**DC综合简单总结**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*clk 需要门电路做gating 的意思\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

就想当于给clk做一个enable的功能，要用的时候有clk，不用的时候就没有

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*set\_dont\_touch和set\_dont\_touch\_network\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**？**  
在综合的过程中，为了不让DC工具自动优化一些我们不希望其优化的模块（比如CLK）我们通常都会设置set\_ideal\_network和set\_dont\_touch，我理解为前者在timing\_report的时候忽略延迟，后者阻止DC插入buffer。  
那么dont\_touch的属性，能不能穿过logic?   
set\_dont\_touch和set\_dont\_touch\_network有什么区别，为什么普遍认为set\_dont\_touch\_network会造成未知问题而不推荐使用？  
对于ideal\_net的这个属性，是不是说直接忽略了延迟，忽略了DRC，如果将clk设定为ideal\_net，是不是就不用再设定dont\_touch了？  
说的有点混乱，请各位大神技术支持，小弟不胜感激！！

=  
set\_dont\_touch不会穿过logic，可以用于cells, nets, references, 和 designs。你不希望DC碰的地方，都可以用它。  
set\_dont\_touch\_network可以穿过logic，可以用于clocks, pins, 或 ports，比上面的范围小。当你对设计不十分熟悉时，这个属性可能会传到你不希望的地方去。  
ideal\_net 顾名思义就是把这条net完全理想化--无穷大的驱动能力，没有延迟。有时会和上面的命令一起用。当你知道了它们的意思，如何使用取决于你的目的和得到的结果

**？**

对于Ideal\_net这个属性，设置的时候有个no\_propagate的属性，对于net来说一定要设这个，是不是就是为了阻止其穿过逻辑？那如果我设的是pin并且不加上no\_propagate能穿过逻辑吗？  
在建立clock的时候，我看到它会自动的设为ideal\_net，但是却还是会计算延迟，会不会因为clock path有逻辑的原因？

=

set\_ideal\_net = set\_ideal\_network -no\_propagate  
clock net 如果被用作data，就会有delay，那是因为net上面挂的clk pin的负载，不是ideal net本身

**？**

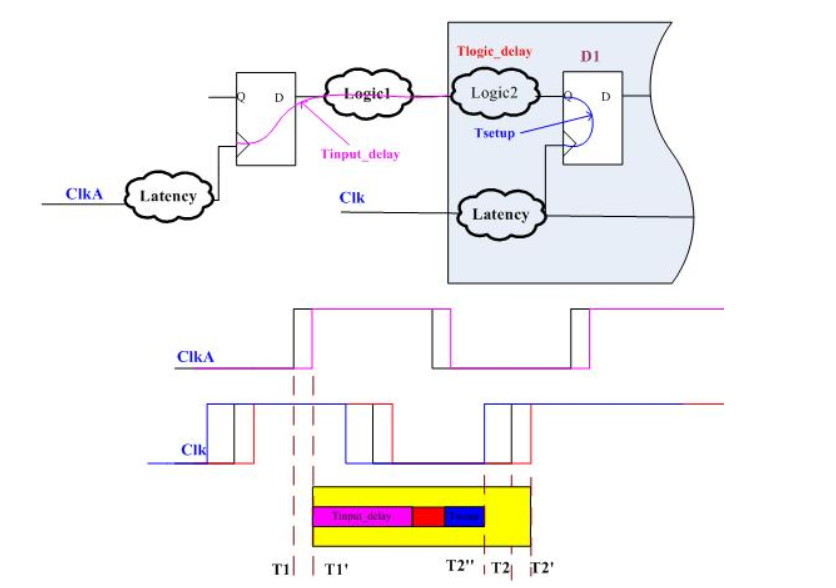
**我大体上明白了，clk在创建的时候，会默认为ideal net的，但当clk接入到data path的时候，D端就会考虑我clk上的负载，但并不会影响clk的ideal net的属性。**  
**假如我的clk需要门电路做gating，gating后的时钟也有很大的扇出，那我为了忽略掉延迟，是不是需要在gating后重新给clk定义ideal net?因为ideal net不能穿过逻辑。**

=

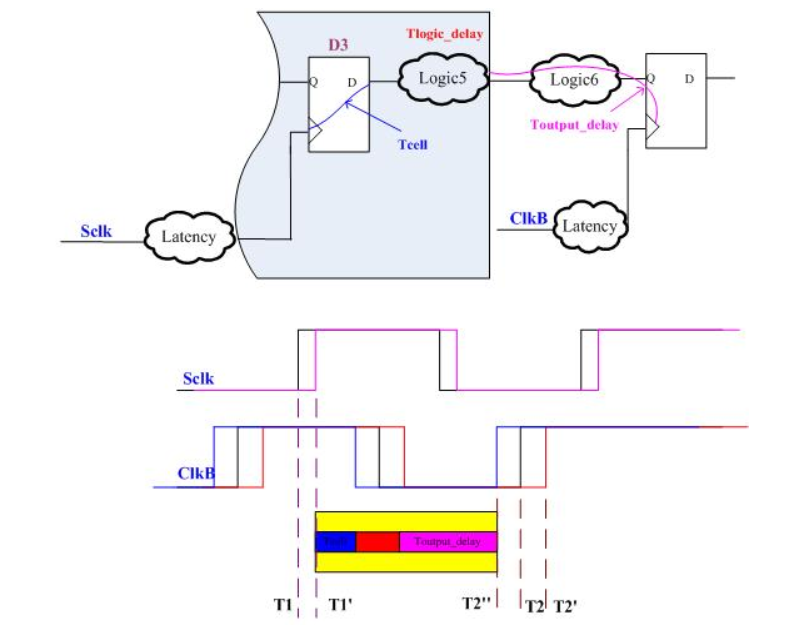
如果你的CG集成好的标准单元，它会自动继承ideal的属性。

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*set\_dont\_touch和set\_dont\_touch\_network\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

输入端口到时序器件的数据端口。

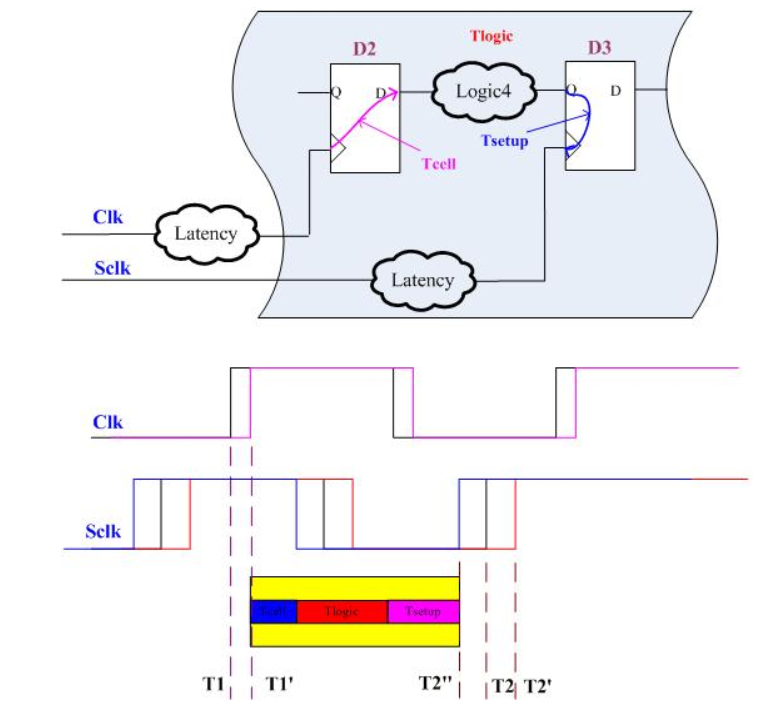
  
要求时间=T2+Tlatency-Tuncertainty\_setup-Tsetup  
到达时间=T1+Tlantency+Tinput\_delay+Tlogic2

时序器件的输出管脚到输出端口



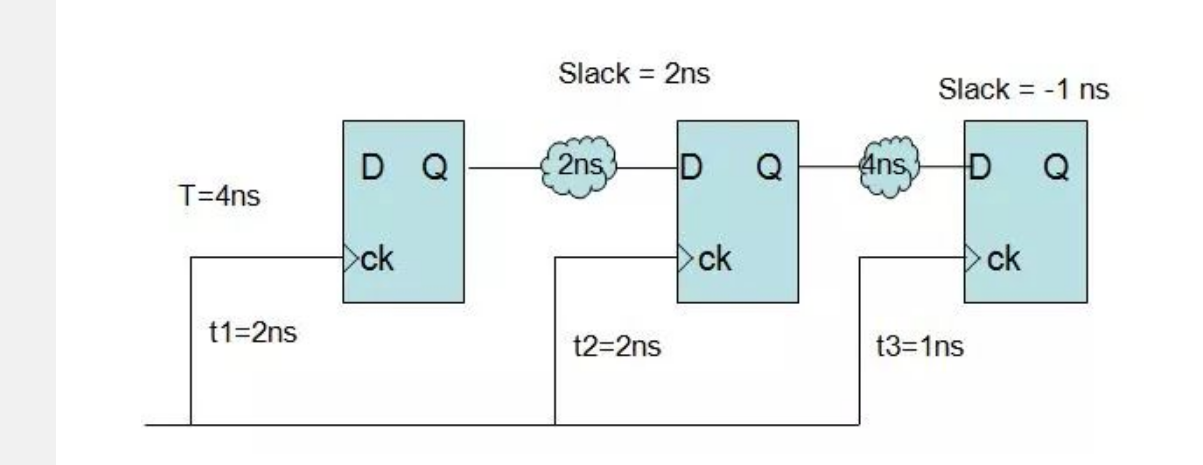
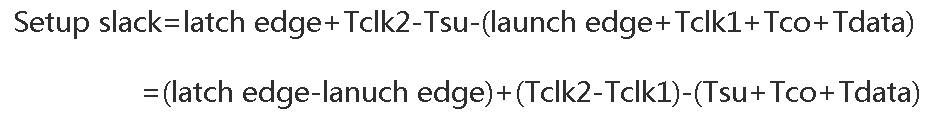
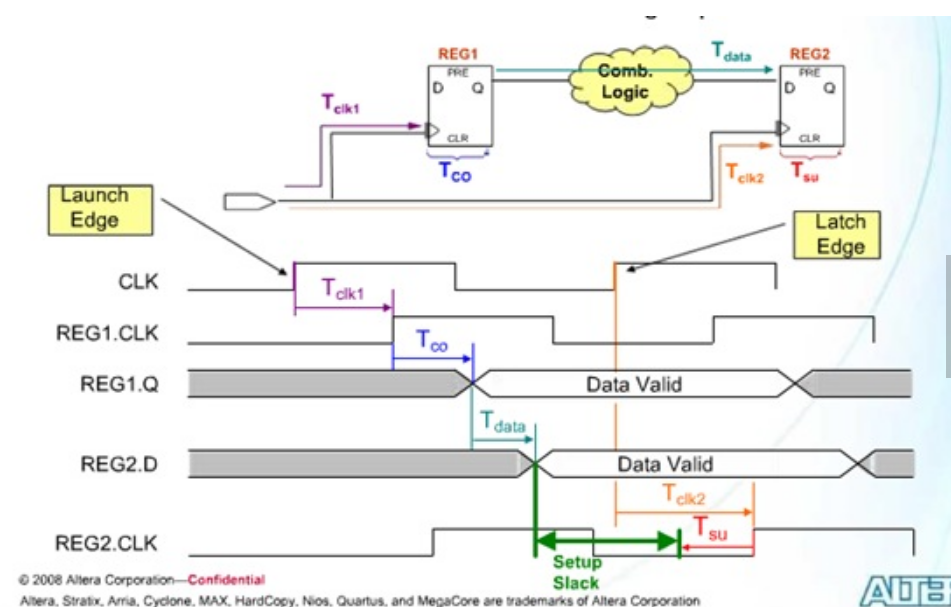
上图中：  
要求时间=T2+Tlatency-Toutput\_delay-Tuncertainty\_setup  
到达时间=T1+Tlatency+Tcell+Tlogic5

时序器件到时序器件



**路径上的 cell 延迟由 input\_transition 和 output\_load(包括扇出 pin 上的 load）决定，这个由查抄表可以得到。  
而 net 延迟是由 net 上的 R， C 决定的。在没有布局布线之前，我们不知道实际的 R， C 是多少， dc 根据互联线模型（set\_wire\_load\_model)来计算出 R， C。然后根据得到的 R， C 计算出 net 上的延迟：  
Net\_delay=R\*C\*OC  
其中系数 OC 是根据操作环境（set\_opearting\_conditions)中设置的 rc树模型得到。**

一般的工艺库的操作环境有三种， WORST， TYPICAL， BEST,分别是最差，典型，最坏。

**T=4，周期为4，Tdata为2，两个clk同时到达，slack=4-2=2，**

**setup check是latch和lauch间隔一个时钟周期，hold check才是在同一个时钟沿**

input\_transition输入信号的转换时间可以采用两种约束：  
1 直接设置转换时间  
set\_input\_transition 0.1 [get\_ports A]

2 采用设置输入驱动能力，驱动能力越大，转换时间越小，驱动能力越小，转换时间越大  
set\_drive或者set\_driving\_cell

**Slew Rate *vs.* Transition Time**  
  
首先，我要说明一下，slew和transition其实并非独立存在使用的词汇。在诸多教材中，一般是以slew rate和transition time，两个词组出现的。  
**Slew rate，信号改变的速率。**  
**Transition time，信号改变的持续时间。**  
在**静态时序分析（STA）**中，一个上升或者下降的波形通常用slew rate来表征其跳变的**快与慢**。Transition time就是用来记录这个信号在两个电平之间的**转换时间**。  
这里需要注意的是，transition time实际是slew rate的倒数。Transition time如果越大，那么slew rate就会越慢，反之亦然。  
  
上图展示了一些CMOS器件的输出波形图。理想状态下，我们期望得到一个完美的方波，不过，这显然不够现实。实际上，由于对电容的充放电，一个数字信号的波形往往存在上升和下降的**电压缓冲区间**。  
  
为了能量化这样的波形，我们近似采用一个线性上升或者下降的信号来模拟真实信号波形。注意到，真实信号波形里，不管是上升还是下降的时候，都会存在一定时间的线性区域。  
这个线性区域的**起点和终点**，可能因为某些估算模型而有所不同。在STA中，我们有时候会采用比较宽松的约束模式，例如20%／80%、10%／90%。  
  
当然，我们也可以采用激进一些的模式，如30%／70%。  
  
在一个**线性时序模型**里，一个信号经过一个时序单元可能产生两种新的信号波形。  
当输出信号的slew rate比输入信号的要快，也就是transition time变短了，说明这个单元对这个信号起到了**增强驱动**的作用。  
反之，当输出信号的slew rate比输入信号还要慢，就是transition time变长了，说明这个单元对这个信号起到了**削弱驱动**的作用。可能的原因是，这个单元的输出端驱动了较大的**负载**。  
  
所以，当我们要描述一个信号电平有所变化的时候，记得要用slew rate来表述**快慢**，用transition time来描述**持续时间**。

* 对于 cell 的延迟，dc 是根据 input\_transition 和 out\_load 对应的查找表来计算的。
* 对于 net 的延迟，dc 是根据 wire\_load\_model 中的fanout\_length 和 resistance,capacitance, area 的查找表计算的。
* 负载其实有两个概念，一个是阻性负载，这个负载当你提供了足够的驱动力就能够正确输出，否则电压就不对；另一个是容性负载，这个负载一般在系统里头，和系统能跑到什么速度相关。  
  一个输出驱动力大的话就能够带更大的负载，从这个意义上说，认为驱动=负载也是可以理解的。但其实这两个概念还是有些区别，侧重点不同。你上面也引用了，“电路的负载能力是下一级的load（即电容）总和”，这都是完全不同的两个量纲，怎么会一样。系统设计上，假设你需要跑10MHz的频率，你需要10mA的驱动力，则同一个容性负载上，要跑到20MHz的频率，你就需要20mA的驱动力。
* **对于Fanout 的理解**

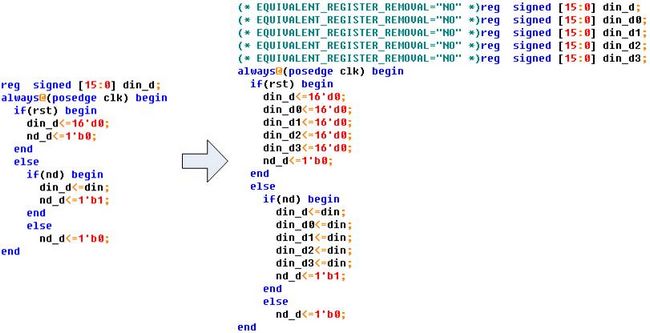
Fanout，即扇出，指模块直接调用的下级模块的个数，如果这个数值过大的话，在FPGA直接表现为net delay较大，不利于时序收敛。因此，在写代码时应尽量避免高扇出的情况。

**https://www.cnblogs.com/aikimi7/p/5945822.html**

综上，在遇到信号高扇出时，对于普通信号可采用

1.寄存器复制或者2.设置max\_fanout属性优化；而3.对于复位信号，可加入BUFG优化。

寄存器复制是解决高扇出问题最常用的方法之一，通过复制几个相同的寄存器来分担由原先一个寄存器驱动所有模块的任务，继而达到减小扇出的目的。通过简单修改代码，如图3所示，复制了4个寄存器：din\_d0、din\_d1、din\_d2、din\_d3，din\_d、din\_d0、din\_d1、din\_d2分别驱动2个DSP48E1，din\_d3驱动3个DSP48E1。其中在代码中为防止综合器优化相同寄存器，在对应信号上加入了(\* EQUIVALENT\_REGISTER\_REMOVAL="NO" \*)属性避免被优化。



  综合实现后得到时序报告如图4所示，该数据路径上输入数据fanout减为2，对应net delay也减小到了0.57ns。得到设计如图5所示，与期望的相同，复制了4个寄存器来分担fanout。在没有优化情况下，该设计的fmax：206.016MHz,经过寄存器优化后得到fmax：252.143MHz**.**

2. max\_fanout属性

         在代码中可以设置信号属性，将对应信号的max\_fanout属性设置成一个合理的值，当实际的设计中该信号的fanout超过了这个值，综合器就会自动对该信号采用优化手段，常用的手段其实就是寄存器复制。属性设置如下代码所示：

(\* max\_fanout = "3" \*)reg  signed [15:0] din\_d;

         将din\_d信号的max\_fanout属性设置成3，经过综合实现后，得到时序报告如图6所示，其中fanout只有2，相应的net delay也只有0.61ns，自动优化效果还不错。结构如图7所示，其中din\_d\_12\_1、din\_d\_12\_2、din\_d\_12\_3是综合器优化后自动添加，即实现了寄存器复制功能。经过设置max\_fanout属性优化后得到fmax：257.135MHz

       3. BUFG

         通常BUFG是用于全局时钟的资源，可以解决信号因为高扇出产生的问题。但是其一般用于时钟或者复位之类扇出超级大的信号，此类信号涉及的逻辑遍布整个芯片，而BUFG可以从全局的角度优化布线。而且一块[FPGA芯片](https://www.baidu.com/s?wd=FPGA%E8%8A%AF%E7%89%87&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnWN-nWNWuyFWnWKWrAFh0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLPWD1P1DsPHD4PWnzrH0krf" \t "_blank)中BUFG资源也有限，在7k325tffg900上也仅有32个，如果用于普通信号的高扇出优化也不大现实。因此，在时钟上使用BUFG是必须的，但是如果设计中遇到某些[复位信号](https://www.baidu.com/s?wd=%E5%A4%8D%E4%BD%8D%E4%BF%A1%E5%8F%B7&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnWN-nWNWuyFWnWKWrAFh0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLPWD1P1DsPHD4PWnzrH0krf" \t "_blank)因高扇出产生的时序问题时，可以在此信号上使用BUFG来优化

2015年10月13日 22:48:18 [Alan5555](https://me.csdn.net/alan5555) 阅读数：10125

综合分为三个部分：*Synthesis= Translate + Mapping + Optimization*。

*1、****Translate***是将*HDL*转化为*GTECH*库[元件](https://www.baidu.com/s?wd=%E5%85%83%E4%BB%B6&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)组成的逻辑电路，这步通过*read\_verilog*进行（*verilog*代码），*verilog*代码被读入后，将会被自动*translate*。*GTECH*是独立于工艺库的通用元件库。这个时候可以用*write -output ./unmapped/design.db*输出*unmapped*的二进制文件。

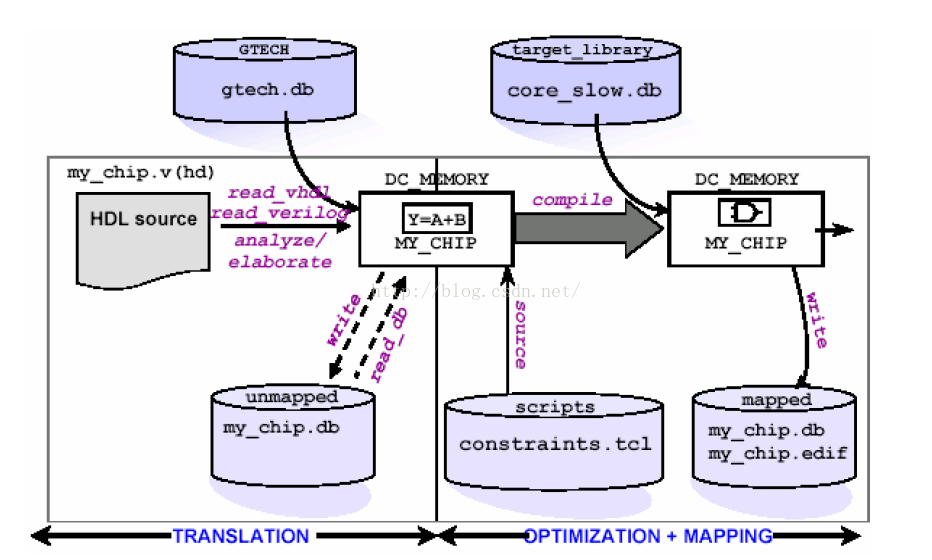
*2、****Mapping***是将*GTECH*映射到某一指定的工艺库，此[网表](https://www.baidu.com/s?wd=%E7%BD%91%E8%A1%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)包含了工艺参数。

*3、****Optimization***是将网表按设计者的约束进行优化。

后两步通过加约束后使用*compile*命令完成。完成*optimization*之后可以通过

*write -format verilog -hierarchy -output./p\_syn\_sim/design.v*

写成*verilog*格式的网表，这是*map*后的结果。如果输出的是顶层模块，所有模块的*module*都会输出到同一个文件中。顶层模块一般在最后。



*Sysnopsys*提供了一个例子，在下面的位置：

*$[synopsys](https://www.baidu.com/s?wd=synopsys&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)/doc/syn/guidelines*

首先，可以在*.synopsys\_dc.setup*里面设置库的位置和一些环境变量。里面包括所使用库的位置，*DC*涉及到几个库文件：

*1、***工艺库*(target\_library)***：工艺库是综合后电路网表要最终映射到的库，读入的*HDL*代码首先由*synopsys*自带的*GTECH*库转换成*Design Compiler*内部交换的格式，然后经过映射到工艺库和优化生成门级网表。工艺库他是由*Foundary*提供的，一般是*.db*的格式。这种格式是*DC*认识的一种内部文件格式，不能由文本方式打开。*.db*格式可以由文本格式的*.lib*转化过来，他们包含的信息是一致的。

*2、***链接库*(link\_library)***：*link\_library*设置模块或者单元电路的引用，对于所有*DC*可能用到的库，我们都需要在*link\_library*中指定，其中也包括要用到的*IP*。此处还要将*search\_path*加进来，这样才能找到*.db*文件，否则只会搜索当前目录。

*3、***符号库*(symbol\_library)***：*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*。符号库的后缀是*.sdb*，加入没有设置，*DC*会用默认的符号库取代。

*4、*综合库*(synthetic\_library)*：在初始化*DC*的时候，不需要设置标准的*DesignWare*库*standard.sldb*用于实现*Verilog*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。

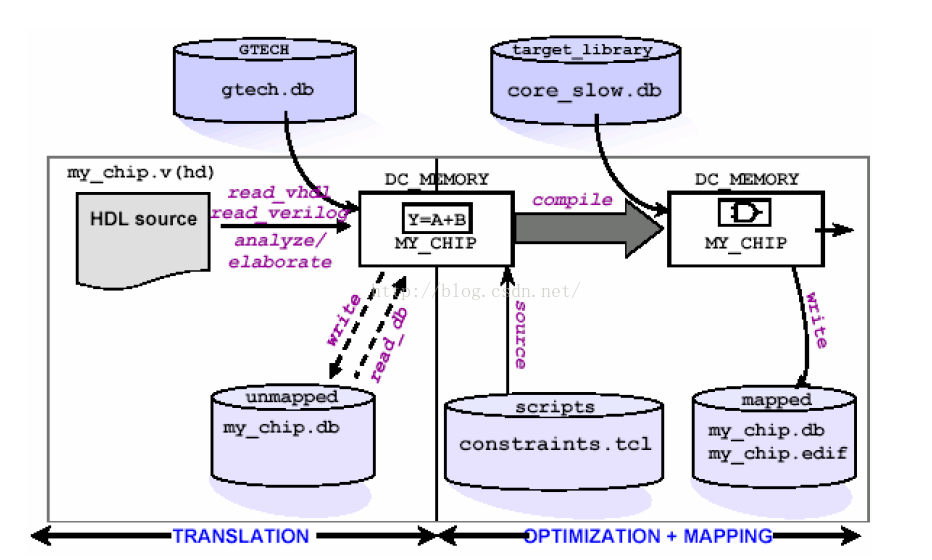
**读入设计：**

*read –format verilog[db*、*vhdl etc.] file //dcsh*的工作模式

*read\_db file.db //TCL*工作模式读取*DB*格式

*read\_verilog file.v //TCL*工作模式读取*verilog*格式

*read\_vhdlfile.vhd //TCL*工作模式读取*VHDL*格式



读取源程序的另外一种方式是配合使用*analyze*命令和*elaborate*命令：*analyze*是分析*HDL*的源程序并将分析产生的中间文件存于*work(*用户也可以自己指定*)*的目录下；*elaborate*则在产生的中间文件中生成*verilog*的模块或者*VHDL*的实体，缺省情况下，*elaborate*读取的是*work*目录中的文件。

***Link*命令：**

当读取完所要综合的模块之后，需要使用*link*命令将读到*Design Compiler*存储区中的模块或实体连接起来，如果在使用*link*命令之后，出现*unresolved designreference*的警告信息，需要重新读取该模块，或者在*.synopsys\_dc.setup*文件中添加*link\_library*，告诉*DC*到库中去找这些模块，同时还要注意*search\_path*中的路径是否指向该模块或单元电路所在的目录。

**约束：**

**约束面积**：*set\_max\_area 100*

单位：定义这个单位有三种可能的标准：一种是将一个二输入与非门的大小作为单位*1*；第二种是以晶体管的数目规定单位；第三种则是根据实际的面积*(*平方微米等等*)*。至于设计者具体用的是哪种单位，可以通过下面的一个小技巧得到*——*即先综合一个二输入与非门，用*report\_area*看他的面积是多少，如果是*1,*则是按照第一种标准定义的；如果是*4*，则是第二种标准；如果是其他的值，则为第三种标准。

**约束时序**：

*DC*可以处理的是同步时序电路。

模块内部的，寄存器之间的逻辑的约束（*X*）：

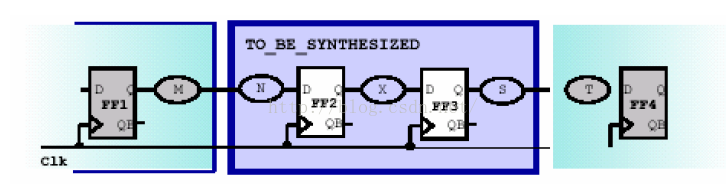
*Creat\_clock –period 10[get\_ports clk]*

*Set\_dont\_touch\_network[get\_clocks clk]*

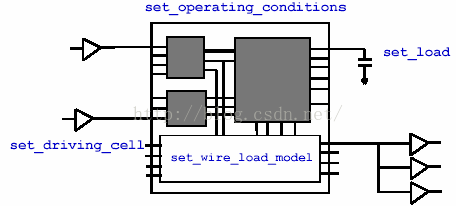
第一句话将*clk*定义为时钟，并定义周期为*10ns*，因此可以约束寄存器间组合逻辑的*critical path*不能大于*10ns*，进行约束。第二句的意义为，不要在综合的时候对时钟树进行优化，因为时钟树有特别的办法进行综合。

*set\_output\_delay -max 3.0 -clockclk [all\_outputs]*用于约束输入段的组合逻辑，在第一个寄存器前的部分（*N*）。

*set\_input\_delay -max 3.0 -clockclk [all\_inputs]*用于约束输出的组合逻辑，最后一个寄存器以后的部分（*S*）。



**约束环境：**



*set\_load 2[all\_outputs]*设置输出的负载（这里的单位也是由*Foundry*提供，具体的单位，可以通过*report\_lib*命令查看，一般而言是*pf*。）

*set\_driving\_cell-lib\_cell FD1 [all\_inputs]*设置输入的负载，此处设置为一个模块来驱动。

*set\_operating\_conditions–max slow –min fast*？？？？？？？？？？？？？？？？设置工作条件。在默认情况下，*Design Compiler*不会自动指定工作条件，我们可以先通过*report\_lib*命令来列出在当前的工艺库里提供了哪几种工作条件。然后指定需要用到的工作条件，在做建立时间分析的时候需要用到最差情况的条件。

*setauto\_wire\_load\_selection true*设置线负载模型，它是也是由*Foundry*提供的。*Foundry*根据其他用这个工艺流片的芯片的连线延时进行统计，从而得到这个值。此处设置的是自动选择。也可以设置成其他，如：*set\_wire\_load\_model–name 160KGATES*。还可以通过*set\_wire\_load\_mode enclosed*设置负载模式。

在定义完环境属性之后，我们可以使用下面的几个命令检查约束是否施加成功*——*

*check\_timing*检查设计是否有路径没有加入约束

*check\_design*检查设计中是否有悬空管脚或者输出短接的情况

*write\_script*将施加的约束和属性写出到一个文件中，可以检查这个文件看看是否正确

***DesignWare*选择**：*DW*选择是结构级优化的一个很主要的特点，在这个阶段*DC*能够根据设计者施加的时序或者面积的约束在*DW*的不同实现方式中找到它认为最佳的实现方案。比如一个加法运算，*DC*会根据约束选择最合适的加法器，如*lookahead*还是选择*carry bypass*。

***Characterize*命令：**

用*characterize*这个命令将约束条件传递到子模块，然后保存这些传递下来的约束条件，再用这样条件对这个子模块再重新编译。

**报告（*report*）：**

*report\_timing*生成时序报告

**综合策略：**

*Top-down*方式的层次化编译策略有这样三个步骤：

*1*．读入整个设计。

*2*．处理多次引用的模块；基于设计要求在顶层加约束条件以及定义属性。

*3*．编译。

*Bottom-up*有下面七个步骤：

*1*．独立编译子模块，用估计的驱动和负载。用一个缺省的脚本文件来估计驱动和负载。

*2*．将整个编译好的设计读入。

*3*．对一个子模块用*characterize*命令来获得端口信息。

*4*．用*write\_script*来将这些传递过来的信息保存下来。

*5*．清除内存，将前面得到传递信息的子模块调入并且用保存下来的脚本文件重新编译。为了这些描述信息能够应用，需要将*db*格式的文件读入。

*6*．将这个新编译的子模块和其余的子模块全部再次调入*DC*。

*7.*选择另外一个子模块，重复*3~7*步，直到所有的子模块都重新编译过了。