
| **input delay max** | **= Board Delay (max) - Board clock skew (min) + $T_{co(max)}$** |
| | $= (T_{data\_PCB(max)} + T_{CL}) - (T_{clk2ext(min)} - T_{clk1(max)}) + T_{co(max)}$ |
| data arrival time | = launch edge + input delay max + $T_{dataint}$ |
| data required time | = latch edge + $T_{clk2int}$ - $T_{su}$ |
| slack | = data required time - data arrival time |

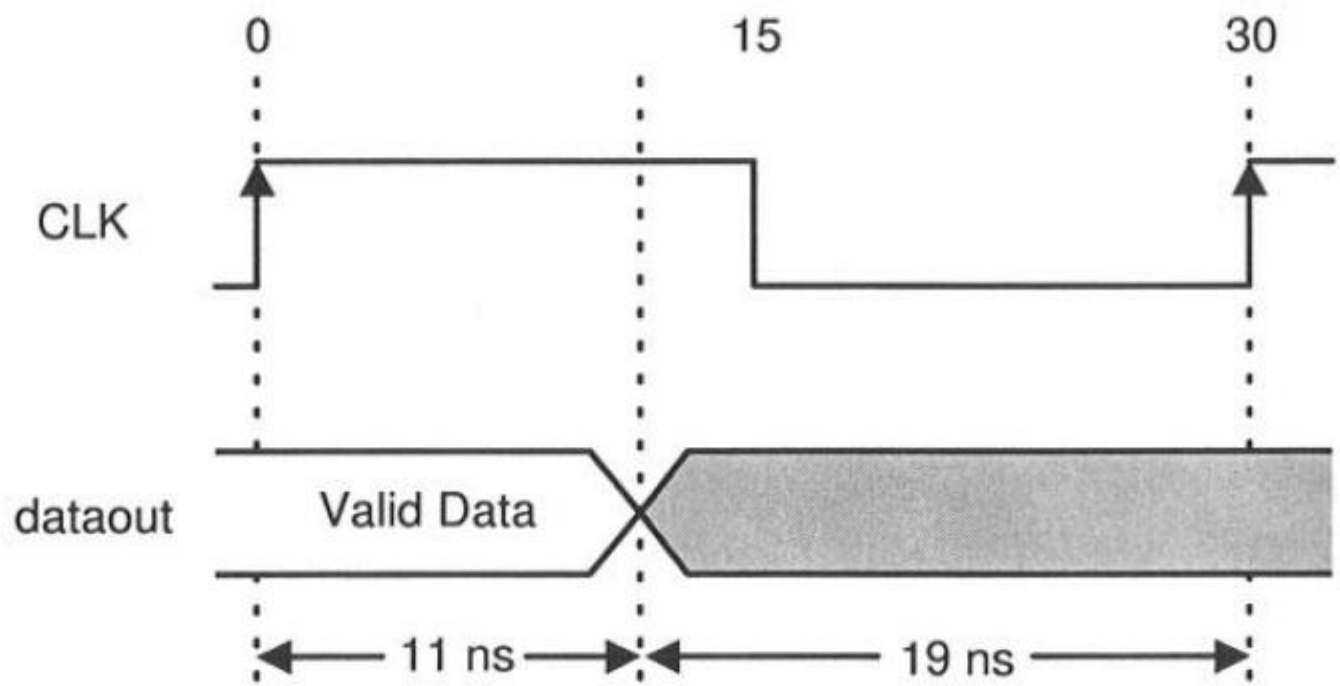
- Use set_input_delay (-min option) command to constrain input hold time (minimum time to stay active and still meet $T_h$)

| | |
|---|---|
| **input delay min** | **= Board Delay (min) - Board clock skew (max) + $T_{co(min)}$** |
| | $= (T_{data\_PCB(min)} + T_{CL}) - (T_{clk2ext(max)} - T_{clk1(min)}) + T_{co(min)}$ |
| data arrival time | = launch edge + input delay min + $T_{dataint}$ |
| required time | = latch edge + $T_{clk2int}$ + $T_h$ |
| slack | = data arrival time - data required time |

# Output Delay

**set_output_delay** command is used at the output port, to define the time it takes for the data to be available before the clock edge.

set_output_delay –max 19.0 –clock CLK {dataout}
This means that the data is valid for 11ns after the clock edge the clock edge.

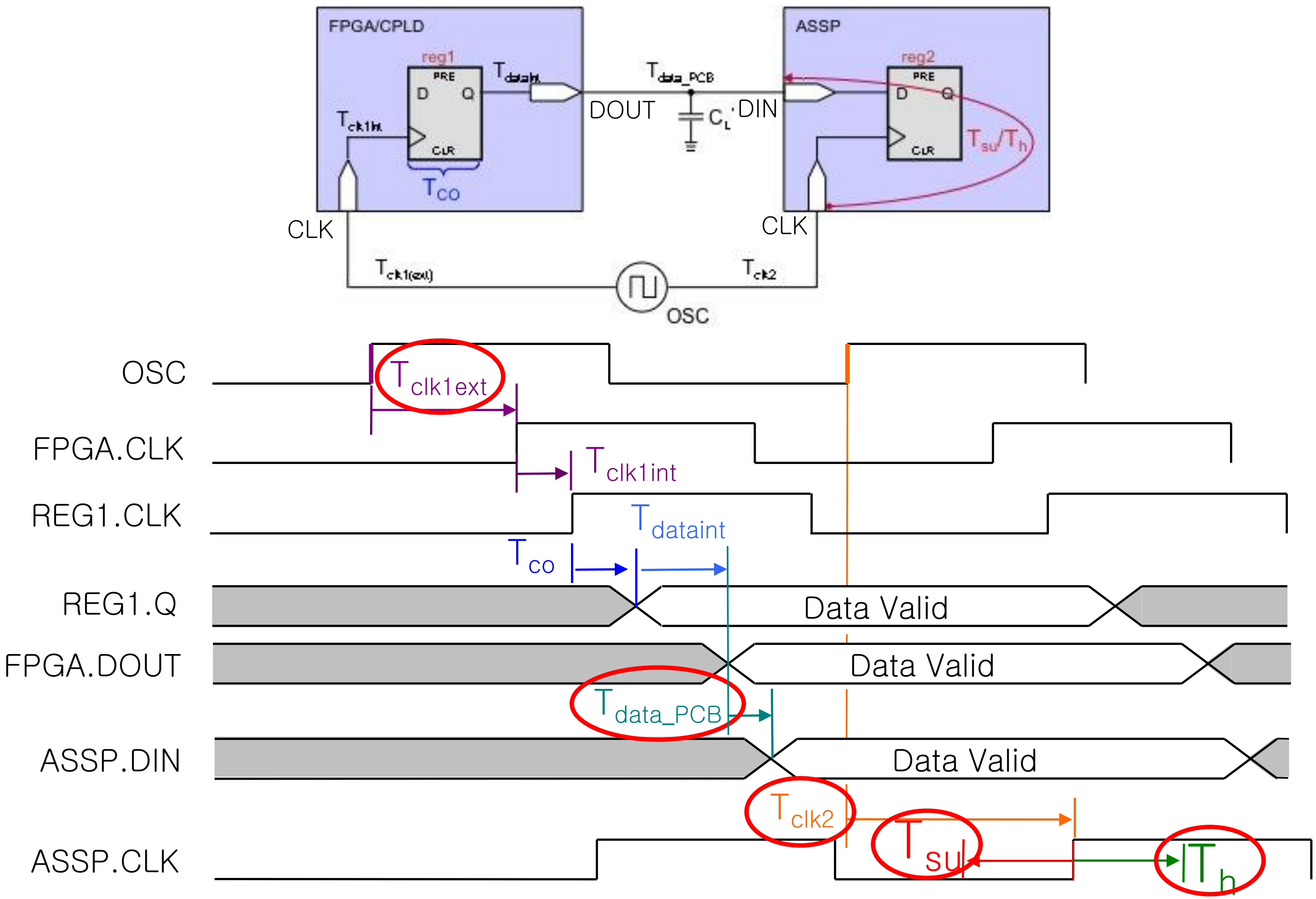


# set_output_delay Command

- Constrains output pins by specifying external devic e timing parameters
- Options

```
-clock <clock_name>
[-clock_fall]
[-rise | -fall]
[-max | -min]
[-add_delay]
[-reference_pin <target>]
<delay value>
<targets>
```

# 同步输出



FPGA/CPLD
reg1
PRE
D
Q
CLR
$T_{clk1int}$
$T_{CO}$
$T_{dataint}$
ASSP
reg2
$T_{data\_PCB}$
DOUT
$C_L$
DIN
$T_{su}/T_h$
CLK
CLK
$T_{clk1(ext)}$
$T_{clk2}$
OSC
OSC
FPGA.CLK
REG1.CLK
REG1.Q
FPGA.DOUT
ASSP.DIN
ASSP.CLK
$T_{clk1ext}$
$T_{clk1int}$
$T_{co}$
$T_{dataint}$
Data Valid
Data Valid
$T_{data\_PCB}$
Data Valid
$T_{clk2}$
$T_{su}$
$T_h$

# 同步输出约束

- Use set_output_delay (-max option) command to constrain maximum clock-to-output (maximum time to arrive and still meet ASSP's $T_{su}$)

**output delay max** **= Board Delay (max) - Board clock skew (min) +** $\mathbf{T_{su}}$

$= (T_{data\_PCB(max)} + T_{CL}) - (T_{clk2(min)} - T_{clk1ext(max)}) + T_{su}$

data arrival time = launch edge + $T_{clk1int}$ + $T_{co(max)}$ + $T_{dataint}$

data required time = latch edge - output delay max

slack = data required time - data arrival time

- Use set_output_delay (-min option) command to constrain minimum clock-to-output (minimum time to stay active and still meet ASSP's $T_h$)

**output delay min** **= Board Delay (min) - Board clock skew (max) –** $\mathbf{T_h}$

$= (T_{data\_PCB(min)} + T_{CL}) - (T_{clk2(max)} - T_{clk1ext(min)}) - T_h$

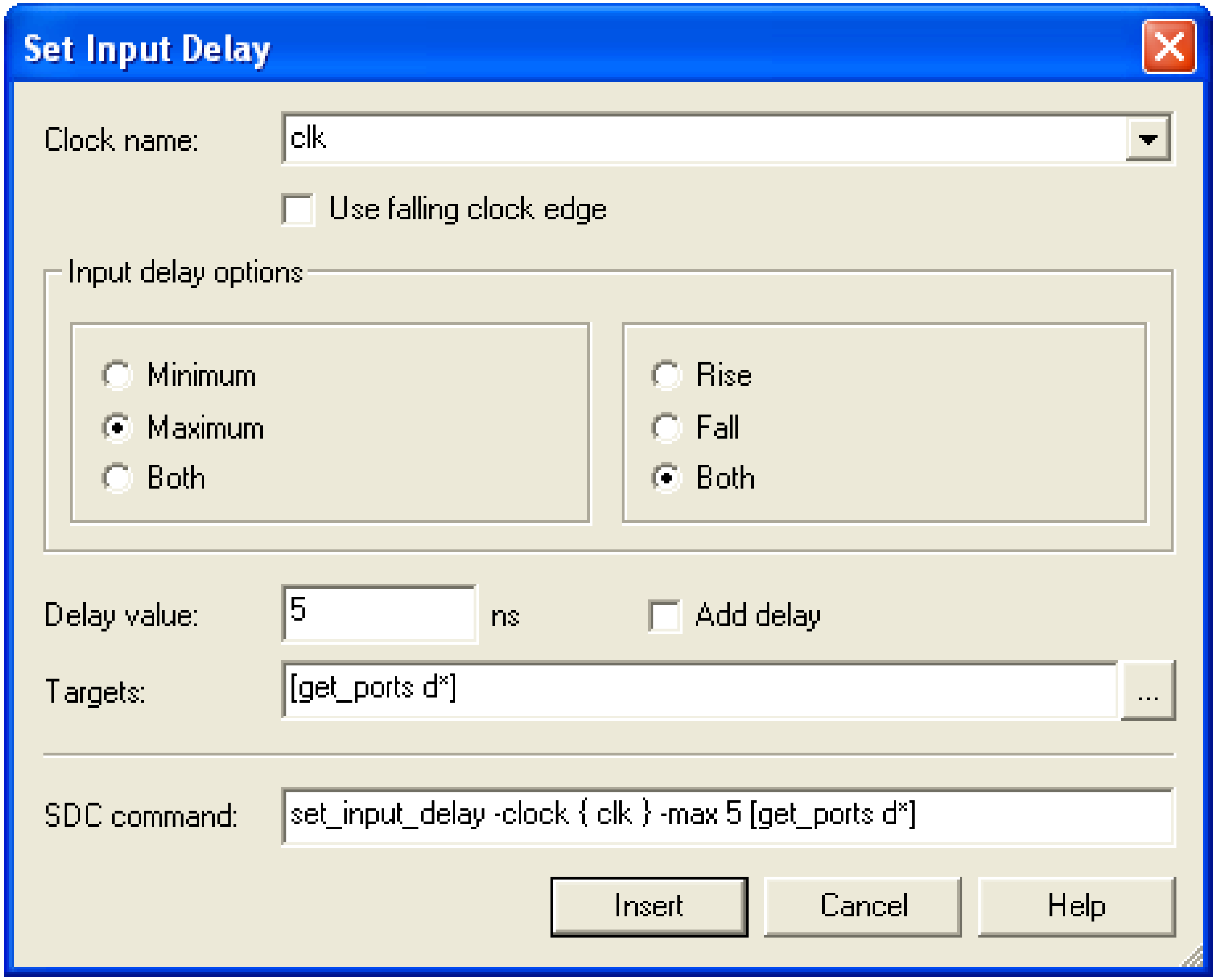
data arrival time = launch edge + $T_{clk1int}$ + $T_{co(min)}$ + $T_{dataint}$

data required time = latch edge - output delay min

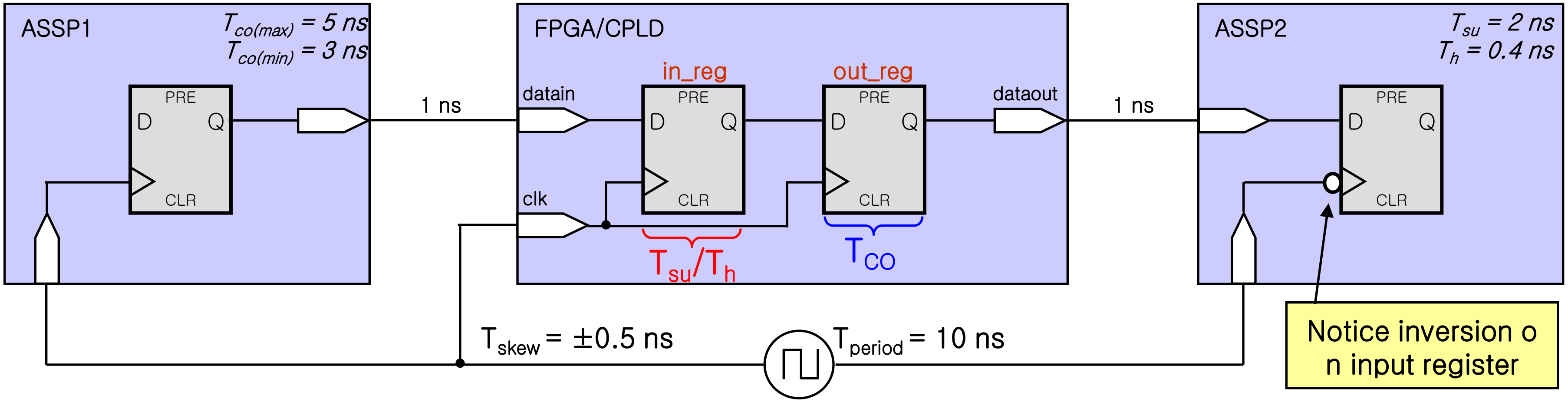
slack = data arrival time - data required time


86

# Input/Output Delays (GUI)



# Synchronous I/O Example




create_clock -period 10 -name clk [get_ports clk]

set_input_delay –clock clk –max [expr 1 – (-0.5) + 5] [get_ports datain]

set_input_delay –clock clk –min [expr 1 - 0.5 + 3] [get_ports datain]

set_output_delay –clock clk –max [expr 1 – (-0.5) + 2] \

-clock_fall [get_ports dataout]

set_output_delay –clock clk –min [expr 1 - 0.5 – 0.4] \

-clock_fall [get_ports dataout]

# SDC Timing Constraints

- Clocks
- I/O
- False paths
- Multicycle paths

# False Paths

- 逻辑相关
  - 正常电路操作间的路径并不相关
  - 比如测试逻辑
- 时序相关
  - 同步电路中的异步时钟穿越信号
- 必须约束好后，让时序分析工具忽略他们

# `set_false_path` Command

- Indicates paths that should be ignored during fitting and timing analysis
- Options

  [-fall_from <*clocks*>]

  [-rise_from <*clocks*>]

  [-from <*names*>]

  [-through <*names*>]

  [-to <*names*>]

  [-fall_to <*clocks*>]

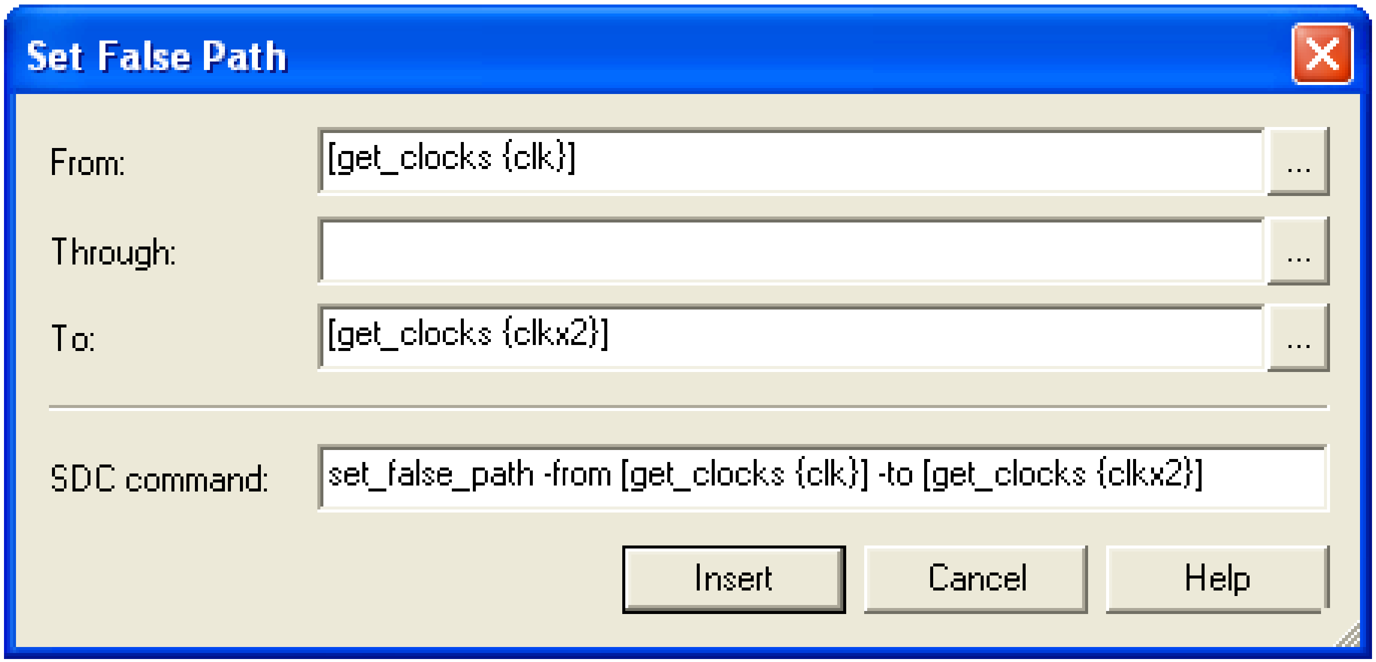
  [-rise_to <*clocks*>]

  [-setup]

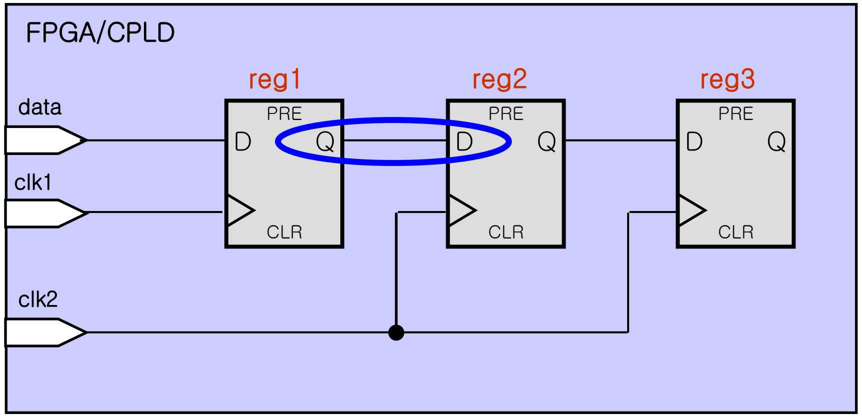
  [-hold]

  <targets>

# Set False Path (GUI)

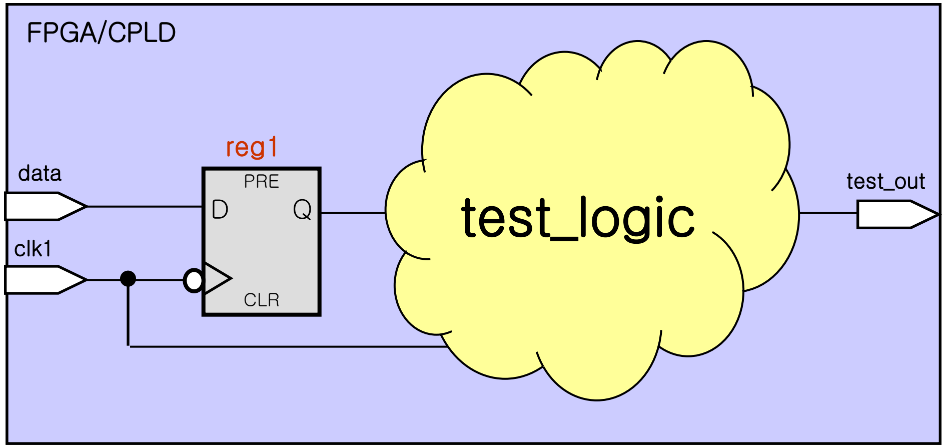


# False Path 实例1




set_false_path –from [get_pins reg1|clk] \
–to [get_pins reg2|datain]

# False Path 实例2




set_false_path –fall_from clk1 \
–to [get_pins test_logic|*|datain]

set_false_path –from [get_pins test_logic|*|clk] \
-to [get_pins test_logic|*|datain]

set_false_path –from [get_pins test_logic|*|clk] \
-to [get_ports test_out]

# SDC Timing Constraints

- Clocks
- I/O
- False paths
- Multicycle paths

# `set_multicycle_path` Command

- Indicates by how many cycles the required time (setup or h old) should be extended from defaults
- Options

  [-start | -end]

  [-setup | -hold]

  [-fall_from <*clocks*>]

  [-rise_from <*clocks*>]

  [-from <*names*>]

  [-through <*names*>]

  [-to <*names*>]

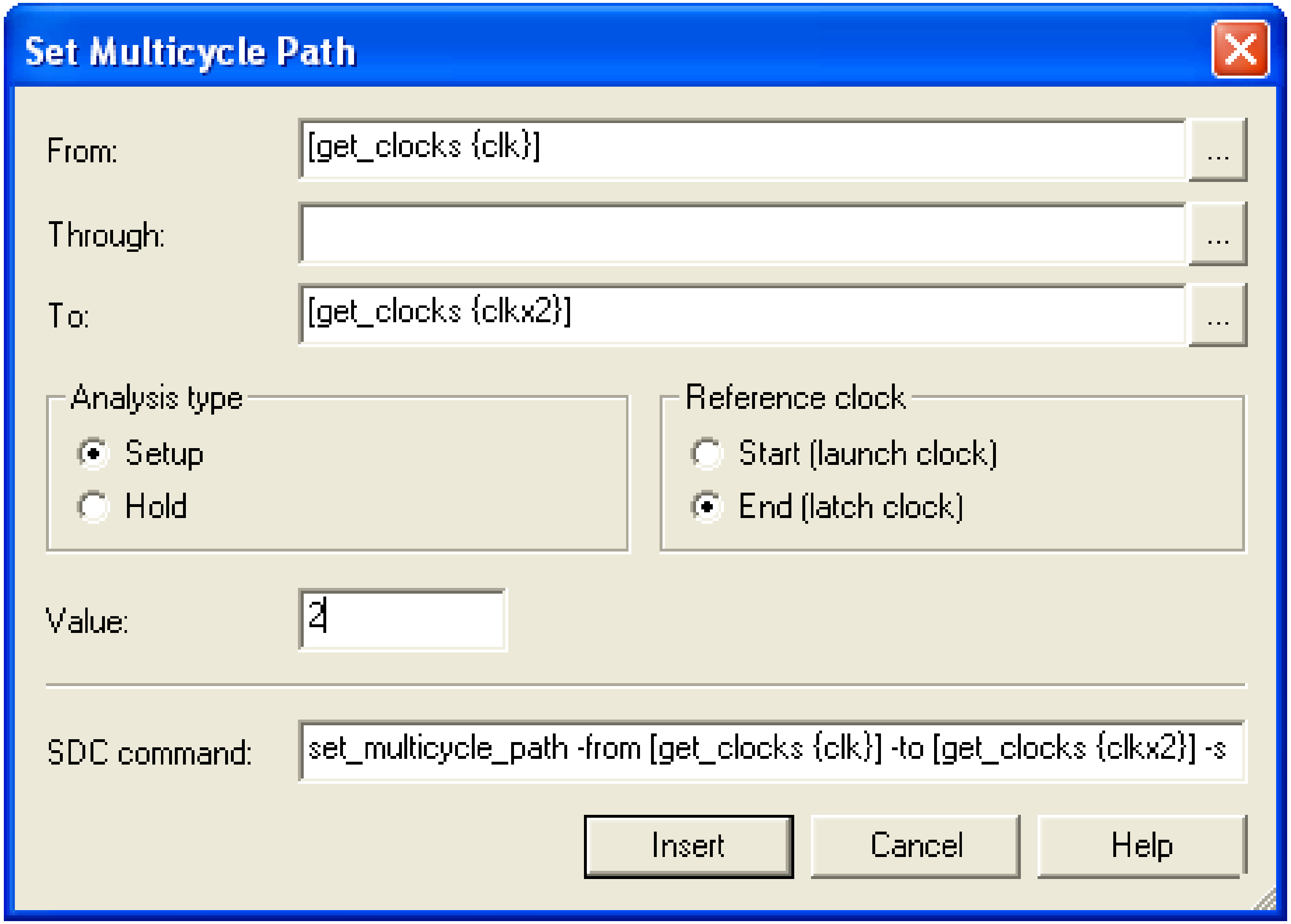
  [-fall_to <*clocks*>]

  [-rise_to <*clocks*>]

  <*targets*>

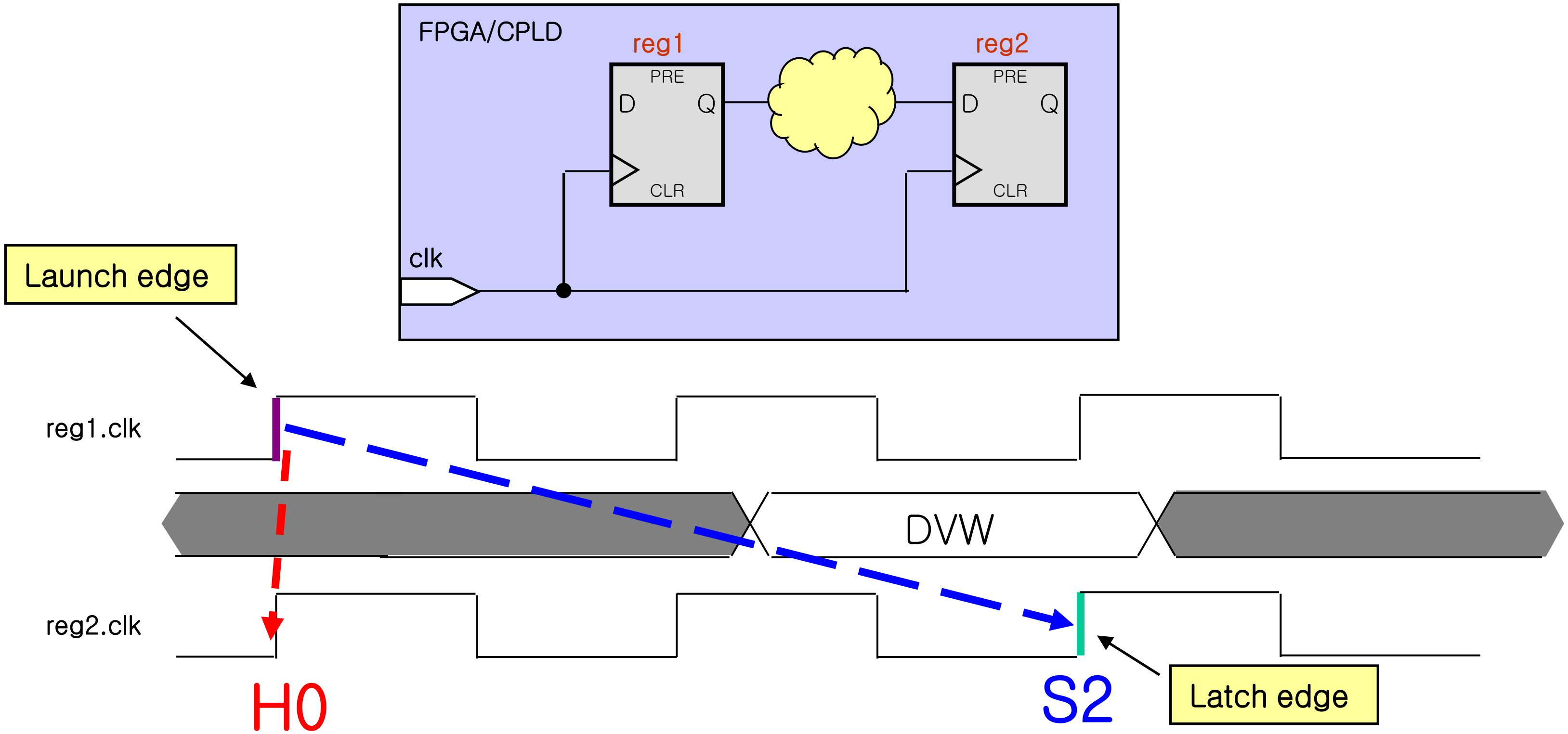
  <*value*>

# Set Multicycle Path (GUI)



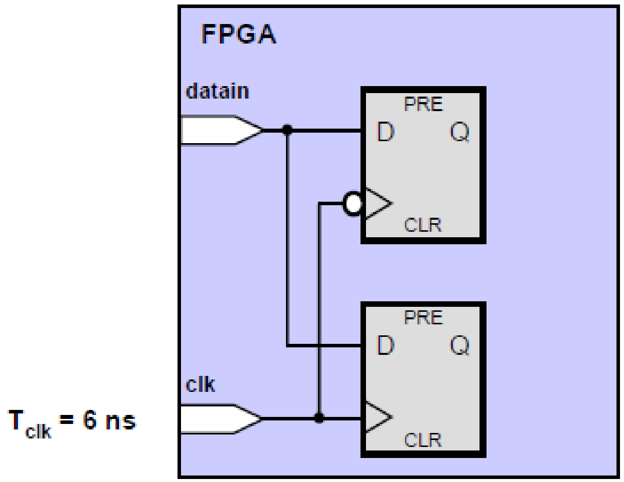
# Multicycle约束实例

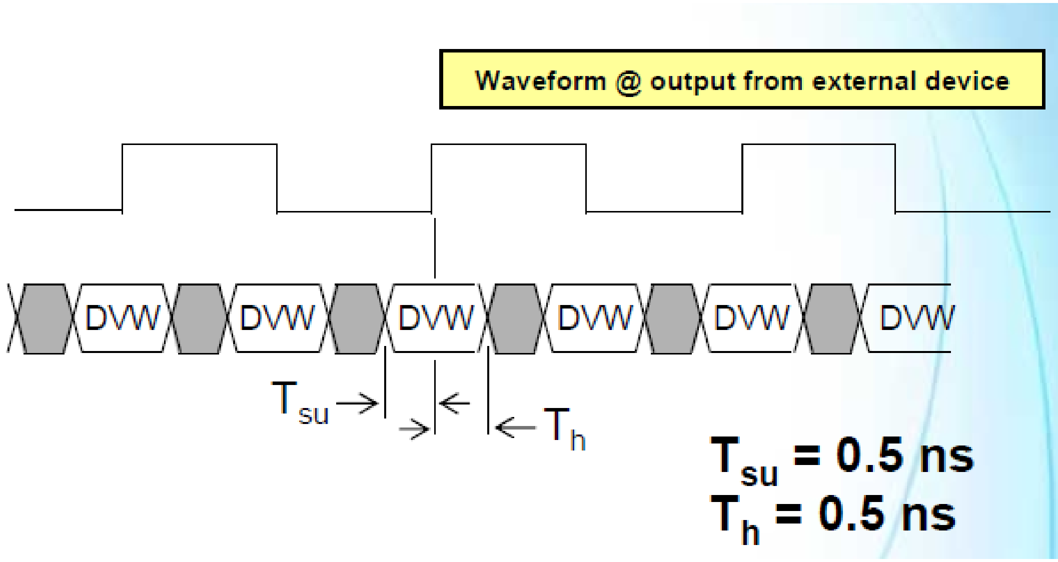
Change to a *two cycle setup*;




set_multicycle_path –from [get_pins reg1|clk] –to [get_pins reg2|datain] \
-end -setup 2

# DDR输入约束




create_clock –period 6 [get_ports clk]

# Rising edge clock constraint

set_input_delay –clock clk –max [expr 6 / 2 - 0.5] datain

set_input_delay –clock clk –min 0.5 datain

# Falling clock edge constraint

set_input_delay –clock clk –max [expr 6 / 2 – 0.5] datain \

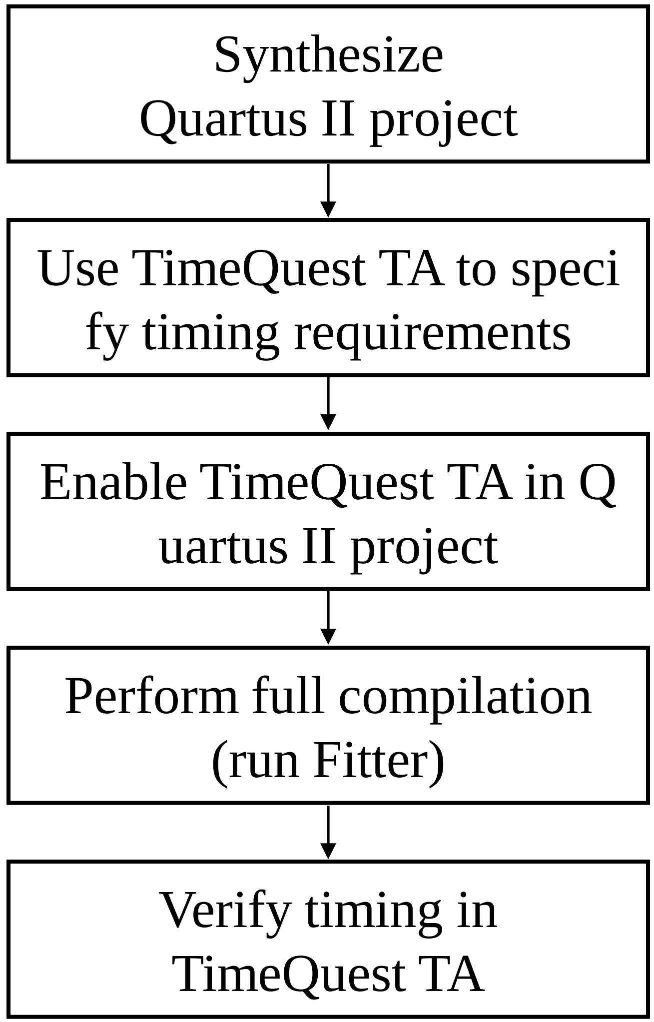
-clock_fall –add_delay

set_input_delay –clock clk –min 0.5 datain -clock_fall –add_delay

# SDC实例

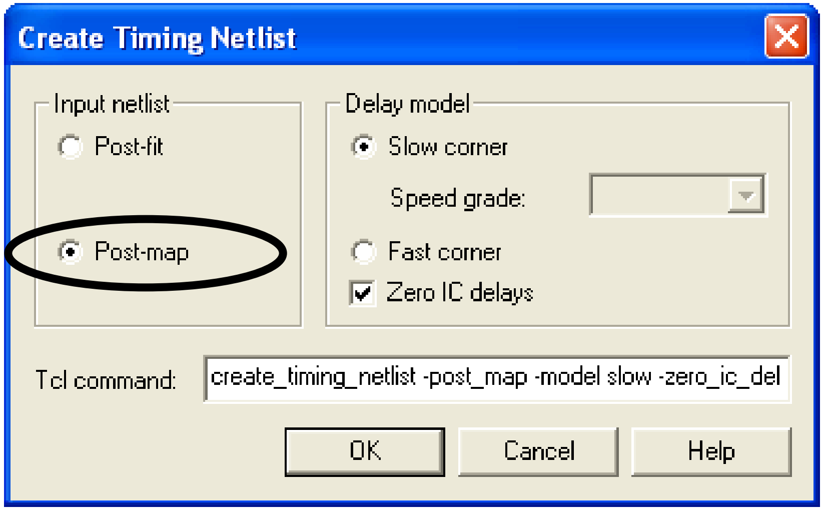
```
create_clock -add -period 36.000 \
        -waveform { 0.000 18.000 } \
        -name xin \
        [get_ports xin]
create_generated_clock -add -source fi_pad|pll_ctrl|ddrc_pll|altpll_component|pll|INCLK[0]
    -name ddr_clk_2x \
        -multiply_by 2 \
        -master_clock xin \
        [get_nodes fiji_pad|pll_ctrl|ddrc_pll|altpll_component|pll|CLK[2]]
#sflash
set_input_delay 23 -max -clock flash_root [get_ports spi_flash_din]
set_output_delay 10 -max -clock flash_root [get_ports spi_flash_dout]
set_false_path -from [get_clocks xin] -to [get_clocks spd_clk_reg]
set_false_path -from [get_clocks spd_clk_reg] -to [get_clocks xin]
set_multicycle_path -hold -from fi_core:fi_core|pl340_dmc_2411:upl340_dmc|pl340_padif_
    2411:u_padif|data_cntl_en_sdr -to ddr_data* 1
set_multicycle_path -setup -from fi_core:fi_core|pl340_dmc_2411:upl340_dmc|pl340_padif
    _2411:u_padif|data_cntl_en_d* -to ddr_dqs[*]  2
```

# Using TimeQuest TA in Quartus II Flow



# Timing Requirements: Create Post-Map Netlist

- Follow TimeQuest flow
- Use -post_map argument for synthe sis (mapping) only netlist
    - If design already fully compiled, choose -p ost_fit (default)
- Tasks list command defaults to post -fit, so must use **Netlist** menu in G UI
- **Zero IC delays** auto-enabled with **Post-map**
    - Assumes no interconnect delays to determi ne if it will be possible to meet timing



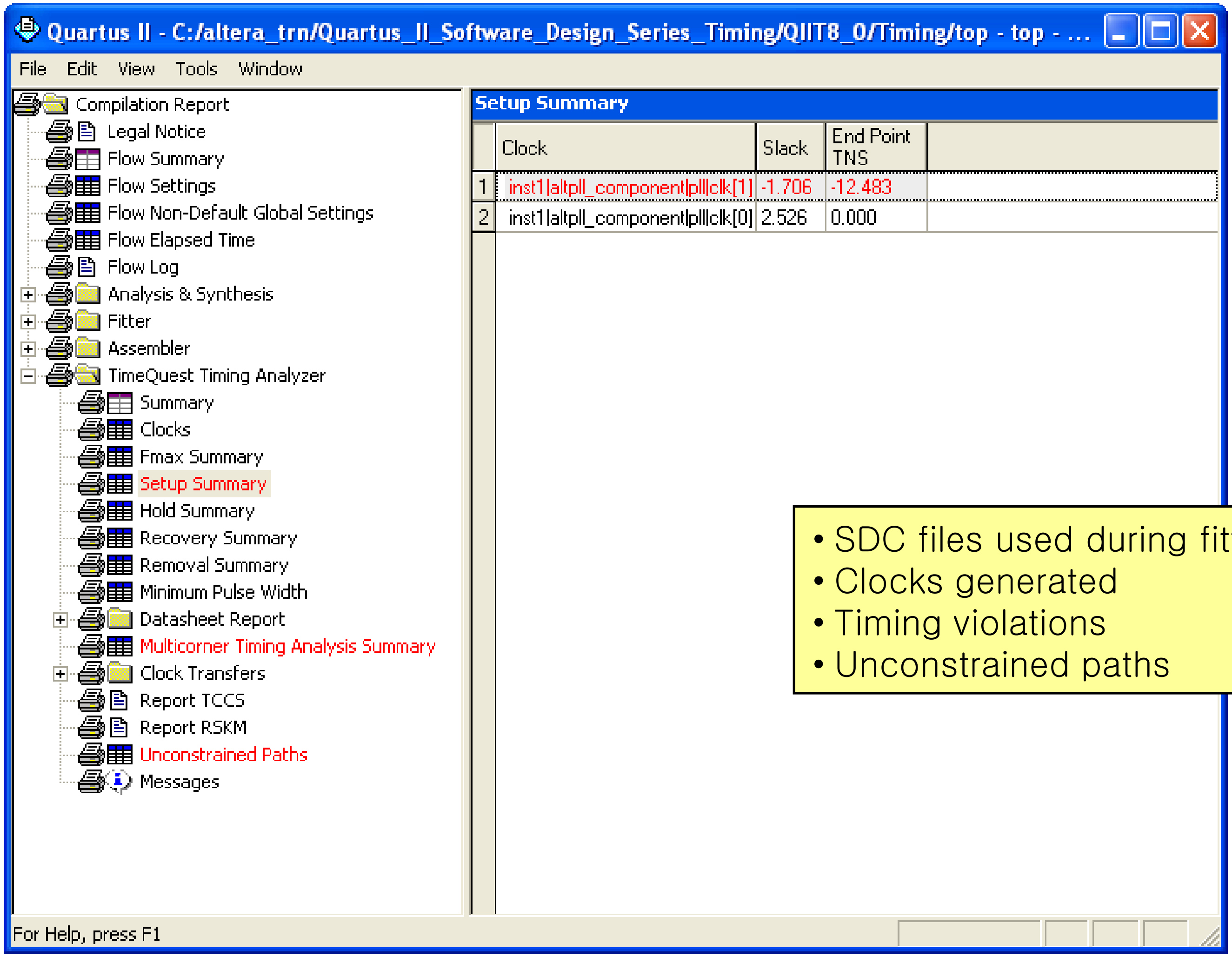
# Timing Requirements: Enter Constraints

- Three ways in two locations
  - SDC File Editor: **Edit ⇒ Insert Constraints** submenu (preferred method)
  - Main TimeQuest window: Enter commands directly into console
  - Main TimeQuest window: Directly into console using GUI dialog boxes in **Constraints** menu
- Be aware of the method you choose!

# Enable TimeQuest TA in Quartus II Software

- Tells the Quartus II software to use SDC constraints during fitting
- File order precedence
  1. Any SDC files manually added to Quartus II project (in order)
  2. *<current_revision>*.SDC located in project directory

# TimeQuest Summary Reports in Compilation Report




- SDC files used during fitting
- Clocks generated
- Timing violations
- Unconstrained paths

# Timing Reports

- Timing results available in both the Quartus II Compilation Report and TimeQuest GUI
- TimeQuest TA includes more extensive reporting capabilities
- Create reports while creating constraints (**post-map** netlist) before fitting to see if design can meet timing requirements
- Create reports after fitting (**post-fit** netlist) to verify that placed & routed design meets timing requirements

# 课程安排

- 时序约束的目的
- 时序约束的内容
- Xilinx FPGA时序约束方法
- Altera FPGA时序约束方法
- **时序约束的原则**

# 时序约束的基本原则

- 通常会对系统的时钟频率约束紧一些，余量根据实际情况大概10—20%；
- 输入和输出的延迟都为同步系统的约束，如果是异步系统就没有意义，但建议延迟设置为时钟周期一半；
- 合理的时序性能约束的原则：60/40 原则
  - 逻辑延迟低于timing budget的60%，时序很容易满足；
  - 逻辑延迟在60%到80%之间，软件run的时间将会提高；
  - 逻辑延迟超过80%，将很难满足时序要求。

# 总结

- FPGA性能与时序约束有着非常紧密的联系
- 时钟和周期约束覆盖了同步单元之间的延迟路径
- 输入延迟约束覆盖了输入管脚到同步单元之间的延迟路径
- 输出延迟约束覆盖了同步单元到输出管脚间的延迟路径
- 虚假路径和多周期路径需要特殊对待
- 用Constraints Editor（Xilinx）和TimeQuest（Altera）进行时序约束