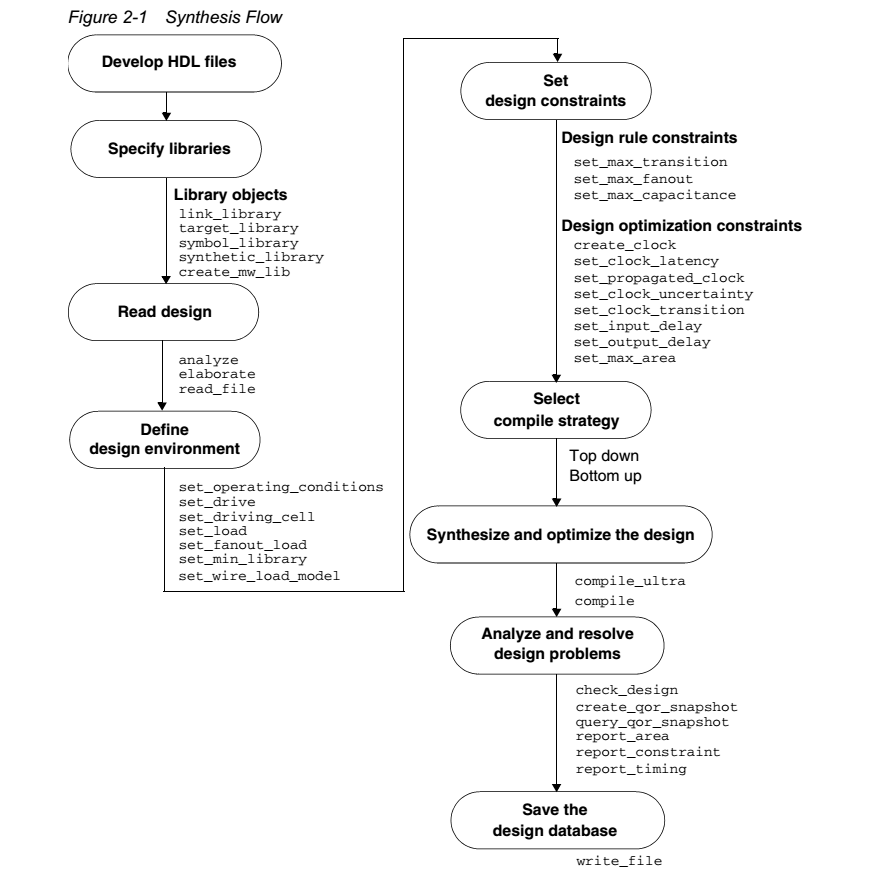
# DC笔记

## DC整体流程



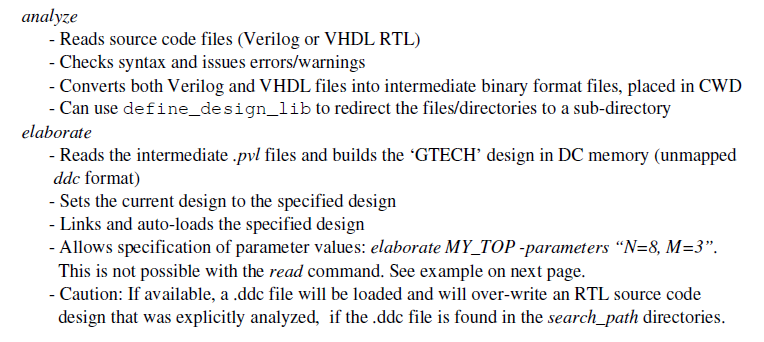
### 准备设计文件、指定库文件

### 读入设计

将设计文件载入内存，并将其转换为DC的中间格式GTECH格式(也即完成综合的translation过程)

读入设计有两种方法：read和analyze & elaborate，最大的区别在于analyze & elaborate会在生成GTECH文件之前生成一个.syn文件存储于工作目录，以便下次elaborate节省时间，而且elaborate可以改变设计的parameter，也是dc中唯一改变设计parameter的方式，采用此种方式之后，便不再需要current\_design和link命令，因为elaborate命令已经指定和链接过了。

下面这张图说明了analyse&elaborate分别在做什么

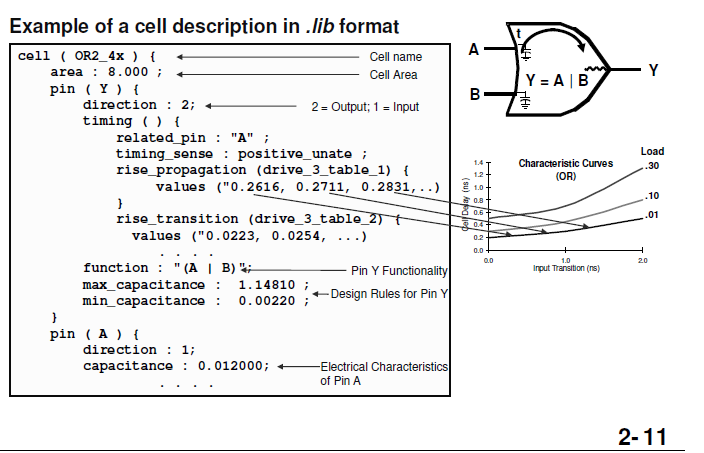


## CCS

composite current source library model，复合电流源库模型

## standard-cell信息

这个。lib库是给设计人员看，通过Synopsys Library Compiler 将.lib转换成DC读取的.db格式。The technology library source is an ASCII file (known as “.lib” file), which is compiled by Synopsys Library Compiler to create a compiled version (known as “.db” file).



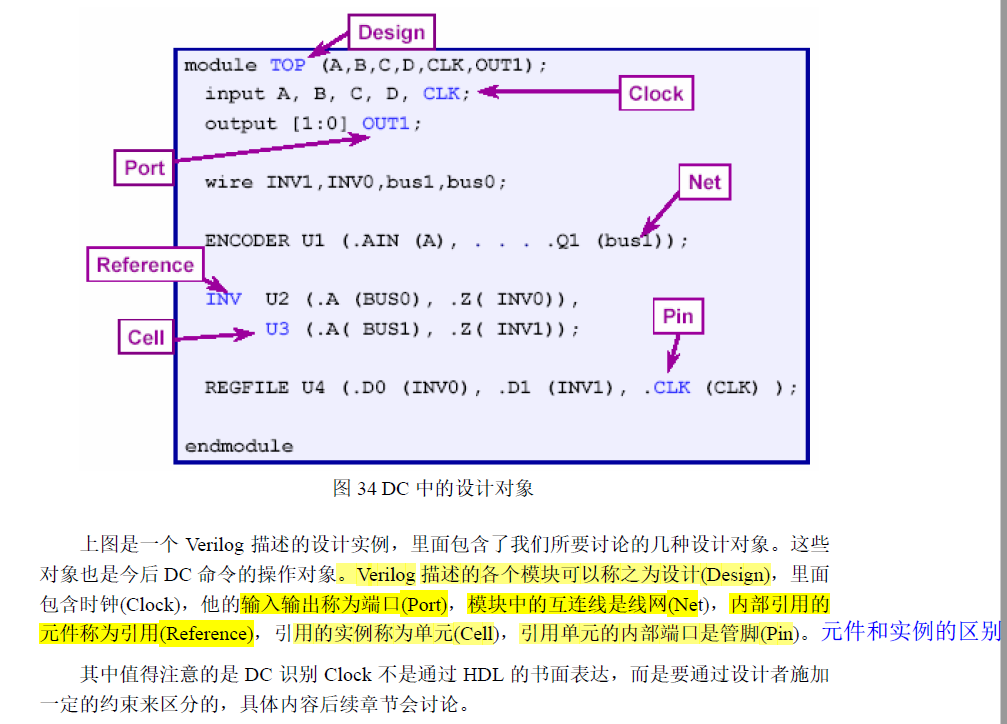
## source -e –v命令

tcl脚本中，source命令：如在一个tcl脚本中 source -e –v ./scripts/dc\_setup.tcl

相当于将source 后面的脚本的内容直接插在此位置，其中-e代表echo，用来显示读取的dc\_setup.tcl中的内容(echo本身的作用就是打印信息)，-v代表verbose，表示执行dc\_setup.tcl之后的结果（盲猜是为了检测该文件是否有bug）

## expr和括号的用法

set b [expr a\*0.5]：expr是表达式计算工具,[ ]作用是将括号中的表达式执行之后，传回执行的结果，即将a\*0.5的值赋给b。

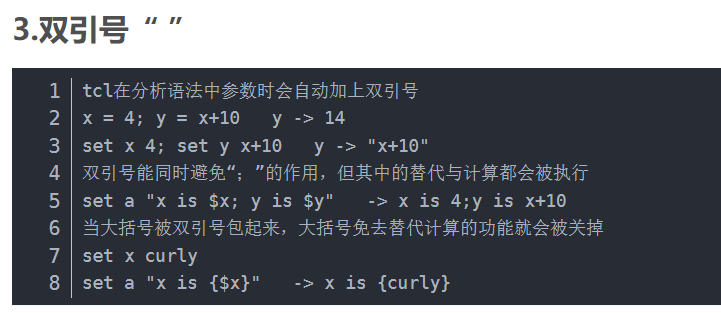


{ }: 与“”作用类似，都是做打包处理，但是会免去大括号中一切的替换和计算，把其中的内容当成子串处理









## DC中collections

是一个集合，不等于tcl语言中的list，两者不能互用，

## remove\_from\_collection语句

remove\_from\_collection：从当前集合中删除一些子集，

remove\_from\_collection base\_collection [-intersect] object\_collection\_or\_list

这个命令可以将在一个集合中的一些子集盒去除。在Innovus或者ICC中，诸如get\_cells, get\_pins等get\_\*命令返回的是一个集合，我们称之为collection。其中，base\_collection是指定的主体集合，object\_collection\_or\_list是指定需要删除的集合

-intersect指需要删除的是输出两个集合中相同的部分，还是不同的部分，

例子：

下面例子可以在当前path group的collection中去掉PathE

>get\_path\_groups \*

PathE PathF

>remove\_from\_collection -intersect [get\_path\_groups \*] {PathE }

PathE

>remove\_from\_collection [get\_path\_groups \*] {PathE }

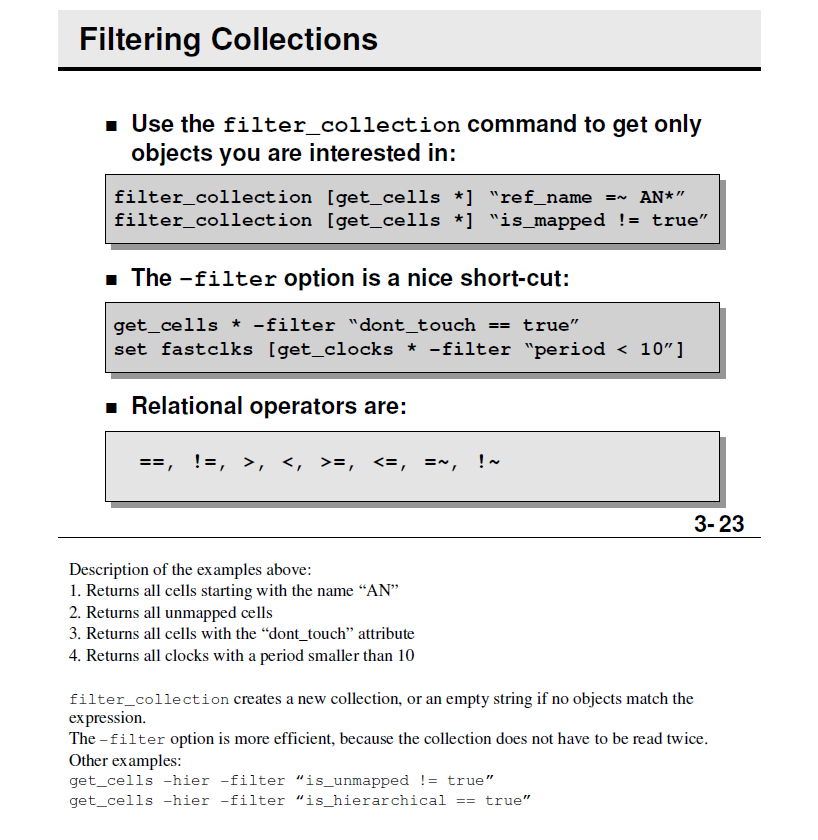
PathF



这条命令是将所有输入中的clk信号除去。

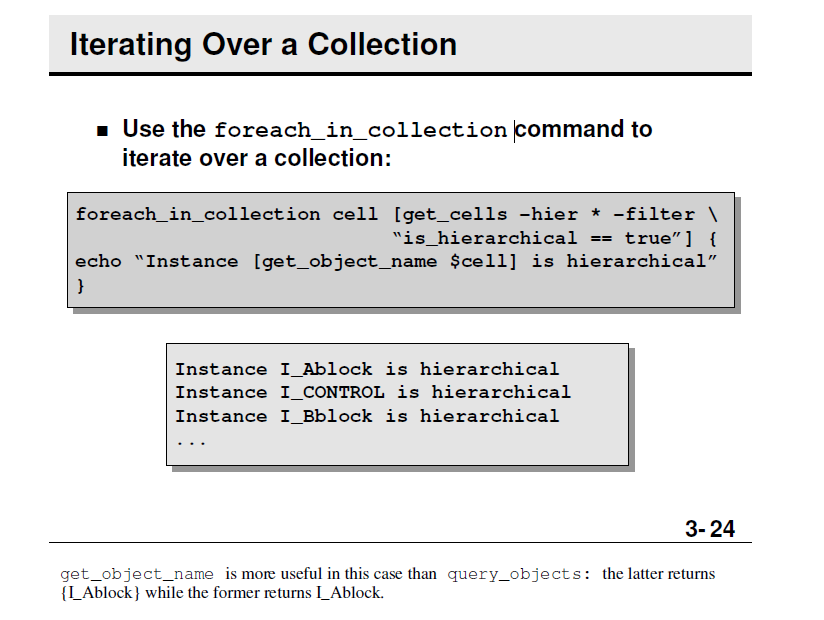
## filter\_collection

对集合进行选择，



## foreach\_in\_collection

对集合进行遍历



## dc\_setup.tcl文件内容

dc\_setup.tcl中主要涉及库和路径的设置，lib\_path是搜索库文件的路径，需要将search\_path设置为这个path，target\_library中的文件是库文件，DC吃的库是db格式，由工艺厂提供，也可以由lib库，用library complier转成db，两者包含的信息相同，格式不同。

## synthetic\_library

synthetic\_library是综合库，主要是负责运算，比如+、-、\*、/等，DesignWare提供这些运算符，其吃的库一般是标准的库，其中DesignWare Basic提供简单的电路，DesignWare Foundation提供性能较高的电路，如超前进位加法器等，一般使用后者的库，如dw\_foundation.sldb。

## target\_library

target\_library：用于设置综合时所要映射的库，target\_library 中包含有单元电路的延迟信息，DC 综合时就是根据target\_library 中给出的单元电路的信息，在满足时序约束条件下，选择最小的门。

## link\_library

link\_library 是链接库，它是DC 在解释综合后网表时用来参考的库。一般情况下，它和

target\_library相同，可能额外增加一些其他的IP库，比如从外面买来的IP库，只提供.db的库，最常用的有SRAM的库，PLL的库，或者用来负责解释运算的综合库synthetic\_library，这时候我们就需要将其放在链接库中；当使用综合库时，需要将该综合库加入链接库列表中。注意：在link\_library 的设置中必须包含’\*’, \*表示DC的memory，表示DC 在引用实例化模块或者单元电路时首先搜索已经调进DC memory 的模块和单元电路，如果在link library 中不包含’\*’，DC就不会使用DC memory 中已有的模块，因此，会出现无法匹配的模块或单元电路的警告信息(unresolved design reference)。

## symbol\_library

symbol\_library为指定的符号库。symbol\_library 是定义了单元电路显示的Schematic 的

库。用户如果想启动design\_analyzer 或design\_vision 来查看、分析电路时需要设置

symbol\_library，如果不指定的话，工具会调用自带的符号库。简单来说就是看电路结构用的。

## synthetic\_library

synthetic\_library 是DesignWare 综合库，在初始化DC 的时候，不需要设置标准的

DesignWare 库standard.sldb 用于实现HDL 描述的运算符，对于扩展的DesignWare，需

要在synthetic\_library中设置，同时需要在link\_library 中设置相应的库以使得在链接的时候DC 可以搜索到相应运算符的实现。

## search\_path

search\_path 指定了综合工具的搜索路径。

## gtech.db

gtech库，DC工具能够解析底层库，相当于python，C++等高级语言在编译过后，最终的表现形式是机器码，gtech就相当于机器码对应的库。由DC自带，是独立于厂家工艺的。该库中包含的元件仅代表一定的逻辑功能而不带有任何工艺参数

## WLM

WLM：wire load model，线负载模型。

## max\_transition max\_fanout的设定

在一般的设计中，需要人工调整的设计规则约束主要是最大转换时间（max\_transition）和最大扇出负载（max\_fanout），其他的约束一般采用库中的缺省值就可以了。下面详细说明max\_transition和max\_fanout 的含义和设定方法。

（1） 设定max\_transition。连线的转换时间是其驱动引脚的逻辑电平变化所需要的时间。DC 计算转换时间的方法是用驱动引脚的驱动强度乘以它驱动的连线上的所有容性负载的综合。设定最大转换时间的命令时set\_max\_transition。设定的对象可以是端口，也可以是整个设计。这个值一般设为周期的10%，在综合的时候对于负载过大的信号DC 会通过插入buffer 降低该信号的负载。

（2） 设定max\_fanout。从前面我们已经了解到，连线的扇出负载并不是一个电容值，而是指它所驱动的所有输入引脚的扇出负载的总和。一个输入引脚的扇出负载在一般情况下为1，但工艺库的开发人员也可能为输入引脚指定更高的扇出负载值，以模拟内部单元的扇出效应。设定max\_fanout 使用set\_max\_fanout 命令。max\_fanout 经验值一般设为20，即一个门的输出端最多驱动20 个输入引脚（如果每个引脚的扇出负载为1 的话）。

## input\_delay 和 output\_delay的设定

设置输入端口延时的命令是set\_input\_delay，设置输出延时的命令是set\_output\_delay。

一般情况下，若不知道外部电路对输入延时的要求，可设输入延时为整个时钟周期的40%。

而且在同步设计中，输出一般应由寄存器输出，在这种情况下，设计可不加输出延时约束。如果有组合逻辑的输出端口，则在相应的端口应设置输出延时，这时应首先估计信号输出后到下一个寄存器有多长时间的延时，用时钟周期减去这个时间就是留给图中C2的延时，可以将输出延时设为这个值。（为时钟周期的60%，涛哥的约束）。

## 综合报告的分析

报告中有个timeing\_max\_path的报告，报告的是各路径的setup time的情况，timeing\_min\_path则反映的是hold time的情况，所有的timing 上的violation 都会在con\_violations.rpt 中报告。

在DC 之后，hold time的violation 可以暂时忽略，并且这种violation 也不会影响在modelsim 中的后仿。在后面Astro 布局布线阶段会重新插入buffer 来解决这些hold time 的violation。Setup time 的violation 一定要解决，这种violation 会直接导致后仿通不过。产生这种violation 一般会有两种原因：1）clock uncertainty 设置过大，一般这个参数设为周期的10%就可以了。2）路径过长，如果出现这种问题就只能修改原来的设计，以使路径变短。这要求我们在写RTL 级代码的时候要清楚我们设计的电路大致是什么样的，一条路径大概会经过多少gate，避免过长的路径出现。一般要求一条路径不要超过20 个gate。另外用于时钟同步的触发器上的时序violation 可以忽略，因为这种触发器允许出现亚稳态。

## .ddc文件

包含网表信息、sdf时序信息、sdc约束信息，约等于.v+.sdc，但是包含的信息比这个更加丰富

## set和set\_app\_var命令

set: 自己自定义的一些变量，方便定义系统变量的时候，不用那么麻烦；主要是定义。

set\_app\_var：定义DC内部的系统变量、指定DC内部的连接等。主要指定有：search\_path、synthetic\_library、target\_library、link\_library、symbol\_library 以及其他的一些命令开关等。

## printvar、get\_app\_var和echo

这三个命令都可以将变量的值给打印出来

## read\_verilog命令和current design

前者用来读取.v文件（实验室中用的较少），如果不设置current design，工具会自动设置顶层，当一行一行读文件的时候，会将最后读入的一个设置为顶层；当同时读入多个文件的时候，会将第一个读入的设置为顶层。



## analyze和elaborate命令

这两个是一个命令对，用来读取设计的.v文件（只支持VHDL和verilog），形式如下:

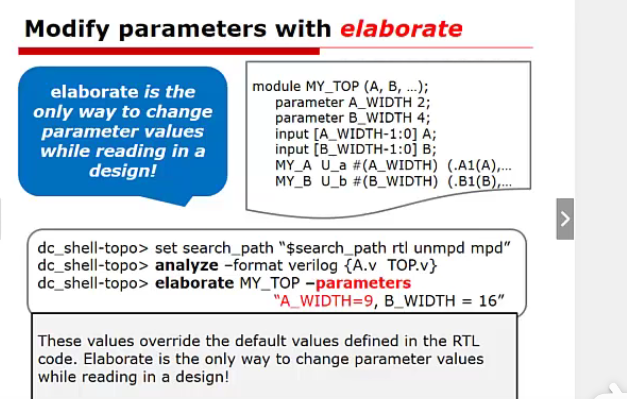


-format用来指定文件格式，待读取的文件可以写成一个.v的文件格式，在该文件中，将设计通过`include命令包含进来。也可以直接用大括号，将待读取的文件放在后面，比如：{TOP.v A.v B.v}。



其中-define命令用来对代码中的宏进行定义，一般不常用，因为可以在代码中直接定义，有点TKZFP的意思。

此命令对主要的功能是：analyze将读入的设计转化成中间格式(.pvl)，存储在当前目录下(如果用read\_verilog命令读取可能会造成分析代码的时候报出cannot find the design或者unable to resolve reference的错误，是有可能，不是一定)，并且会检查设计代码是否有问题，再由elaborate命令将其转化成GTECH格式，同时此命令还会用DesignWare组件替换代码中的HDL算术运算符，并且会自动执行link命令，来解析设计中的模块实例化（所以才不会报前面的错误），最后比较特别的功能是，此命令是唯一可以在读入设计的时候对参数进行修改，如下图所示。



## current\_design命令

用来设置当前设计的顶层模块，一般将顶层的模块称为一个design

## link命令

将所有的库组件和设计连接起来，此过程叫做链接设计或者解析模块实例化。

## check\_design命令

此命令是用来检测当前的设计是否存在问题，当返回值为0时表示出错，当返回值为1时才可能能综合，但也要查看其中的warning。

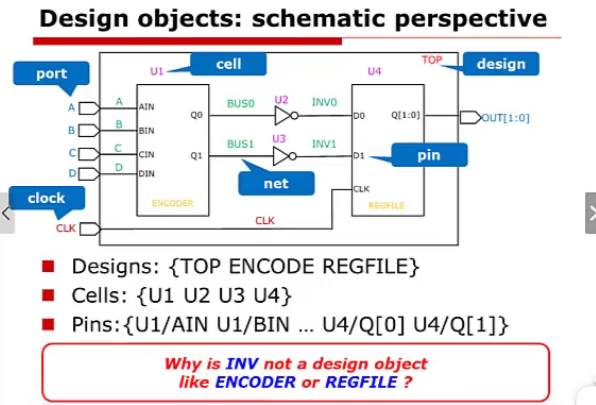
## list\_designs和current\_design

这两个命令可以列出当前代码中的设计模块有哪些，前者列出所有的，其中带\*号的为顶层，后者如果不带参数的话可以，直接列出顶层名

## 设计objects，各个设计对象的意义

在 DC 中，总共有8 种设计对象（另说是6种，除了reference，library）：

* 设计(Design)：一种能完成一定逻辑功能的电路。设计中可以包含下一层的子设计。
* 单元(Cell)：设计中包含的子设计的实例。
* 参考(Reference)：单元的参考对象，即单元是参考的实例。
* 端口(Port)：设计的基本输入输出口。
* 管脚(Pin)：单元的输入输出口。
* 连线(Net)：端口间及管脚间的互连线。
* 时钟(Clock)：作为时钟信号源的管脚或端口。
* 库(Library)：直接与工艺相关的一组单元的集合。



## 设置约束

### 静态时序分析主要步骤

静态时序分析确定电路在综合中以及之后是否满足时序约束,包括三个主要步骤:

1) 将设计打散成很多条时序路径

2) 计算每条路径的延迟

3) 将路径延迟与数据预期到达时间进行比较，来确定是否满足约束

### 时序路径

DC会将设计打散成很多条时序路径,每条会有一个起始点和终点:



以上述图为例,sequential device的非时钟输入引脚:D(数据)，S(设置)或R(重置)，E(使能)。

### set\_max\_area

设置最大面积约束，一般在实验室的脚本中，不会对面积进行约束，因为don’t care，如果不设置，DC会默认执行最小面积优化

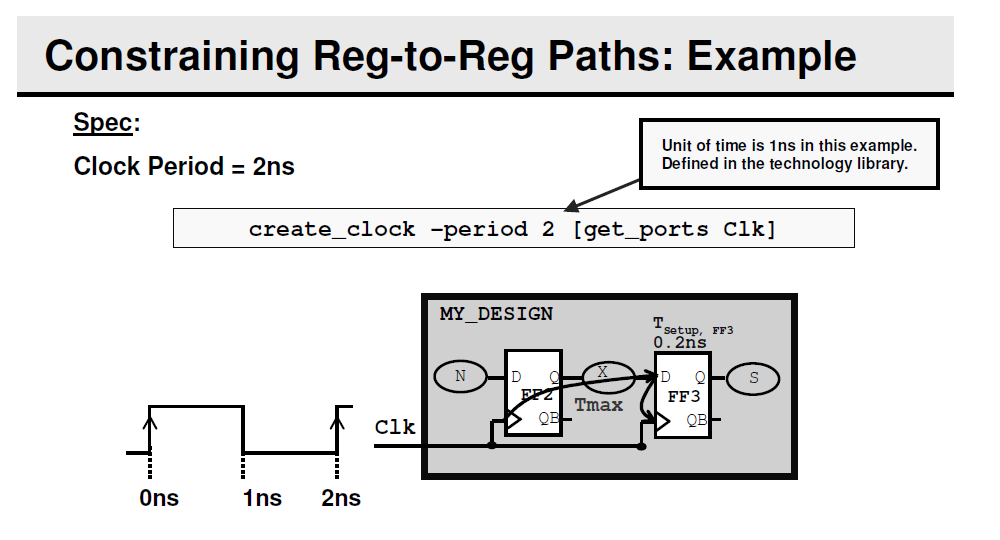
### create\_clock

时钟实际上是对reg-to-reg的一种约束，其中周期的单位与technology library中定义的相同，一般为ns，可以通过report\_lib [library\_name]来查看时间单位。

一般情况下，DC不会对时钟进行插buffer，即便clk驱动了很多引脚，在做后端的时候会有专门的时钟树综合（CTS）。

命令相关参数：

* -name：定义创造的这个时钟名字，当采用get\_clocks命令获取此时钟的时候，后面加的就是这个名字，比如create\_clock –name clk\_name –period 2 [get\_ports clk]，得到此时钟的命令为：get\_clocks {clk\_name}（对单个时钟可以不加大括号，如果有多个时钟，需要用大括号，时钟之间用空格分开，不是用逗号）
* -waveform：指定时钟的占空比，一般不设置的时候，默认为从0时刻为上升沿，0.5\*period的时候为下降沿。-period 20 -waveform {10，15.5}表示10ns时刻为上升沿，15.5ns为下降沿，即空比为（15.5-10）/20=27.5%



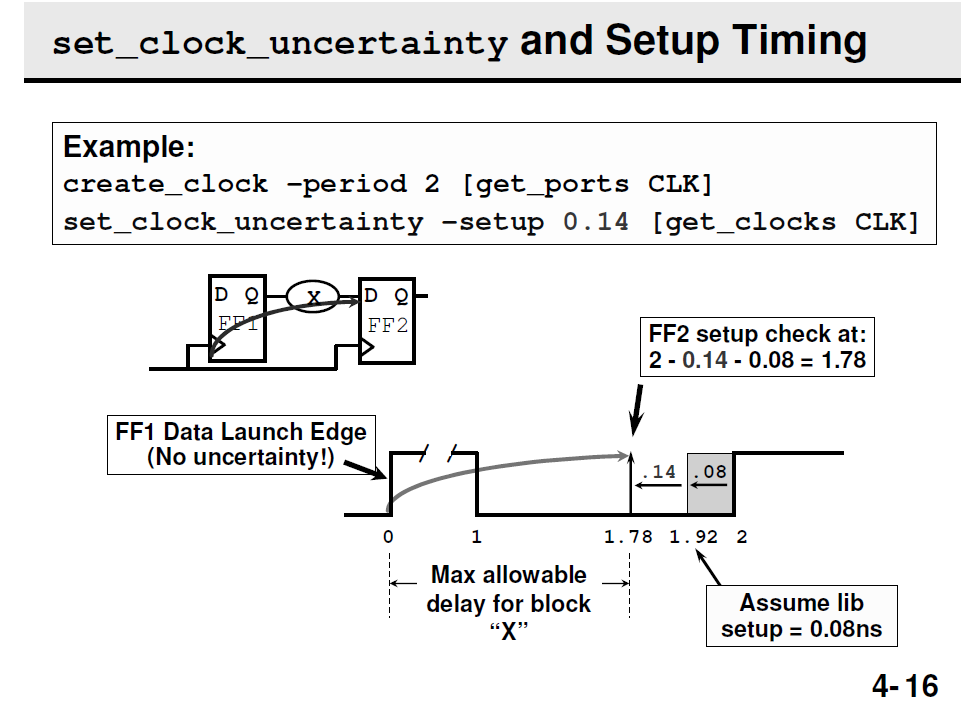
### set\_clock\_uncertainty

模拟时钟的skew和jitter

命令相关参数：

* -setup：表示将不确定的值只运用在setup time的时序检查上，如下图。
* -hold：表示将不确定的值至运用在hold time的时序检查上。、
* 还可以指定从clk1到clk2的不确定值，详细的看man set\_clock\_uncertainty

如果不指定-setup和-hold，set\_clock\_uncertainty命令会将此不确定的值运用在setup time，hold time两个的检查上，即对建立和保持时间的约束是一样，在时序宽松的时候可以这样做，但是在高频的时候，最好还是分开约束。

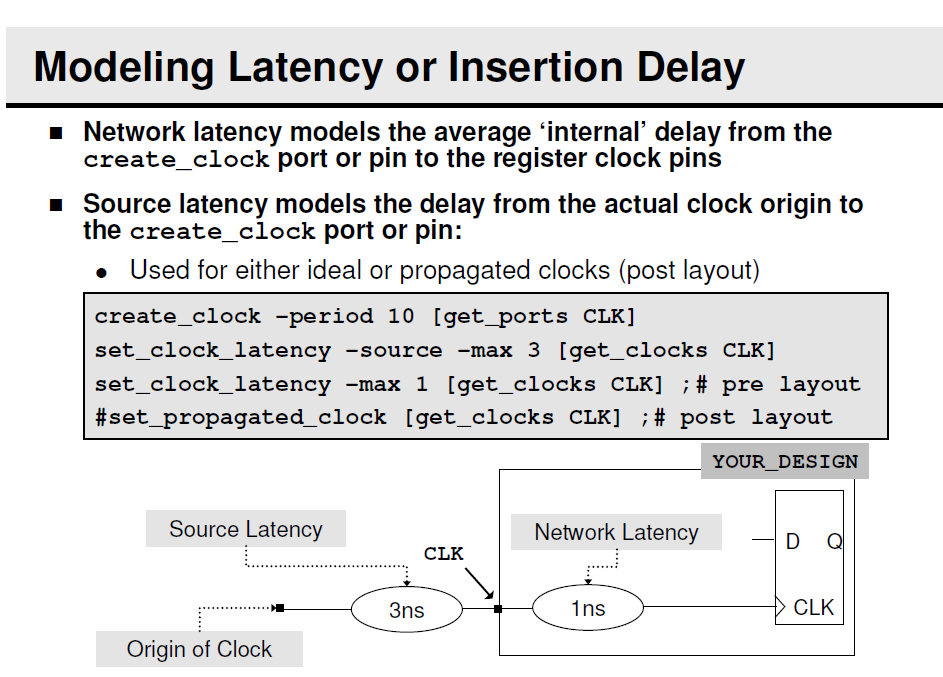


### set\_clock\_latency

模拟时钟源到reg的clk端的延时（也可以用于）。

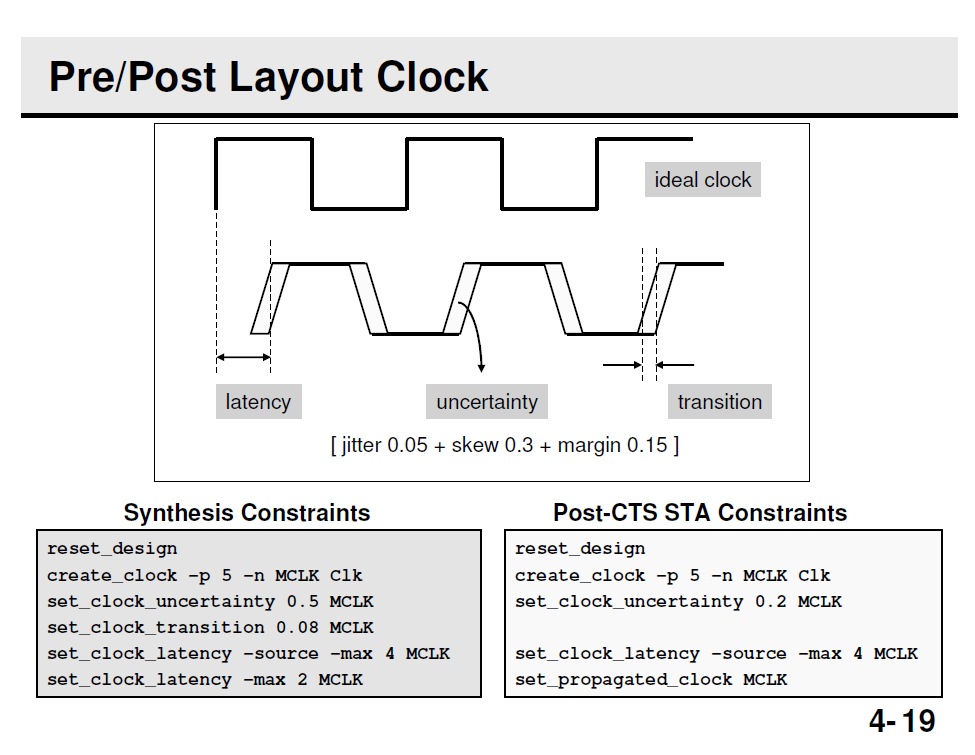
参数：

* -source：模拟的时钟源到current\_design的时钟端（port）source latency，如果不指定-source，则默认为network latency，即从当前设计的时钟端到reg的clk端的延时。
* -max，-min：模拟network latency的最大、最小延时，如果不指定最大、最小，则在最大和最小延时检查的时候，都采用此延时。
* -rise，-fall：指定上升、下降延时。



### set\_clock\_transition

模拟寄存器clk端口的上升和下降时间。



### set\_input\_delay、set\_output\_delay

input\_delay是指该输入信号是在时钟沿后多长时间到达模块的port上。

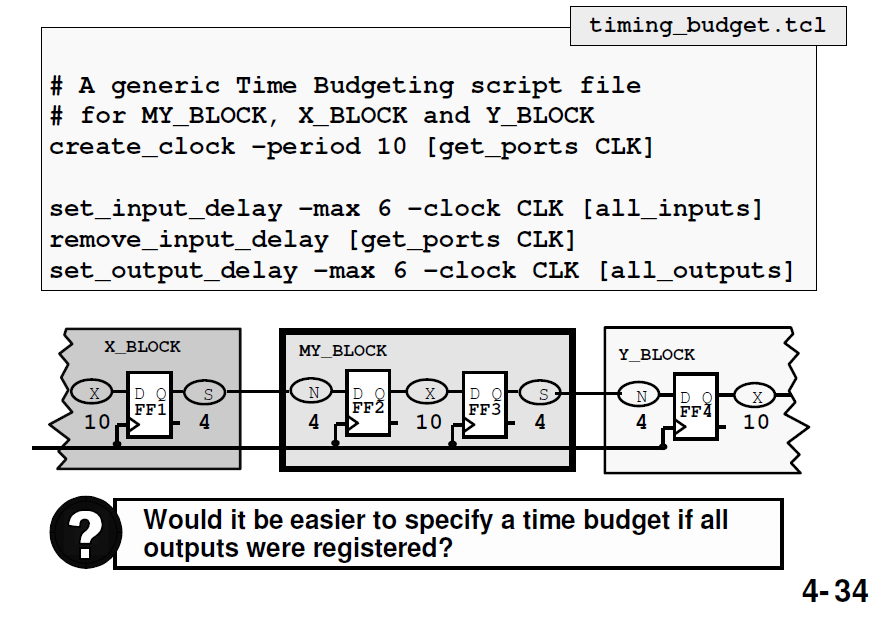
output\_delay是指该输出信号在后级模块中需要在时钟沿之前提前多长时间准备好。

设置输入端口延时的命令是set\_input\_delay，设置输出延时的命令是set\_output\_delay。

一般情况下，若不知道外部电路对输入延时的要求，可设输入延时为整个时钟周期的60%。

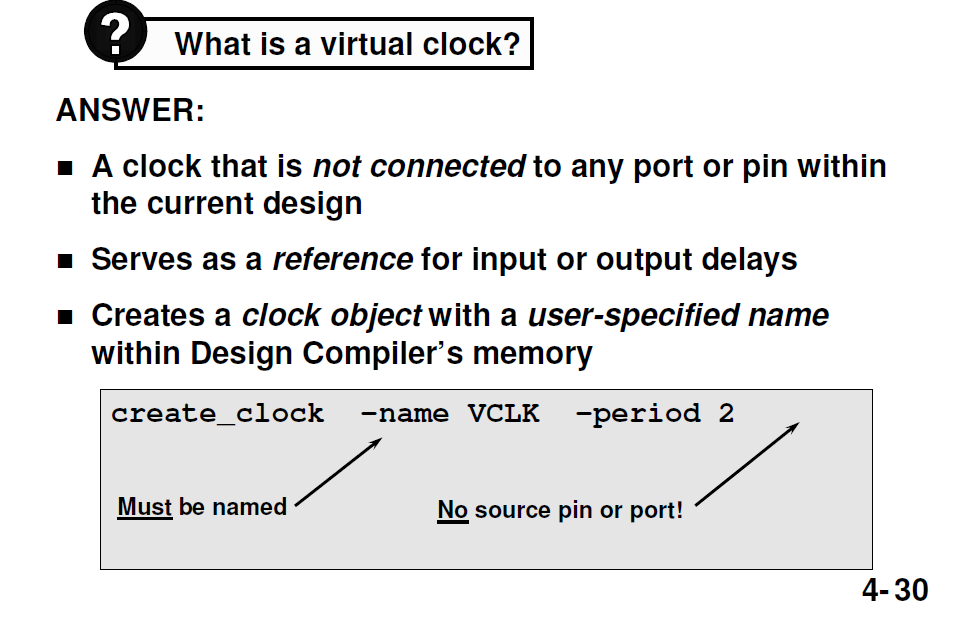
而且在同步设计中，输出一般应由寄存器输出，在这种情况下，设计可不加输出延时约束。如果有组合逻辑的输出端口，则在相应的端口应设置输出延时，这时应首先估计信号输出后到下一个寄存器有多长时间的延时，用时钟周期减去这个时间就是留给图中C2的延时，可以将输出延时设为这个值。（为时钟周期的60%，涛哥的约束）。

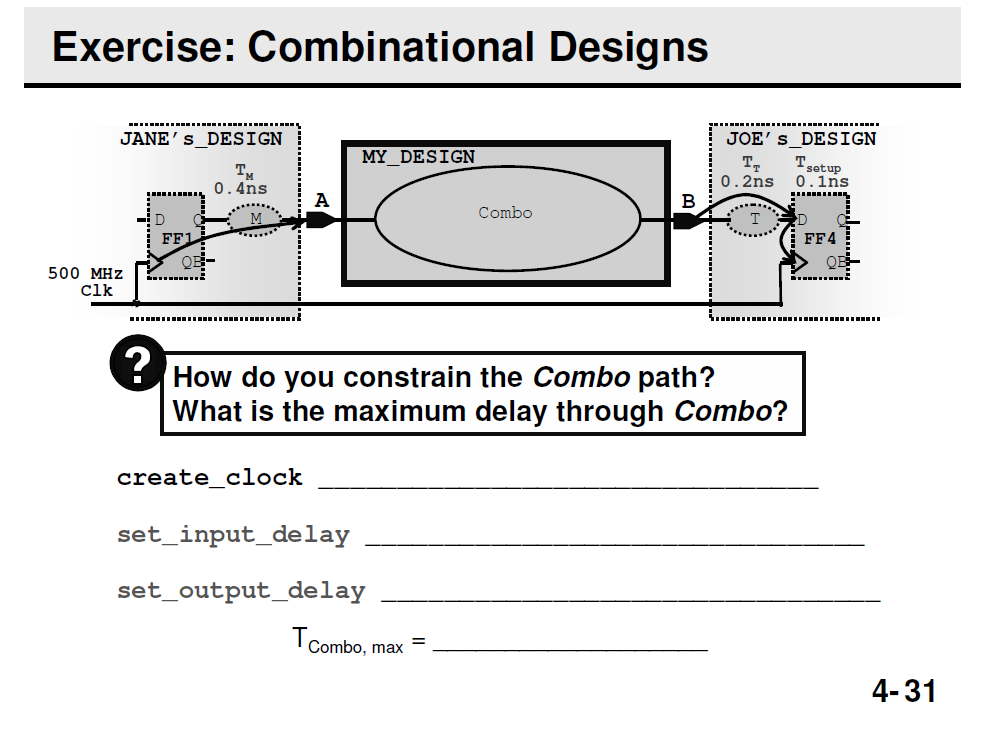
对于不知道外部电路的情况时，一般将input\_delay和output\_delay都设置为时钟周期的60%，用此种方法，如果前后级模块也是采用相同的约束，则会留出20%的margin。如下图所示。



### 纯组合逻辑电路的约束

纯组合逻辑没有时钟，这时候需要创建一个虚拟时钟，





create\_clock –name VClk –period 2

set\_input\_delay –clock VClk –max 0.4 [get\_ports A]

set\_output\_delay –clock VClk –max 0.3 [get\_ports B]

TCombo, max = 2 – 0.4 – 0.3 = 1.3ns

### reset\_design

将当前设计的约束全部去除掉。在进行综合约束文件的设置之前，一般先reset\_design来去除所有的约束。

### dcprocheck

unix自带的tcl脚本语法检查命令，不能在dc\_shell下使用，需要在unix或者linux下使用，使用方法为：dcprocheck constraints.tcl。

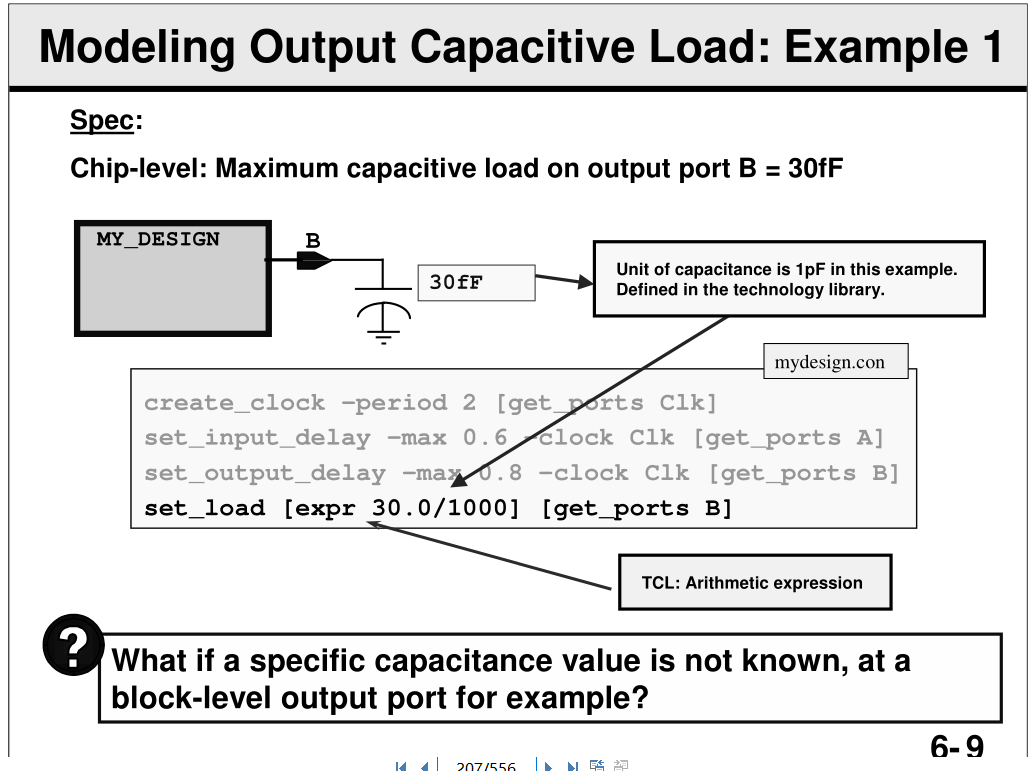
在dc\_shell下使用时，需要加上sh命令，如：sh dcprocheck constraints.tcl

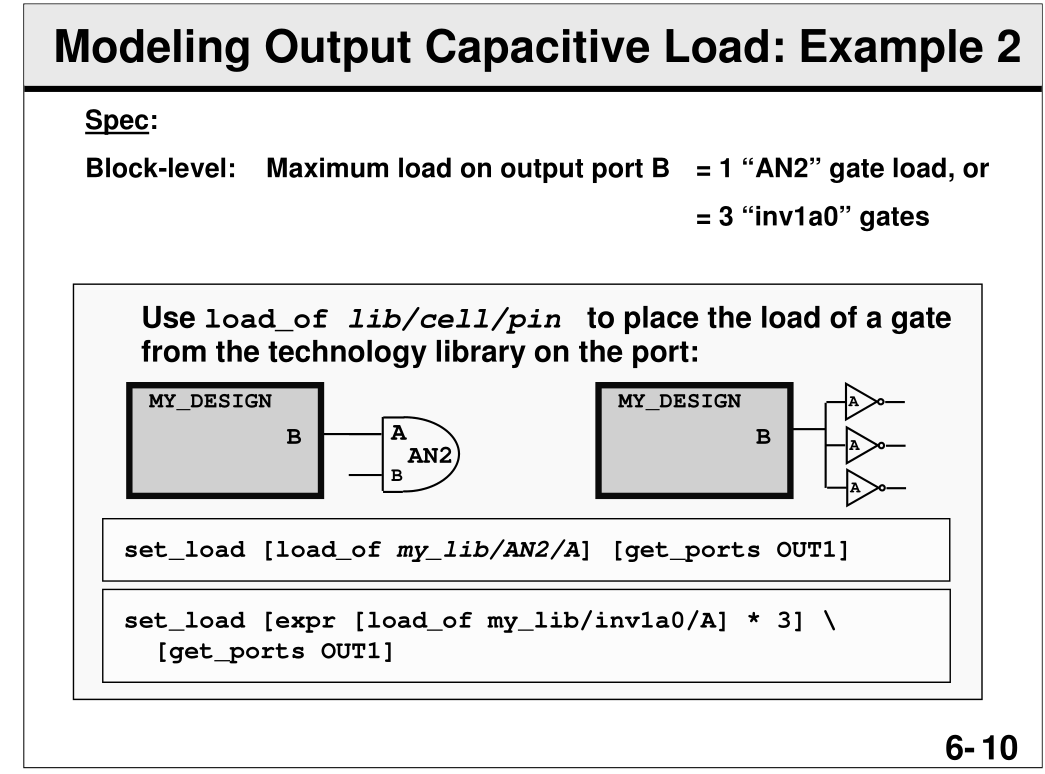
### check\_timing

用来检测当前的设计约束中，有没有路径没有约束好的，因为综合工具需要根据约束条件来选择器件，如果有路径没有约束到，那么综合这部分电路的时候，工具就不知道如何选择。在设置好约束之后，需要进行此项检查，确保约束的完整和正确性。

### set\_load

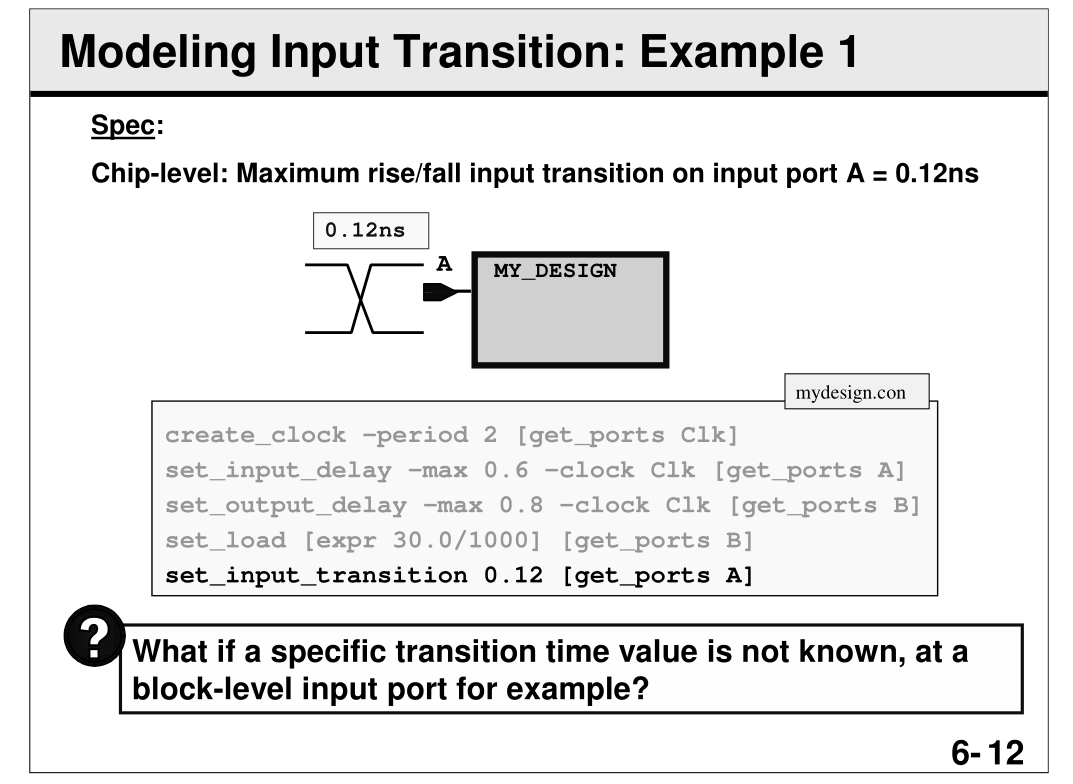
为输出设置负载，单位跟lib库中的单位相同，T28的是pf。如果是驱动门电路，但是不知道门电路的负载是多少，可以采用图2的方式，直接用库中cell的负载信息。

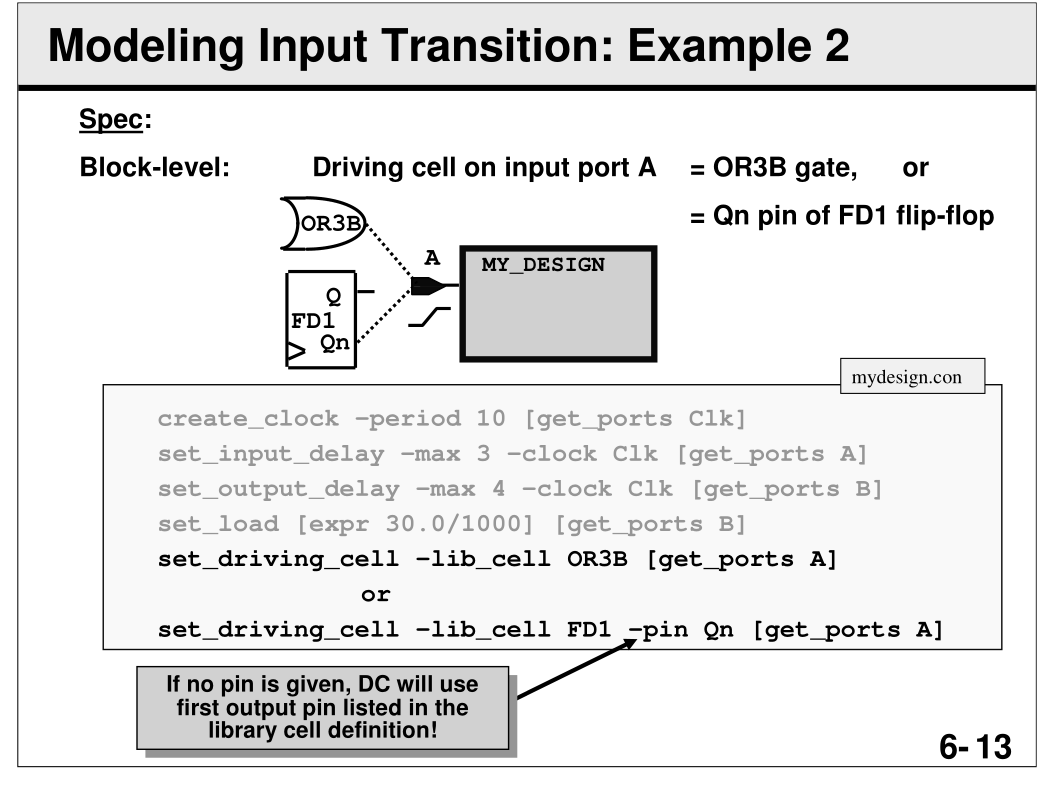




### set\_input\_transition

设置输入信号的传输延时，同样可以采用设置绝对值的方式，也可以采用库中的cell进行驱动，即set\_driving\_cell





### set\_max\_capacitance

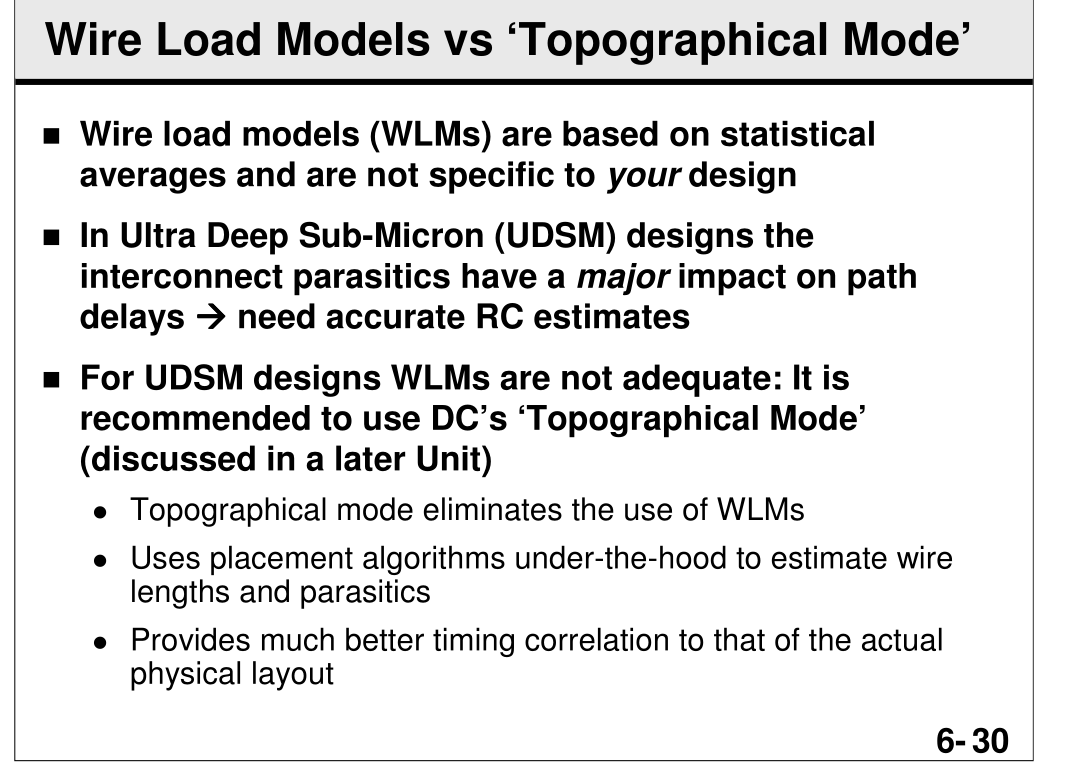
对指定的端口、设计或者时钟进行最大扇出约束，约束的单位与set\_load相同。

### set\_operating\_conditions

电路可能工作在不同的PVT条件下，因此在综合的时候一般选择比较差的条件，如果一个库中只有一种工作条件，可以不指定，否则的话需要指定哪种工作条件。比如采用get\_attribute my\_lib default\_operating\_conditions便可以得到库的工作条件。

### Wire Load Models（WLM）

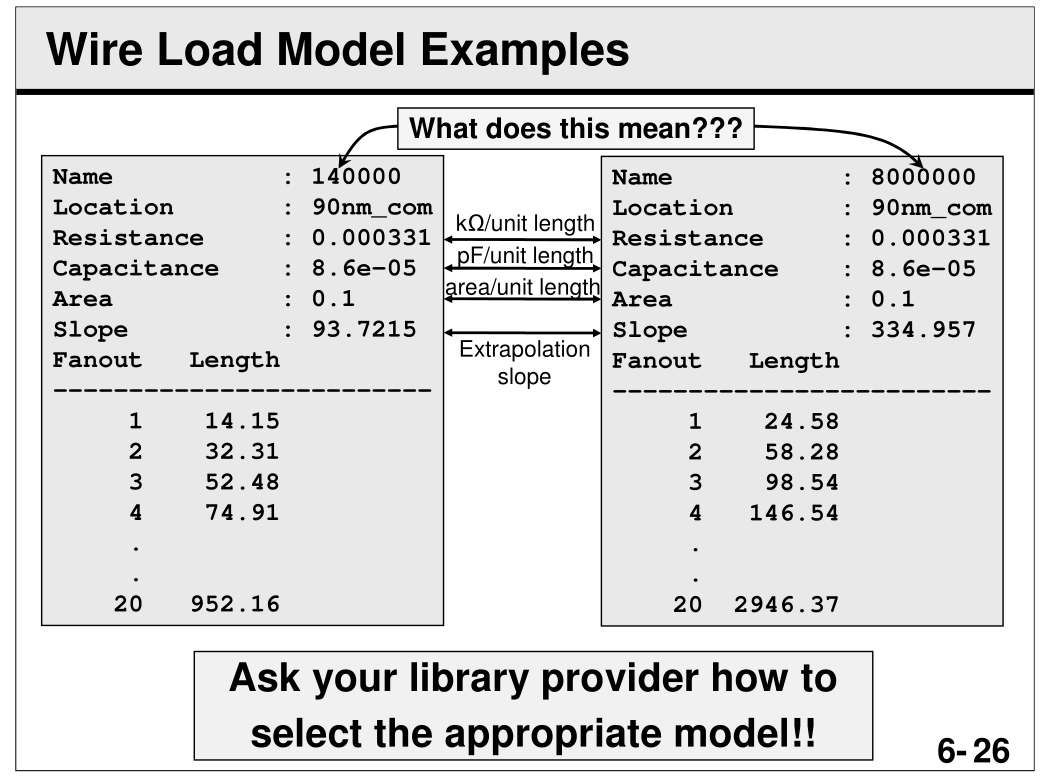
线负载模型

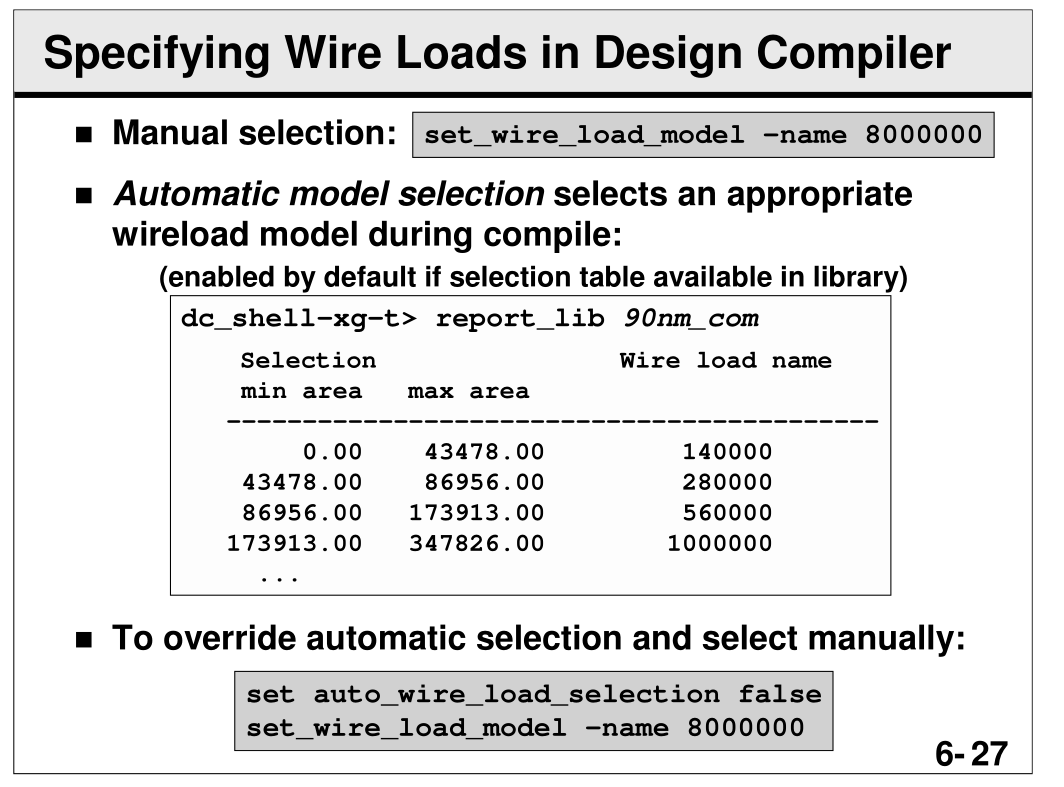


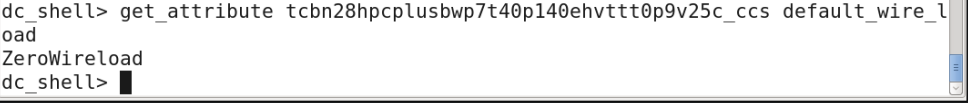
### set\_wire\_load\_model、auto\_wire\_load\_selection

用来设置一个模块内部连线的负载模型，自己指定模型的时候，需要将设置：set auto\_wire\_load\_selection False，在实验室中，一般不会自行设置，默认就行（默认为True），此时DC会自动选择线负载模型，此时的线延时为0，即只有cell的延时，线上的延时，是等做完PR之后，根据实际的情况进行评估的，在DC阶段模拟此种延时，感觉意义不是很大，可能是因为28nm影响不大？

可以采用get\_attribute <lib\_name> default\_wire\_load来获取线负载模型，图三中所示，表示库中的延时为0。







### set\_wire\_load\_mode

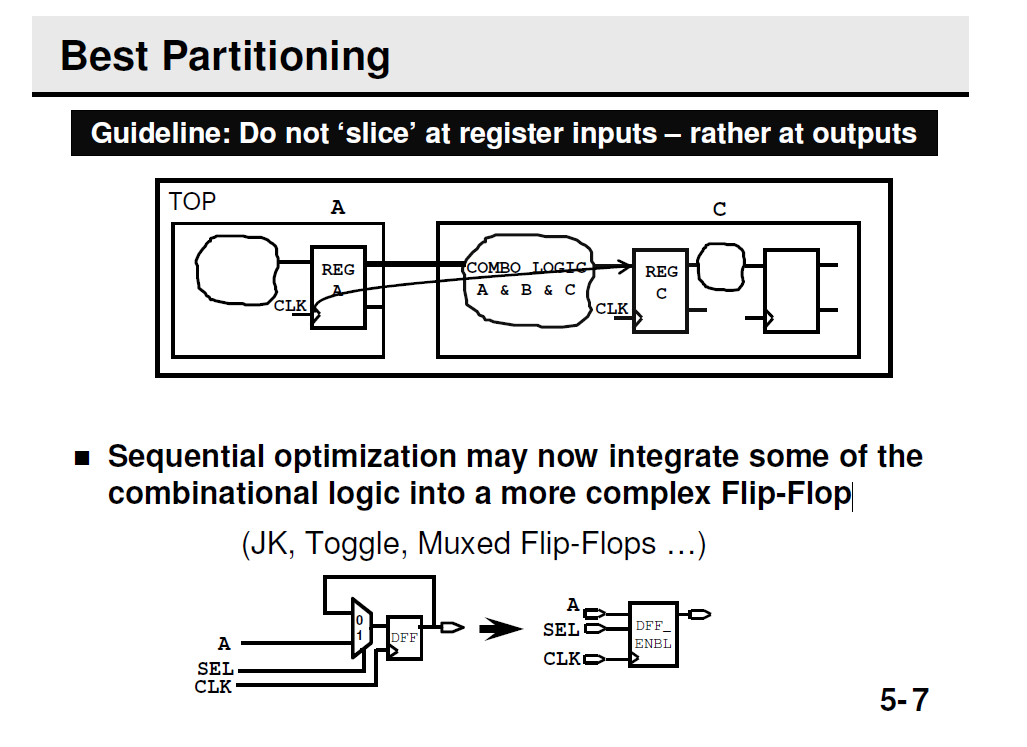
set\_wire\_load\_model用来设置一个模块内部连线的负载模型，如果连线连接的是不同的模块，那么它的负载模型又将怎么估计呢？这就要用到连线负载模式(set\_wire\_load\_mode)这个命令。连线负载模式一共有3种，围绕(enclosed)、顶层(top)以及分段（segmented）。如上图所示，一根连线连接了B2和B2两个模块，这两个模块都位于TOP下的SUB这个子模块中，围绕模式是指连接B1和B2的连线的负载模型用围绕它们的模块的负载模型代替，即用SUB的负载模型；顶层模式是指用顶层模块的负载模型代替；分段模式顾名思义，分别根据穿过的三段的模型相加得到。如果要设置成围绕模式，可以使用如下命令：set\_wire\_load\_mode enclosed

## 设计模块划分

将一个大的电路划分成不同部分。

推荐的划分方式：

* 将组合逻辑放在寄存器的输入端，让寄存器吸收一部分组合逻辑，综合出更加复杂的寄存器来降低组合逻辑的代价。



## compile\_ultra

DC中的优化命令，根据指定参数对电路进行优化，比compile更加强大。

参数：

* -no\_autoungroup：关闭DC对电路进行自动分组功能，如果不加，DC会自动分组。
* -no\_boundary\_optimization：关闭DC对电路进行边界优化的功能，如果不加，DC会自动优化。
* -timing\_high\_effort\_script：在DC中用来支持与现有脚本的向后兼容，但是当进行电路优化时，可能会被忽略，因此加上这条命令（另一条解释为，DC进行优化的时候，即便设置了，也可能被忽略）。
* -gate\_clock：打开clock\_gating的优化，即让DC能够自动插入clock\_gating模块。

## 库相关命令

### lc\_shell

进入library compiler。

### read\_lib

读入一个库

### report\_lib

报出库的信息，有些常用的参数，可以在dc\_shell中用man进行查阅。

### list\_lib

列出当前设计所读入的库。

## Attribute相关命令

对于库和设计，都会有相应的attribute，可以用相关的命令获取指定的信息，比如get\_attribute，report\_attribute。

## 编译相关命令

compile的二次编译，即在第一次综合结果不满足约束的时候，进行二次综合。



## DC的三个优化层次

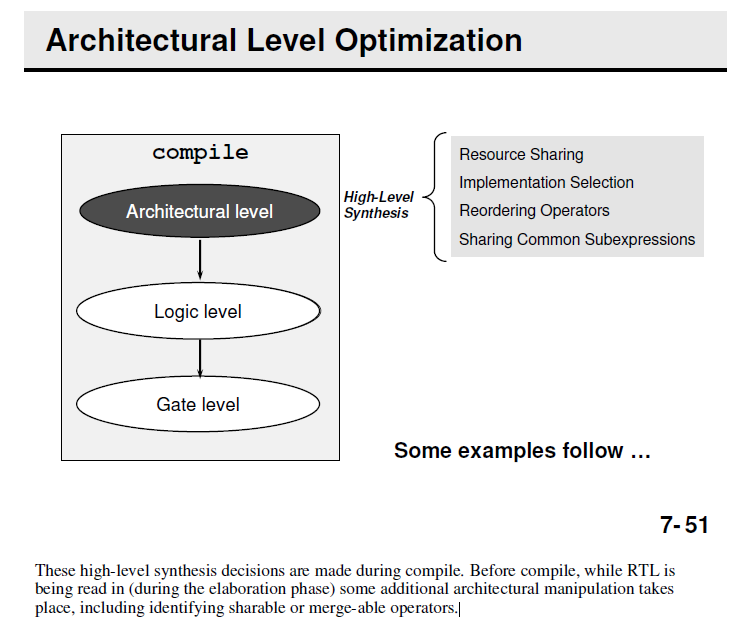


最高层次的综合为**架构级**：这些高级综合决策是在编译期间做出的。在编译之前，当RTL被读入(在细化阶段)时，会发生一些额外的体系结构操作，包括识别可共享或可合并的操作符。

比如将先加再选择，替换成先选择再加，类似于操作数隔离；选择加法的实现方式，是行波进位还是超前进位等。

主要包括：

* 计算电路复用
* 实现方式的选择
* 重新排序操作顺序
* 共享公共子表达式



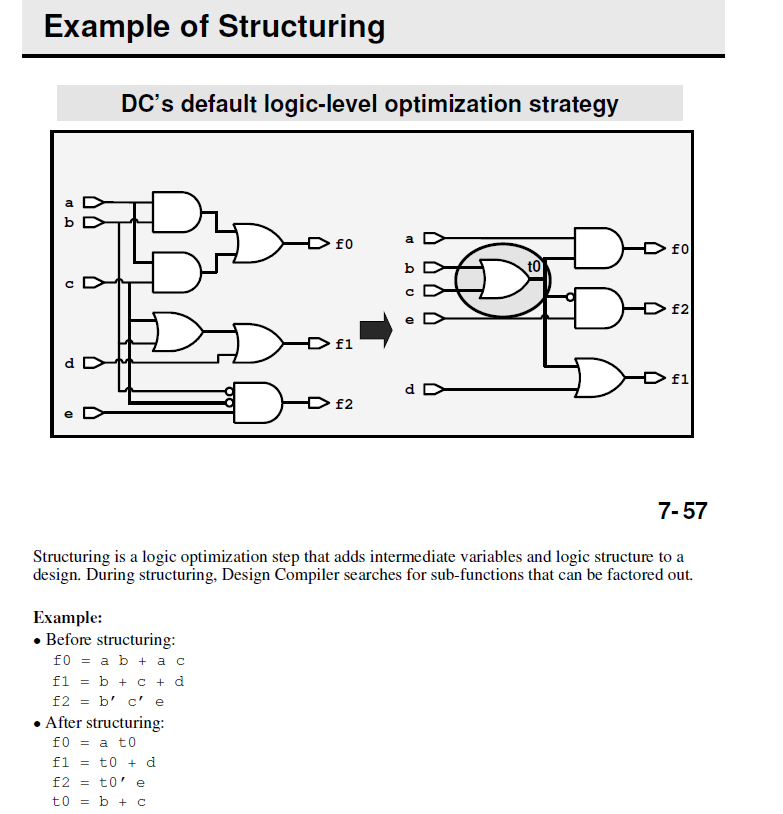
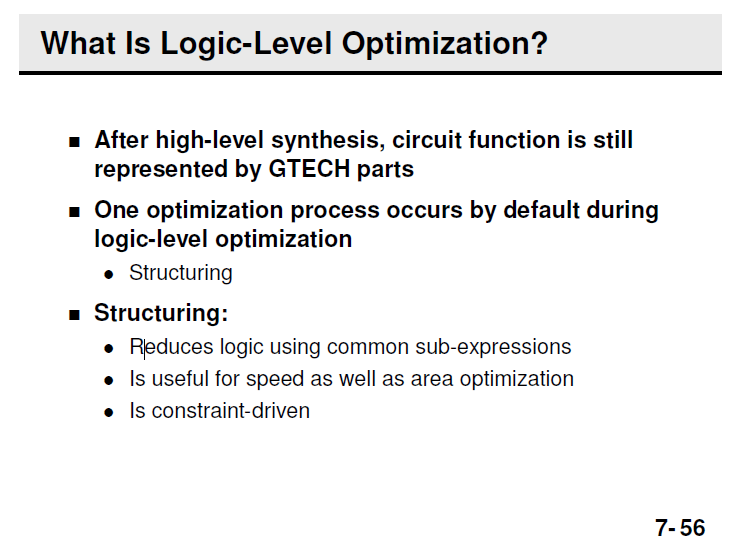
**逻辑级**为第二级优化：经过高级综合后，电路功能仍然由GTECH的形式表示，

主要包括：

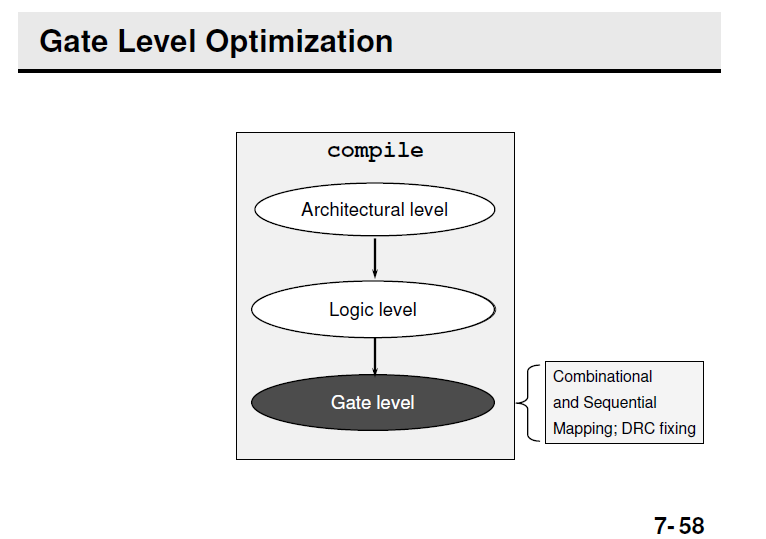
使用公共子表达式减少逻辑电路

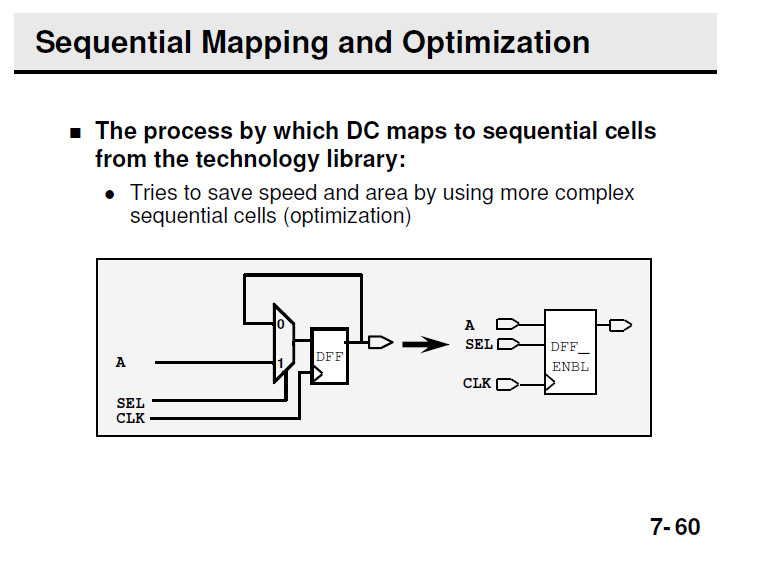
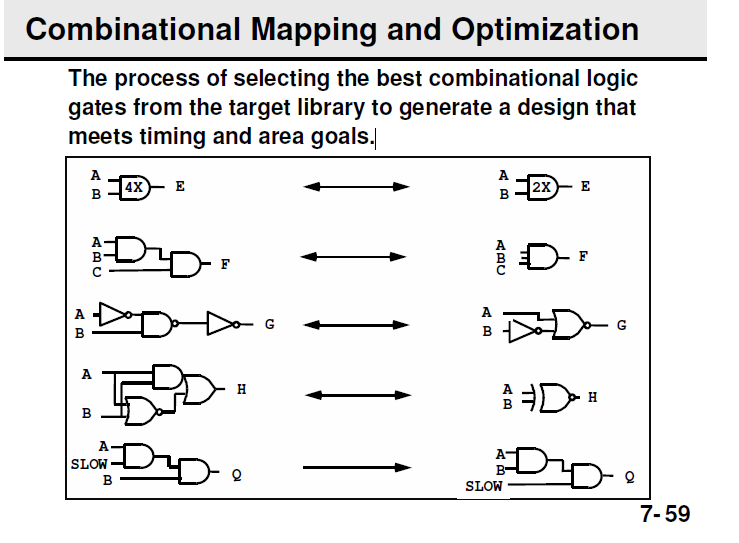
有益于延时和面积优化

是约束驱动的



门级优化是最后一级，也是优化力度最小的，主要是对某个极小的电路模块选择不同的器件来实现，主要还是约束和DRC驱动的。





## DFT（design-for-test）

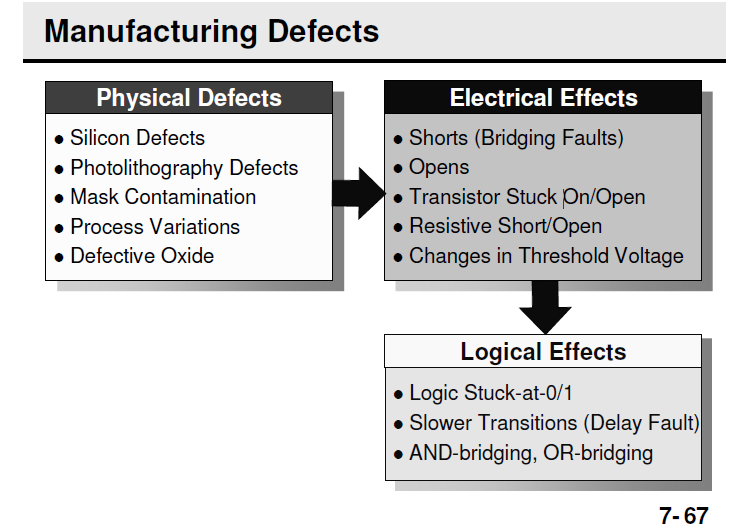
由于工艺制造过程中可能会产生缺陷，主要有：硅的缺陷，光刻缺陷，掩膜污染，制造过程的variations，有缺陷的氧化，会造成一系列的电学影响，主要有：短路(桥接故障)，开路，晶体管引脚开路，电阻短路/开路，阈值电压的变化，进一步造成逻辑错误，主要有：逻辑被固定在0/1，传输延时较大(延迟故障)，与型桥接、或型桥接。

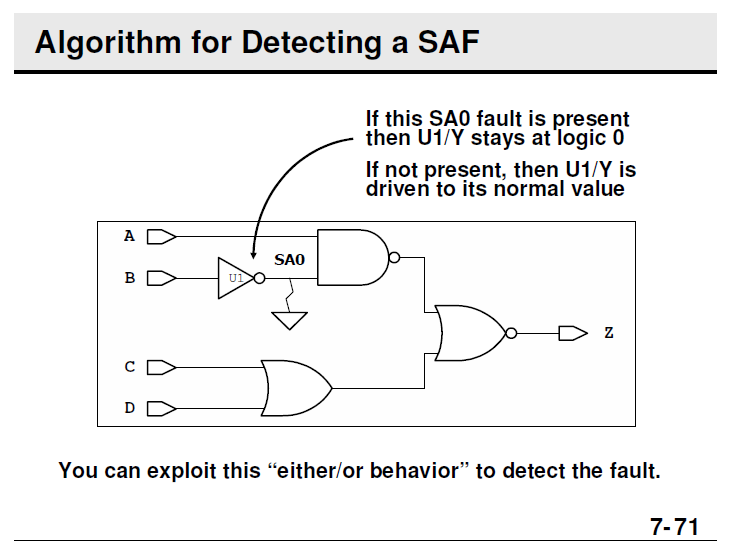
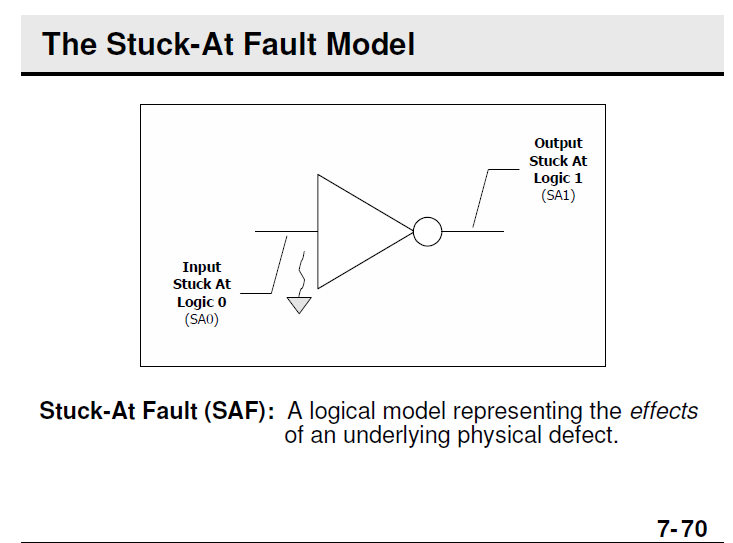
SAF（Stuck-At Fault）：指内部节点卡在一个固定的错误值。

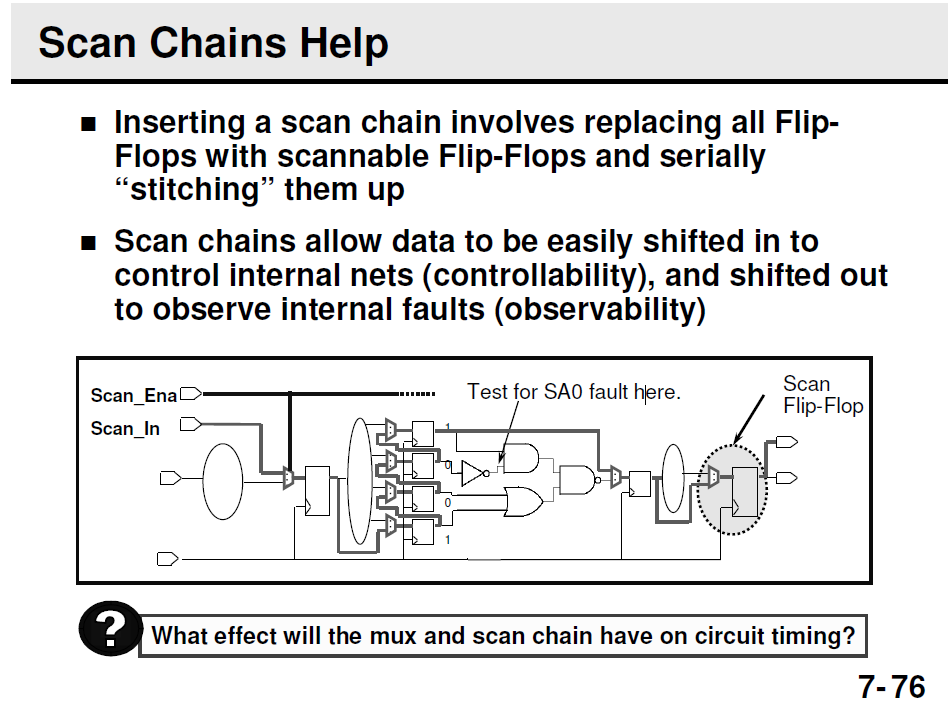
DFT的作用：在出货前发现制造缺陷并剔除，debug工艺制造过程，提高工艺收益。

主要目的：将故障影响从内部节点传播到主输出端口。

主要过程：将电路中的DFF替换成带扫描链的DFF，再将这些带扫描链的DFF连成一整个链。通过逐次移位的方式将不同的值送到不同的节点，观测输出来定位内部的问题节点，如图4。





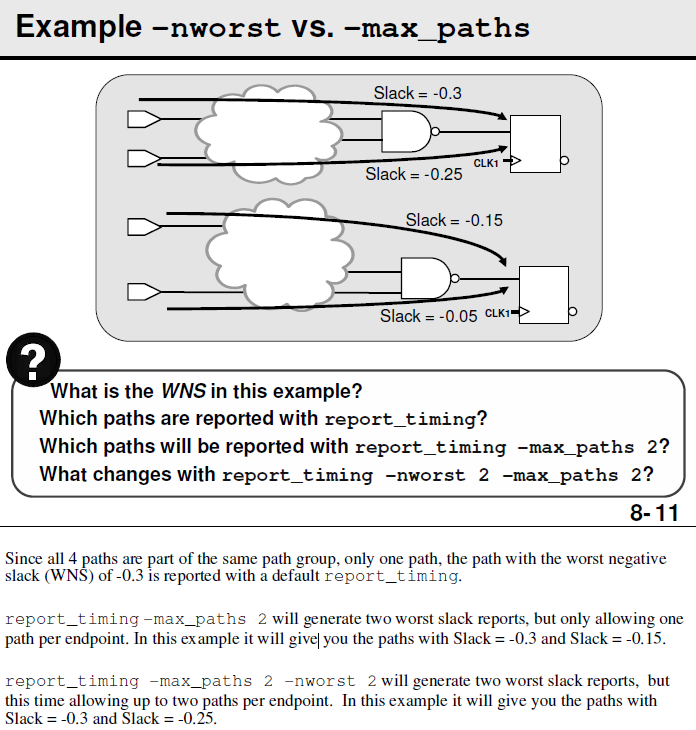


## report\_timing

默认报出的是每个路径组中最差的路径worst negative slack (WNS)，通过指定参数报出不同的时序信息。

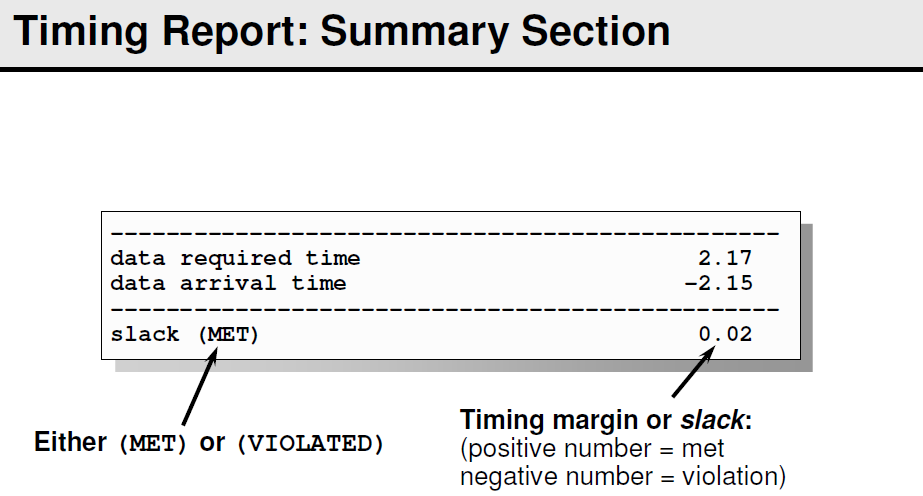
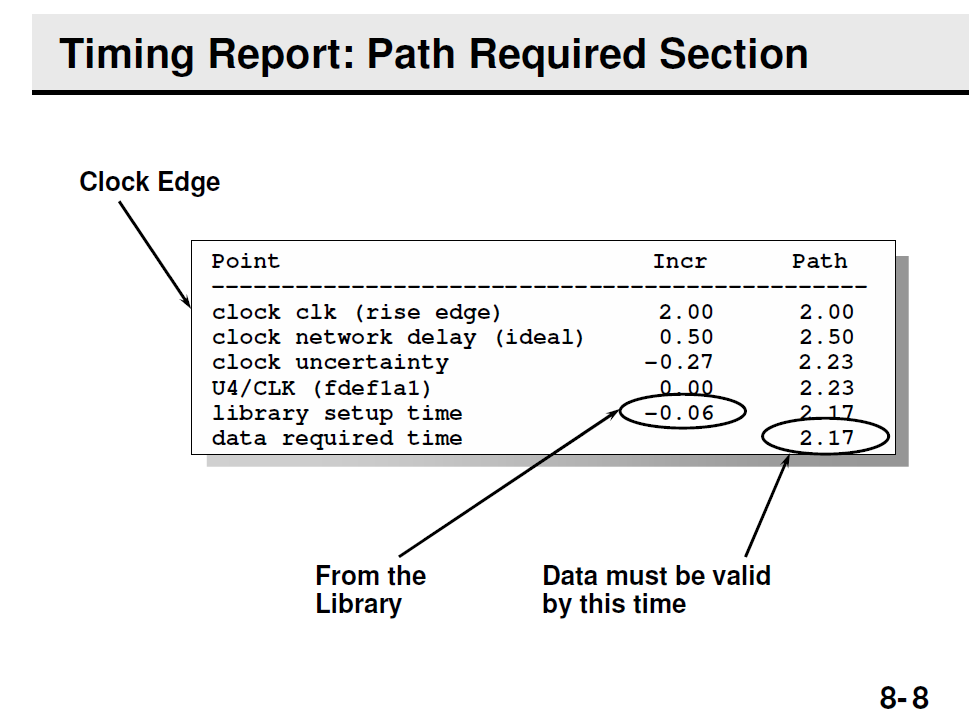
参数：

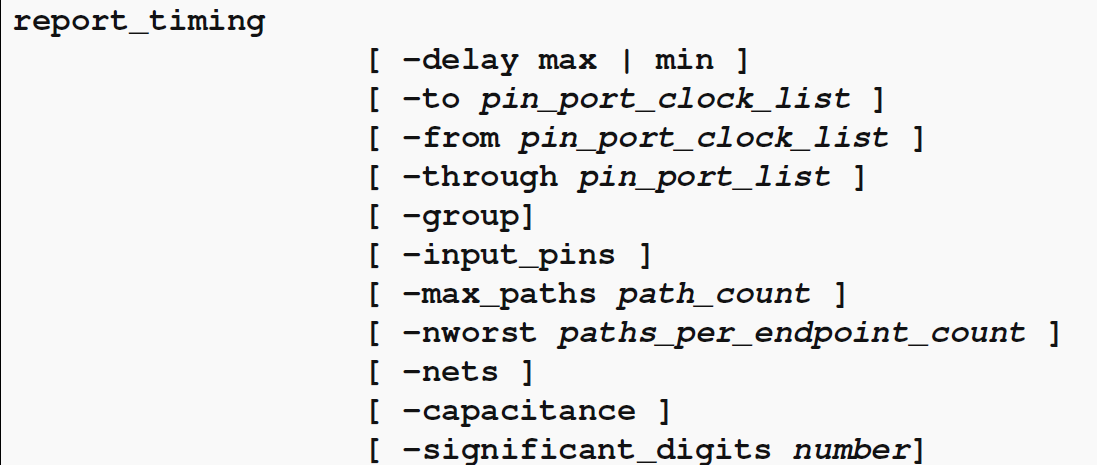
* -delay:有max和min两个选项，分别报出最长和最短的路径。
* -max\_paths：指定每个路径组报出的路径条数。
* -path\_type：指定时序路径的报出格式，信息最全的是full\_clock\_expanded。
* -sort\_by：对报出的路径信息进行排序，slack表示按照slack的大小进行排序，group表示按照组的顺序排序。
* -nosplit：当输出信息超过默认的列宽，不对输出信息进行分行，这样便于用脚本处理输出信息。
* -nworst：指定对于每个路径终点，报出的最差路径的数量，默认是1。
* -transition\_time：指出每个pin的传输时间，默认在报告中不报出。
* -attributes：报出在timing\_report\_attributes中指定的属性。



时序路径的信息如下。







## report\_timing\_requirements

报出时序是否达到需求。

report\_timing\_requirements # report timing\_requirements

[-attributes] (path timing attributes)

[-ignored] (ignored path timing attributes)

[-from <from\_list>] (from clocks, cells, pins, or ports)

[-through <through\_list>]

(list of path through points)

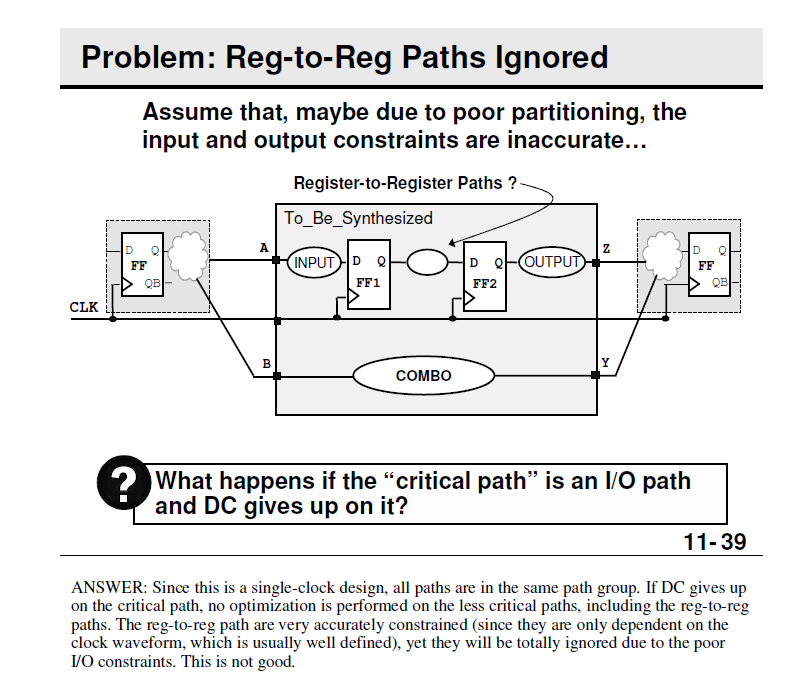
[-to <to\_list>] (to clocks, cells, pins, or ports)

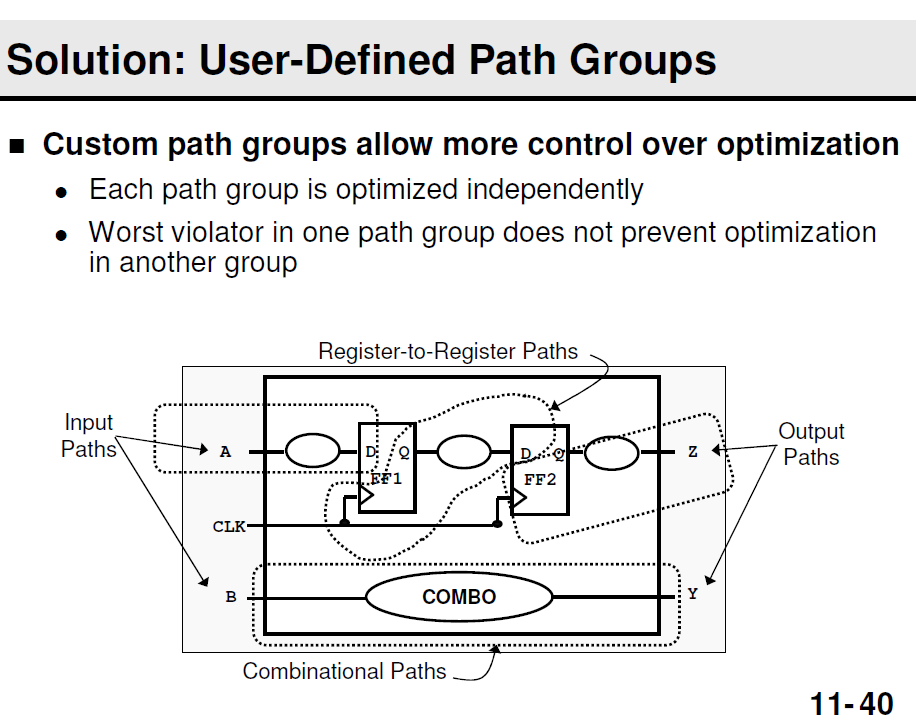
[-expanded] (report exceptions in expanded format)

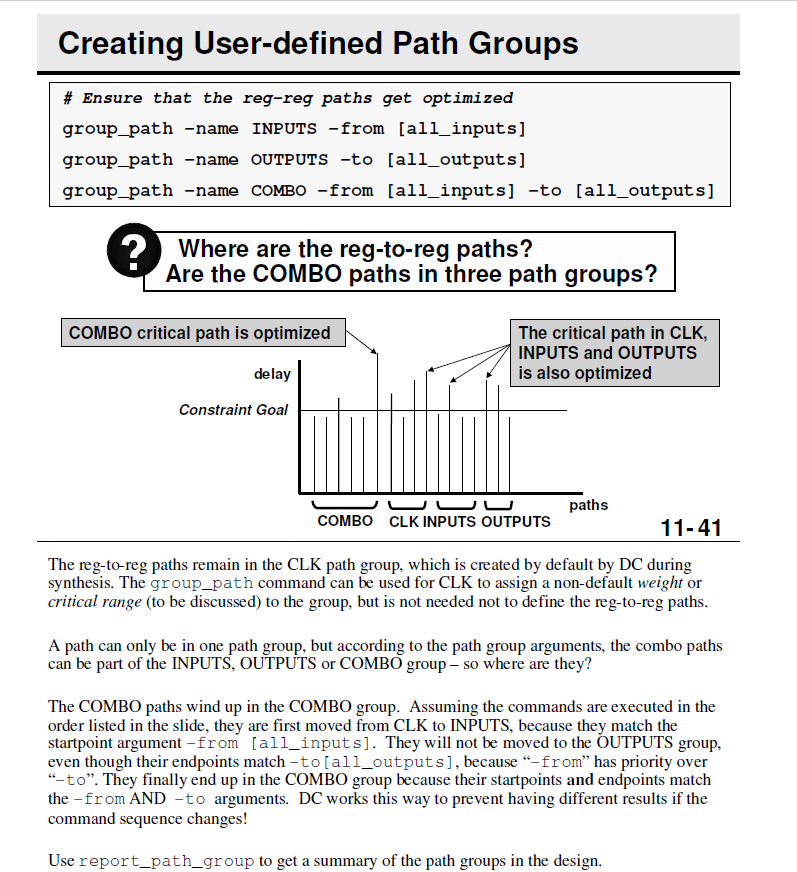
[-nosplit] (do not split lines when column fields overflow)

## group\_path

为不同的路径分配组，DC默认会给reg-to-reg的路径分配为clk组。

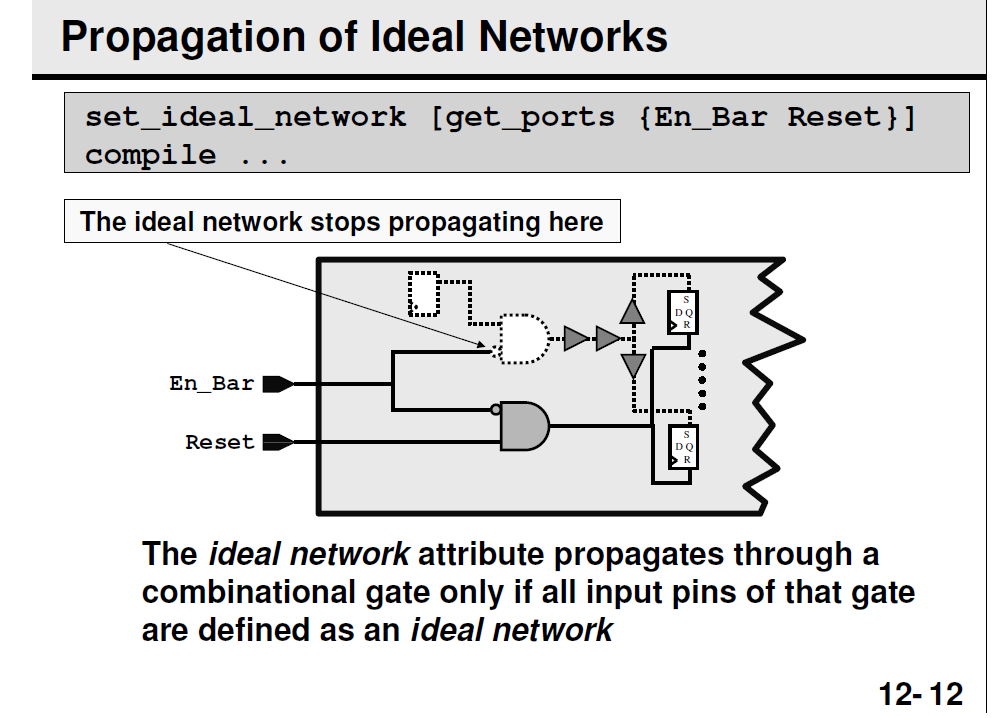
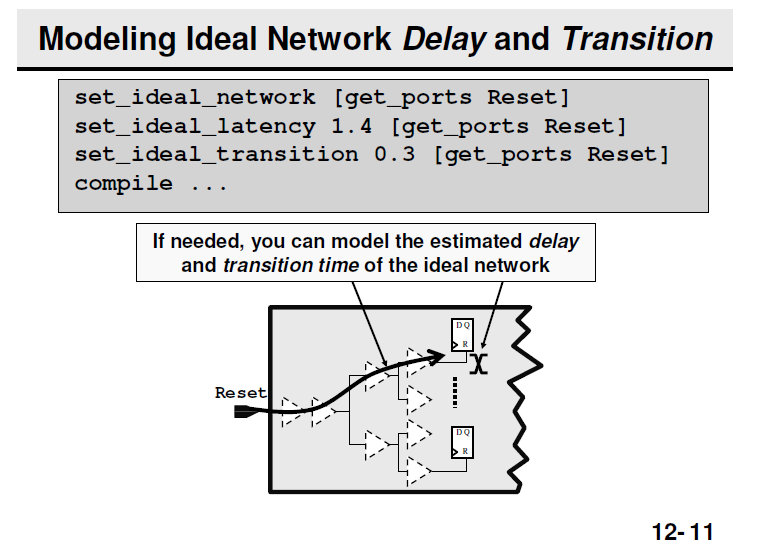






## set\_ideal\_network

设置理想的网络，–no\_propagate选项表示理想网络不能通过逻辑门传播，但它仍然通过层次结构传播。默认情况下，这个选项是关闭的。默认情况下，理想网络属性仅在组合逻辑门的所有输入引脚定义为理想网络时才通过组合门传播



## 对于网表中assign语句的处理

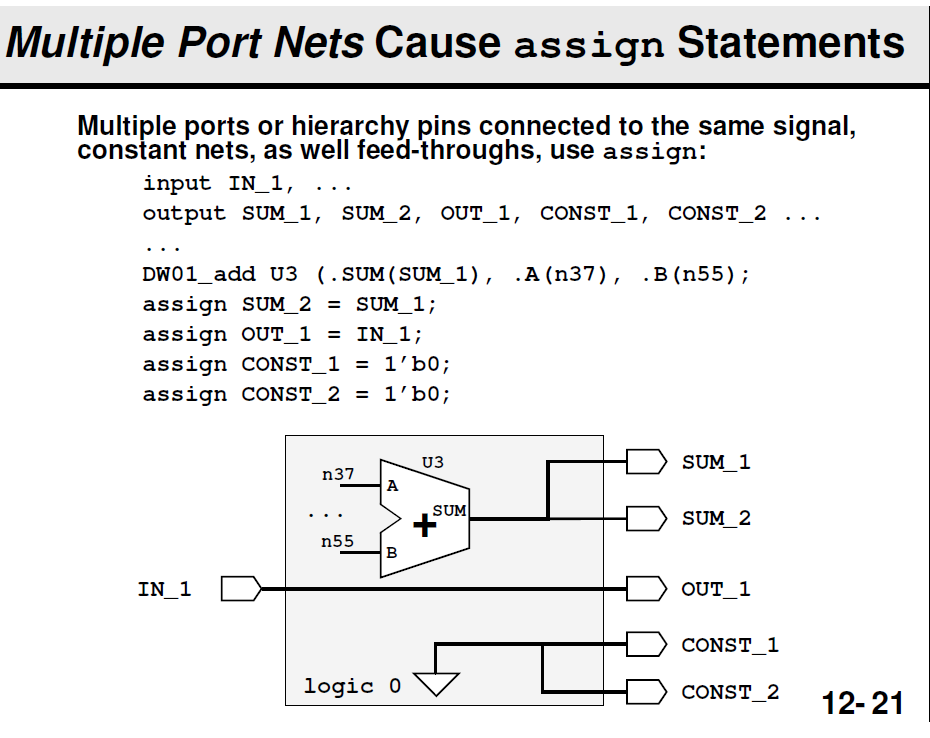
由于布局工具可能无法处理Verilog netlist中的assign语句，用命令阻止由于多端口线网型（Multiple port nets）和verilog中的三态声明（Verilog tri declarations），造成网表中出现assign语句。

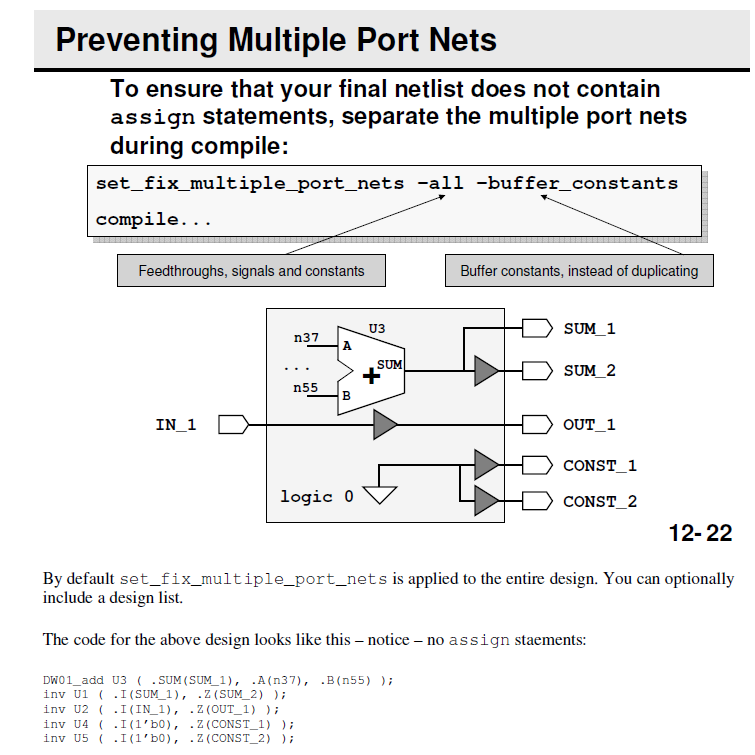
### set\_fix\_multiple\_port\_nets

当一个端口被多个输出端口引出、将端口赋值为常量，或者输入直接接到输出的时候，会出现assign赋值语句，此时可以采用

set\_fix\_multiple\_port\_nets –all –buffer\_constants 命令，-all包括-feedthroughs –outputs –constants三个的功能

强制在上述端口之前插入buffer，从而避免assign语句的产生，此命令默认应用于整个design，也可以指定一个设计的列表，如图2（必须在compile设计之前应用，因为修复发生在编译期间）。

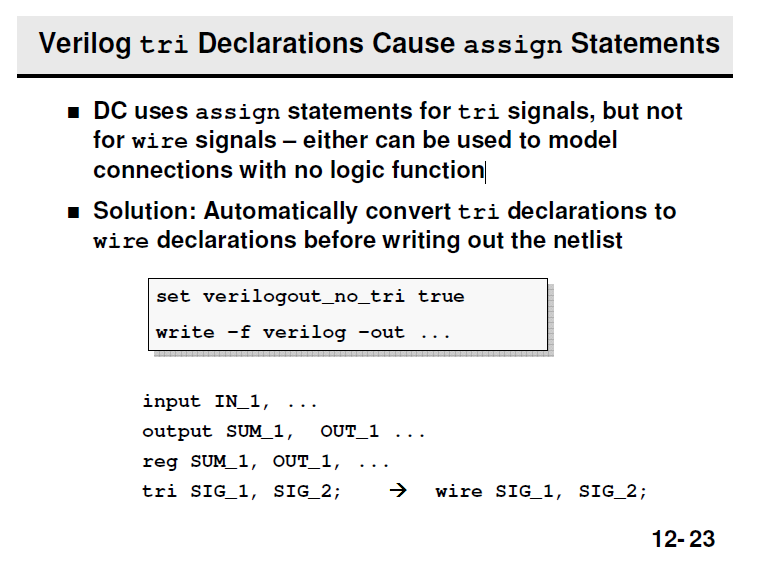




### verilogout\_no\_tri，****verilogout\_equation****

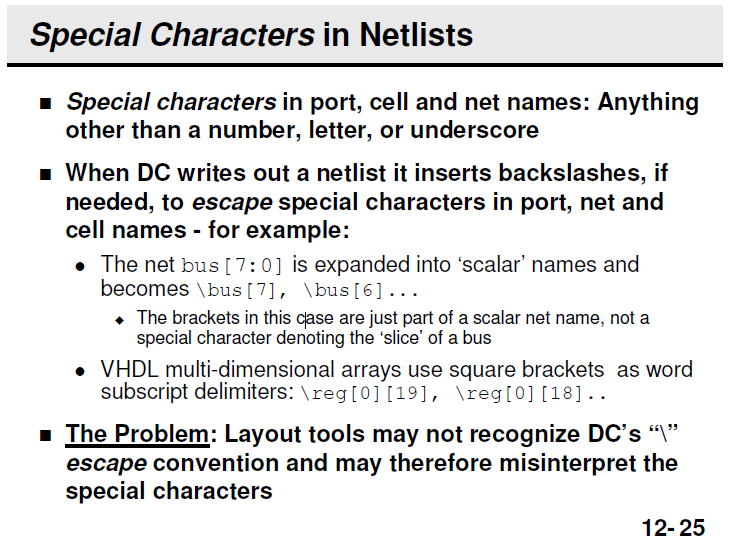
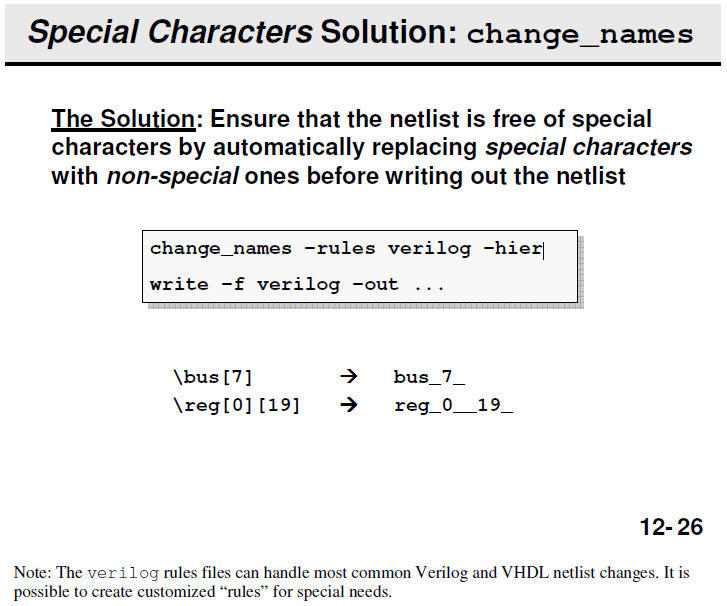
在verilog中，对于三态（tri）的声明也会产生assign语句，因为DC对tri信号使用赋值语句，但对线网信号不使用赋值语句——两者都可以用来建模没有逻辑功能的连接。因此采用set verilogout\_no\_tri true 来强制将tri型变量转换为wire型变量（应该在所有优化之后使用，在写出netlist和/或约束之前应用）。

**还需要 set verilogout\_equation false 将检查变量设置为false，这两条命令是一般连着使用的，猜测是为了屏蔽前一条命令带来的输出前后内容一致性检查不通过问题。**



## 对于网表中特殊字符的处理：change\_names

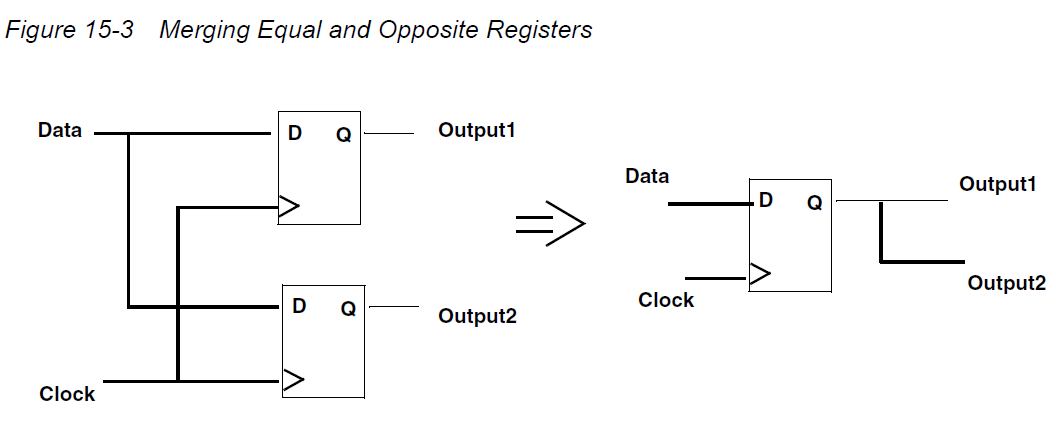
对于ports、cells和nets的名字，布局布线工具只能认识数字、字母或下划线，解析不了其他特殊字符。但是当DC写出一个网表时，它会插入反斜杠(如果需要的话)来转义port、net和cell名称中的特殊字符，如图1所示。采用change\_names –rules verilog –hier命令，将特殊字符转换为下划线，如图2所示（应该在所有优化之后，在写出netlist和/或约束之前应用）。

## 综合相关的设置命令

### compile\_enable\_register\_merging

DC识别并合并相同或者相反的寄存器，设置为true的时候，表示允许识别并合并，设置为false的时候，表示禁止。



### compile\_seqmap\_propagate\_constants

控制DC是否识别和删除常量寄存器，并在整个设计中传播常量值。当该值为true(默认值)时，compile尝试识别并删除设计中的常量序列元素，一般设置为true。

### mv\_default\_level\_shifter\_voltage\_range\_infinity

当这个命令设置为true的时候，猜测是将level shifter的电压转换范围设置为任意值。

### compile\_timing\_high\_effort

使能DC进行高度的时序优化，一般在高性能电路中会设置为true，但在KWS中不需要。

### change\_names\_dont\_change\_bus\_members

控制change\_names命令修改总线成员名称的方式，默认为false。当设置为true的时候，不对总线名称进行修改。

### compile\_disable\_hierarchical\_inverter\_opt

控制编译命令中边界优化特性的行为。默认情况下，边界优化尝试跨层次移动反相器，以提高设计的优化。然而，设置为true，边界优化将不会跨越层次边界移动反转器，即使这样做可以改善设计。

### auto\_insert\_level\_shifters\_on\_clocks

自动将level\_shifter插入到指定的时钟上，当指定-all参数的时候，会自动在需要level\_shifter的寄存器上插入。

### auto\_insert\_level\_shifters

当设置为true的时候，自动插入level\_shifter。

### timing\_enable\_multiple\_clocks\_per\_reg

设置为true的时候，对到达单个寄存器的多个时钟进行分析。

### enable\_page\_mode

设置为true的时候，控制长报表是否在一个页面上显示(类似于UNIX More命令)，实验室设置为false。

### sh\_enable\_page\_mode

设置为true的时候，控制长报表是否在一个页面上显示(类似于UNIX More命令)，实验室设置为false。

### compile\_seqmap\_identify\_shift\_registers在编译扫描中控制移位寄存器的识别，此功能只支持在DesignCompilerUltra中使用多路扫描样式进行测试准备的编译，默认值为true，在没有插入dft的时候，设置为false。

### compile\_seqmap\_identify\_shift\_registers\_with\_synchronous\_logic

控制是否识别包含寄存器之间的同步逻辑的移位寄存器。当设置为true的时候，启用移位寄存器标识时，此变量仅对dc超优化有影响，主要用在dft中，实验室设置为false。

### set\_clock\_gating\_style

为插入和替换的clock\_gate的进行设置。

参数设置：

* -sequential\_cell：选择clock\_gating的实现单元，如果指定latch，表示使用带有latch的CG，如果为none，则采用不带latch的CG，可能采用与门、或门，但是与门、或门可能会存在毛刺，与门做CG，时钟使能为1时钟穿透AND，posedge trigger存在毛刺；或门做CG，时钟使能为0时钟穿透OR，negesge trigger存在毛刺；如果使用集成的CG，那么这个参数就不需要指定了。
* -setup：为2输入clock gate的使能信号（en）指定setup time约束；
* -hold：为2输入clock gate的使能信号（en）指定hold time约束；
* -minimum\_bitwidth minsize\_value，为了节省gating cell的数量，需达到一定寄存器数量的register bank才使用gating cell；
* -positive\_edge\_logic、-negative\_edge\_logic {cell\_list | integrated [active\_low\_enable] [invert\_gclk] }，{cell\_list}用于指定“上升沿”触发使用的CG单元，需要注意的是-sequential\_cell指定的是否有latch应该和cell\_list的对应电路是否含有latch保持一致，如：

set\_clock\_gating\_style -sequential\_cell latch -positive\_edge\_logic {latch and}

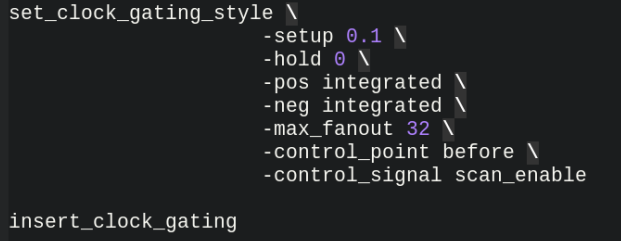
set\_clock\_gating\_style -sequential\_cell none -positive\_edge\_logic {or}

{integrated}用于表明使用上升沿触发的ICG单元。

一般都是使用集成的CG，即ICG，此时只需要进行如下设置:

-pos integrated –neg integrated, -sequential\_cell不需要进行设置。

* -control\_point none | before | after 对于DFT测试电路，为了满足电路可控，需要引入TE信号来控制latch的使能端。before就是在Latch之前插入或门，将TE信号和时钟使能信号或起来然后连接到Latch的D端。同理，after就是在Latch之后插入或门，将TE信号和Latch的Q端或起来然后连接到IGG的AND门。
* -max\_fanout：指定一个ICG单元能gating的寄存器数量。



### insert\_clock\_gating

在读入的gtech网表中，插入ICG。

# PT笔记

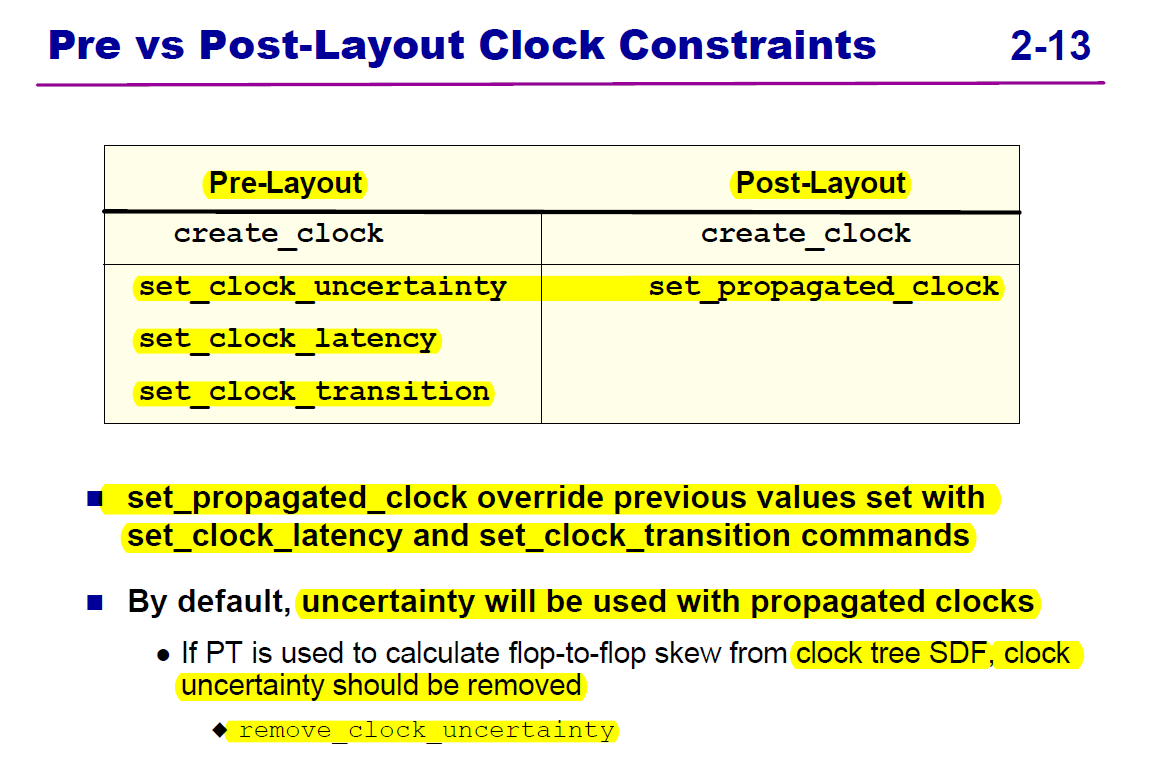
## link\_library.txt

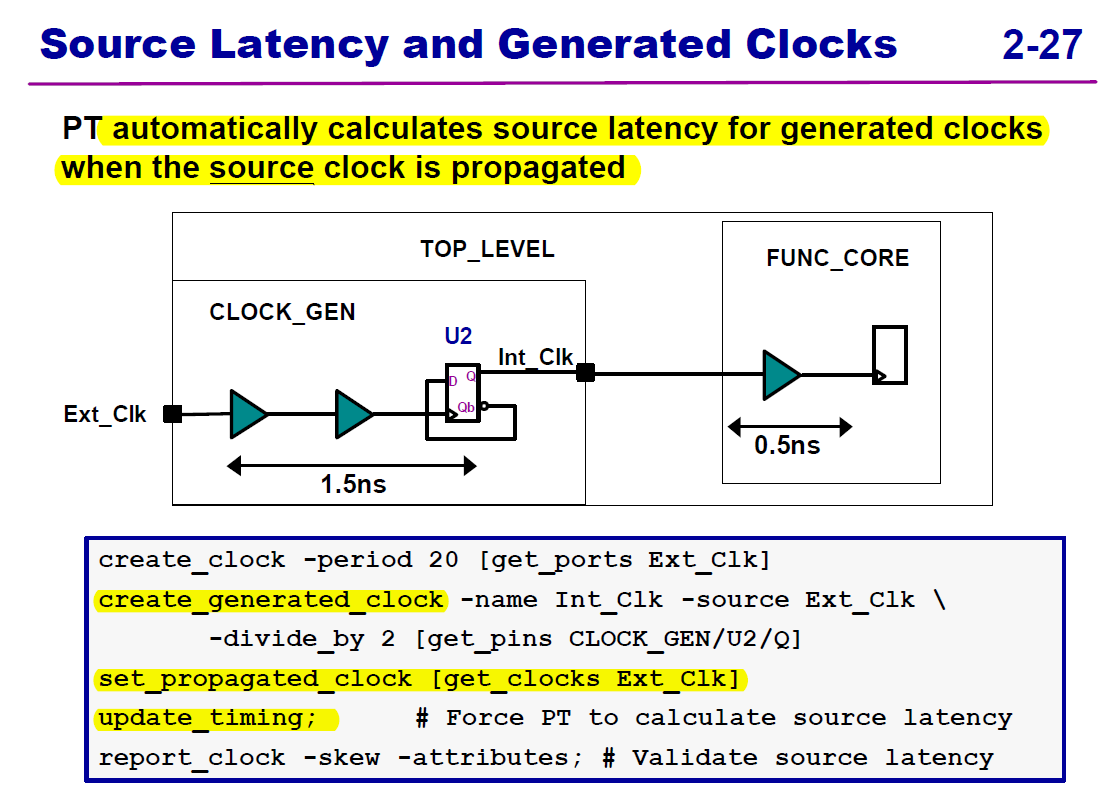
这是DC的输出文件之一，里面设置了link\_path，这个文件是给PT用的，用来指定综合网表中的reference，link\_path命令设置整个设计的reference库，如果对其中某几个单元采用单独的库，可以采用link\_path\_per\_instance，指定某一个或几个模块的库。这里需要注意的是，在综合之前采用set\_target\_library\_subset命令对一个模块指定一个库，但是在link\_path\_per\_instance中可能会出现多个模块对应多个库的情况，这是因为综合之前的是RTL级的代码，不同模块的reference可能是相同的，但是在综合之后，由于设置了set\_target\_library\_subset，因此不会出现这种情况了，采用多个库对应多个模块可以减少代码量，也可以一个模块一个模块进行指定。

## set\_propagated\_clock，create\_ generated\_clock

create\_generated\_clock是用来生成内部时钟，比如二分频时钟。set\_propagated\_clock用来在post\_layout（post\_CTS）阶段，让DC自动计算network latency，因为此时已经做过CTS，模块的时钟端到每个寄存器的时钟端的延时已经能够计算出来，因此需要计算source\_latency，方式就是通过set\_propagated\_clock。

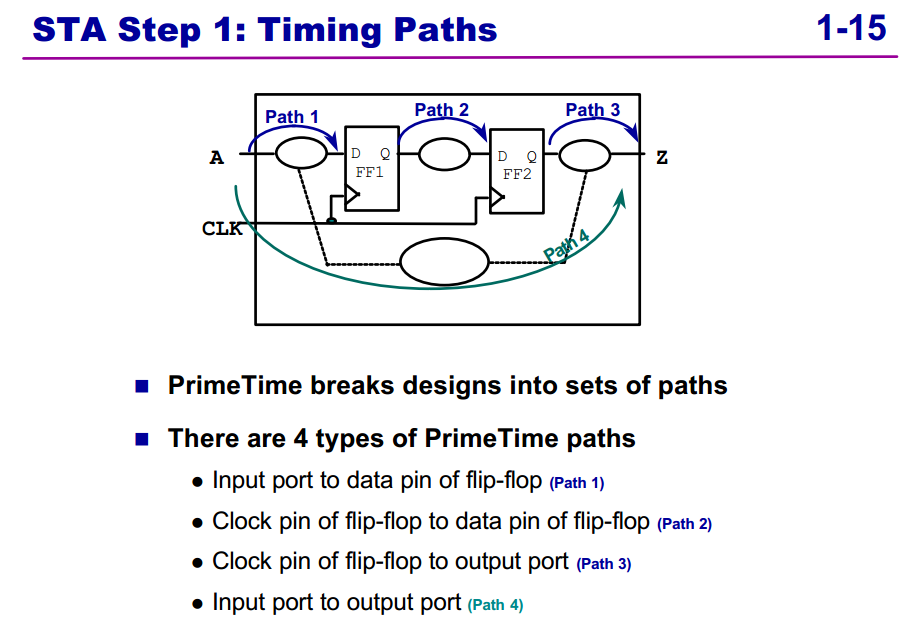
但是在pre\_layout（pre\_CTS）之前，network latency还无法计算。在实验室的脚本中，在CTS之前，也会使用set\_propagated\_clock，可能是因为存在generated的时钟，generated时钟的主时钟必须为PrimeTime进行propagated，以计算生成的时钟的源延迟。否则，PrimeTime将不会计算此源延迟。

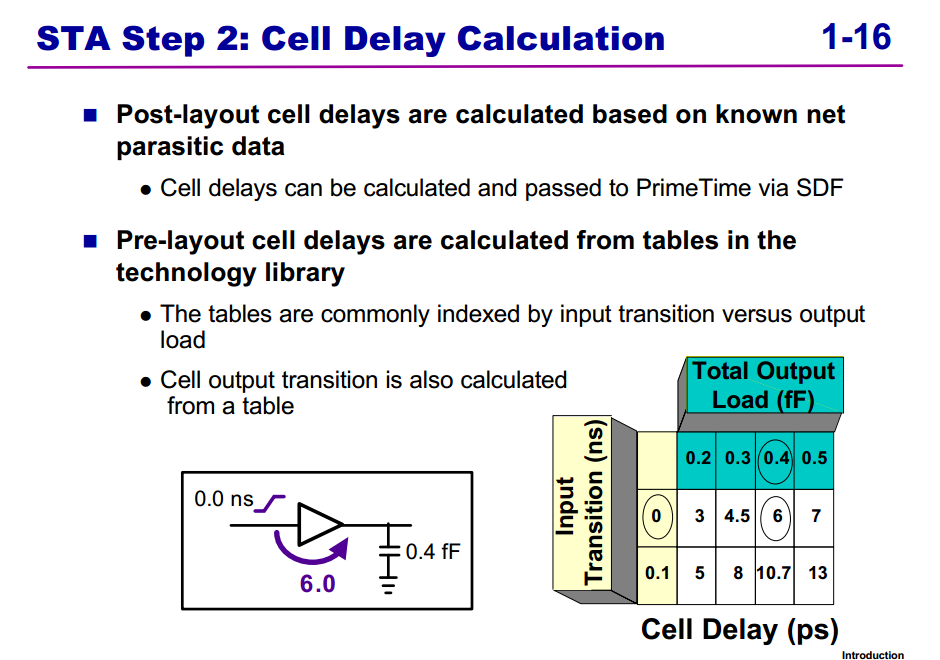


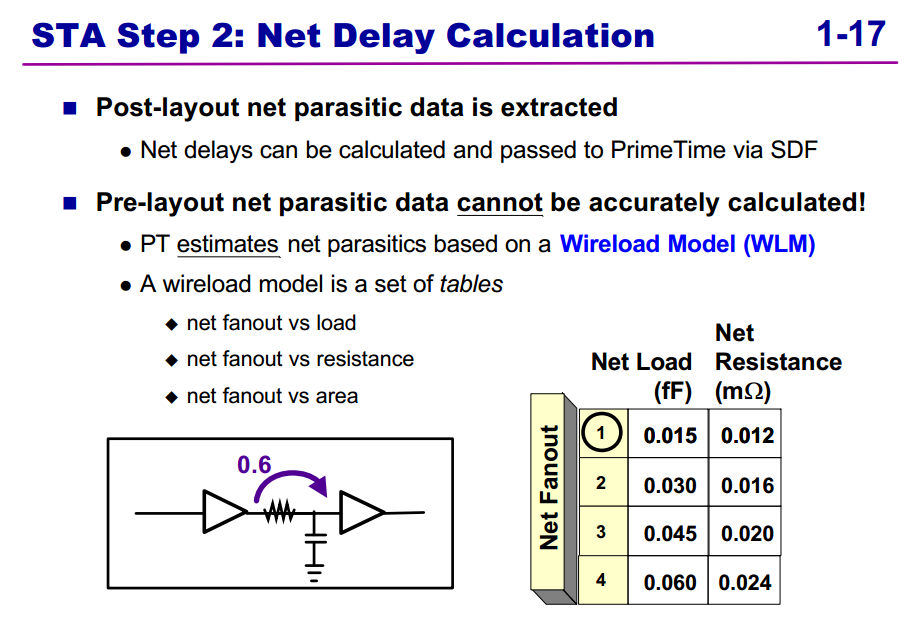


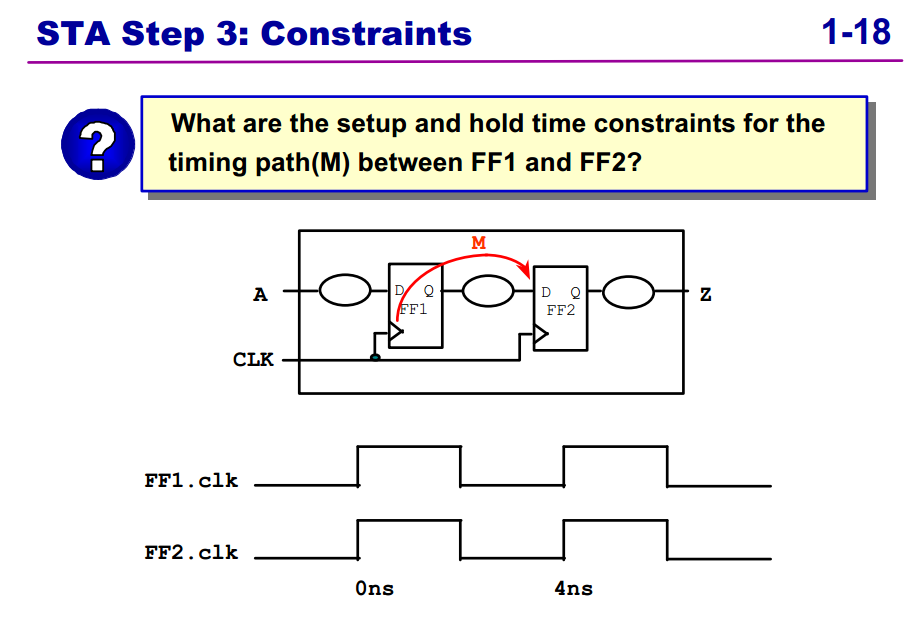
## SDF和SPEF在DC、PT、PTPX中的用法

### PT







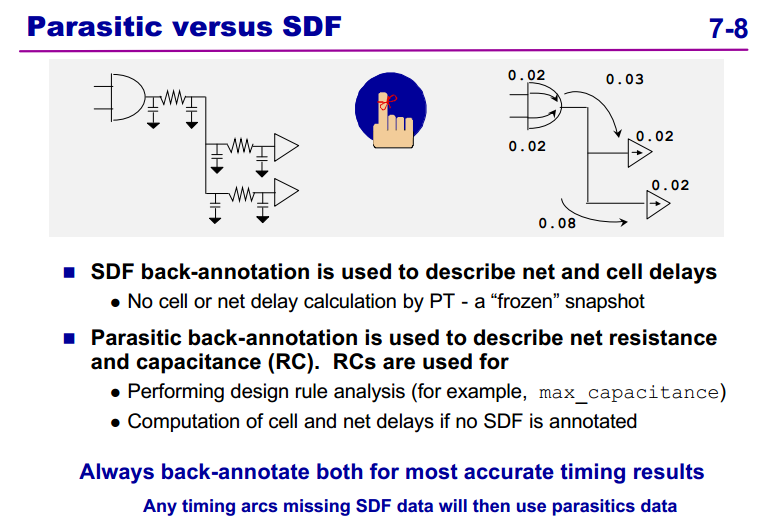


PT在做STA时先对设计做timing path的分解，然后对每一条path，计算cell delay和net delay，最后再进行check看timing是否满足。在Pre-layout阶段，cell dalay的计算是基于lib中的查找表，由input transition和output load决定，对于net delay，PT利用Wireload Model估计连线寄生（R和C与fanout相关）。在Post-layout阶段，基于已知的连线寄生数据，cell delay和net delay可以被计算出来并通过SDF传递到PT（但是不建议使用布图工具生成完整的SDF反标到PT，而是仅从版图数据库提取连线RC延迟和集总寄生电容反标到PT，这样PT可以使用自己的延迟计算器依据反标注的互连RC和容性连线负载来计算cell dealy）。

总结一下，为使用PT进行STA，可以生成如下信息类型：

1. SDF格式的连线RC延迟；
2. set\_load格式的容性连线负载值；
3. DSPF（Detailed Standard Parasitic Format）、RSPF（Reduced Standard Parasitic Format）、SPEF（Standard Parasitic Exchange Format，IEEE standard）文件格式的时钟及其他关键连线的寄生信息。

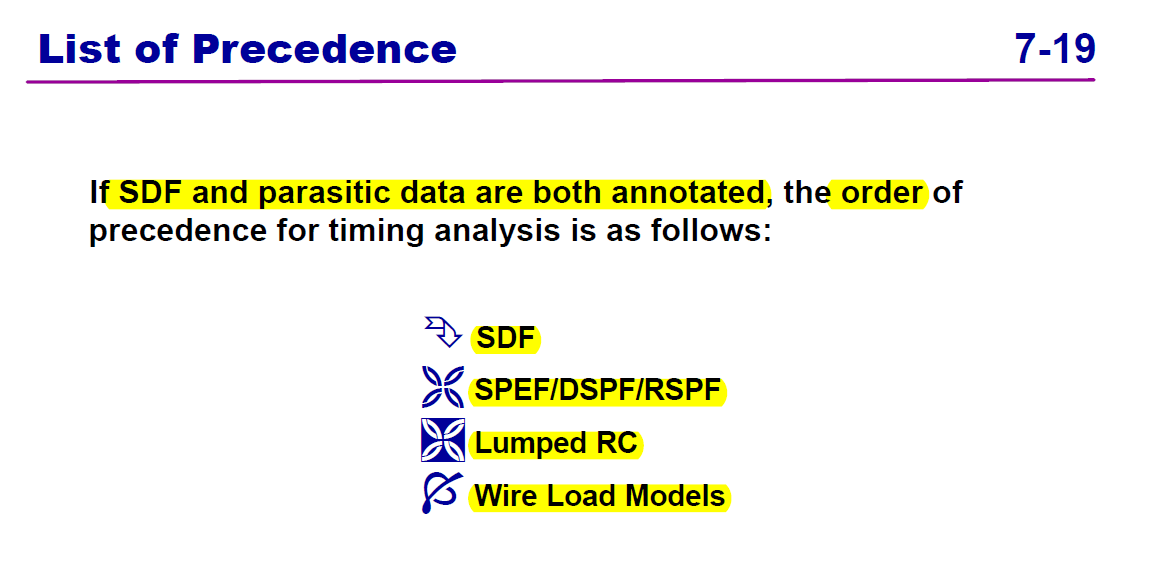
可以结合下图理解parasitic和SDF的关系：



如果将SDF反标到PT中，那PT将不会计算cell和net delay，处于“frozen”状态，而在没有SDF反标的情况下，parasitic反标就被PT用来计算cell和net delay。

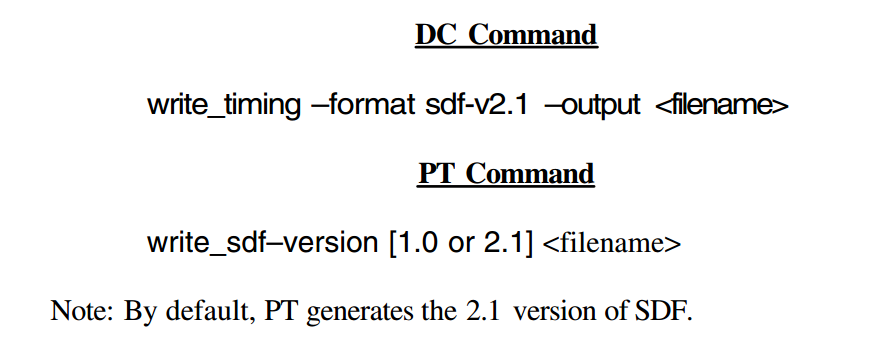
read\_sdf

read\_parasitic

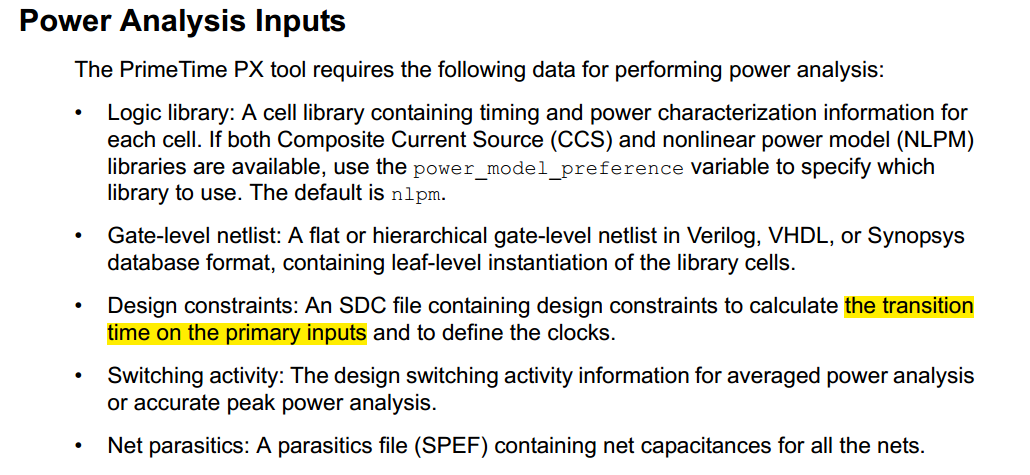


### DC

与PT类似，但是DC仅能读取SDF和set load文件格式，不能读取DSPF、RSPF和SPEF。



### PTPX



首先可以看看PTPX所需的输入文件，logic库、门级网表、设计约束、开关活动和连线寄生。其中，SDC用于计算primary inputs的transition time，并且定义时钟，SPEF包含了所有连线的电容信息。

所以PTPX的流程可以归纳为：

1. Pre-layout

PT生成SDF，将此SDF反标到VCS，生成VCD，将VCD输入到PTPX中进行功耗分析。

1. Post-layout

将布图工具生成的带时钟树信息的门级网表、SDF格式的连线RC延迟和set\_load格式的容性连线负载值输入给PT，生成SDF，将此SDF反标到VCS，生成VCD，将此VCD以及SPEF输入到PTPX中进行功耗分析。



