Xilinx时序约束笔记

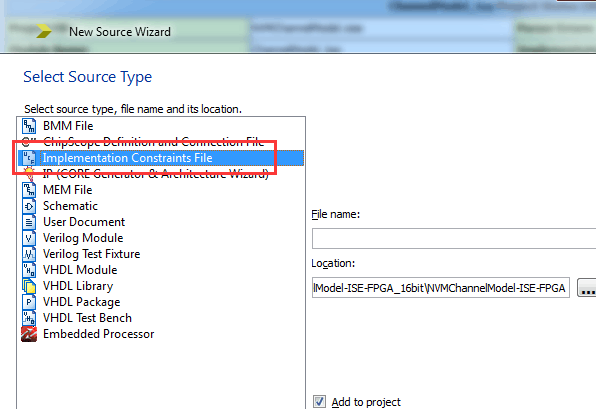
# 用户文件

FPGA设计中的约束文件有3类：用户设计文件（.UCF文件）、网表约束文件（.NCF文件）以及物理约束文件（.PCF文件），可以完成时序约束、管脚约束以及区域约束。**3类约束文件的关系为：用户在设计输入阶段编写UCF文件，然后UCF文件和设计综合后生成NCF文件，最后再经过实现后生成PCF 文件。**

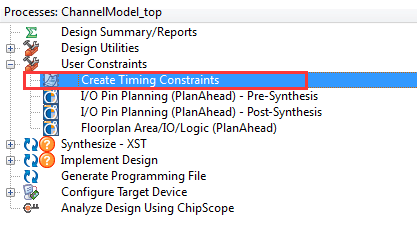
UCF文件是ASC 2码文件，描述了逻辑设计的约束，可以用文本编辑器和Xilinx约束文件编辑器进行编辑。NCF约束文件的语法和UCF文件相同，二者的区别在于： UCF文件由用户输入，NCF文件由综合工具自动生成，当二者发生冲突时，以**UCF**文件为准，这是因为UCF的优先级最高。PCF文件可以分为两个部分：一部分是映射产生的物理约束，另一部分是用户输入的约束，同样用户约束输入的优先级最高。一般情况下，用户约束都应在UCF文件中完成，不建议直接修改 NCF文件和PCF文件。

约束文件的后缀是.ucf，所以一般也被称为UCF文件。创建约束文件有两种方法，一种是通过新建方式，另一种则是利用过程管理器来完成。

第一种方法：新建一个源文件，在代码类型中选取“Implementation Constrains File”，在“File Name”中输入约束文件的名称。单击“Next”按键进入模块选择对话框，选择要约束的模块，然后单击“Next”进入下一页，再单击“Finish”按键完成约束文件的创建。



第二种方法：在工程管理区中，将“Source for”设置为“Synthesis/Implementation”。“Constrains Editor”是一个专用的约束文件编辑器，双击过程管理区中“User Constrains”下的“Create Timing Constrains”就可以打开“Constrains Editor”。



需要注意的是，UCF文件是大小敏感的，端口名称必须和源代码中的名字一致，且端口名字不能和关键字一样。但是关键字NET是不区分大小写的。

# UCF文件的语法说明：

## 1．语法

UCF文件的语法为：

{NET|INST|PIN} "signal\_name" Attribute;

其中，“signal\_name”是指所约束对象的名字，包含了对象所在层次的描述；“Attribute”为约束的具体描述；语句必须以分号“；”结束。可以用“#”或“”添加注释。需要注意的是：UCF文件是大小写敏感的，信号名必须和设计中保持大小写一致，但约束的关键字可以是大写、小写甚至大小写混合。例如：

NET "CLK" LOC = P30;

“CLK”就是所约束信号名，LOC = P30；是约束具体的含义，将CLK信号分配到FPGA的P30管脚上。

对于所有的约束文件，使用与约束关键字或设计环境保留字相同的信号名会产生错误信息，除非将其用" "括起来，因此在输入约束文件时，最好用" "将所有的信号名括起来。

## 2．通配符

在UCF文件中，通配符指的是“\*”和“?”。“\*”可以代表任何字符串以及空，“?”则代表一个字符。在编辑约束文件时，使用通配符可以快速选择一组信号，当然这些信号都要包含部分共有的字符串。例如：

NET "\*CLK?" FAST;

将包含“CLK”字符并以一个字符结尾的所有信号，并提高了其速率。

在位置约束中，可以在行号和列号中使用通配符。例如：

INST "/CLK\_logic/\*" LOC = CLB\_r\*c7;

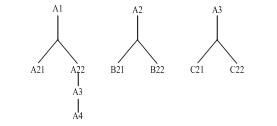
把CLK\_logic层次中所有的实例放在第7列的CLB中。

## 3．定义设计层次

在UCF文件中，通过通配符\*可以指定信号的设计层次。其语法规则为：  
\* 遍历所有层次

Level1/\* 遍历level1及以下层次中的模块

Level1/\*/ 遍历level1中的模块，但不遍历更低层的模块例4-5 根据图4-75所示的结构，使用通配符遍历表4-3所要求的各个模块。

[](http://www.openhw.org/data/11-10/19200027021056/1319682536_8c19ba69.jpg)

# 管脚和区域约束语法（或使用Floorplanner约束）

LOC约束是FPGA设计中最基本的**布局约束和综合约束**，能够定义基本设计单元在FPGA芯片中的位置，**可实现绝对定位、范围定位以及区域定位**。此外， LOC还能将一组基本单元约束在特定区域之中。LOC语句既可以书写在约束文件中，也可以直接添加到设计文件中。换句话说，**ISE中的FPGA底层工具编辑器（FPGA Editor）、布局规划器（Floorplanner）和引脚和区域约束编辑器的主要功能都可以通过LOC语句完成。**

LOC语句语法

INST "instance\_name " LOC = location;

其中“location”可以是FPGA芯片中任一或多个合法位置。如果为多个定位，需要用逗号“,”隔开，如下所示：

LOC = location1,location2,...,locationx;

目前，还不支持将多个逻辑置于同一位置以及将多个逻辑至于多个位置上。需要说明的是，多位置约束并不是将设计定位到所有的位置上，而是在布局布线过程中，布局器任意挑选其中的一个作为最终的布局位置。

范围定位的语法为：

INST “instance\_name” LOC=location:location [SOFT];

**表-常用的LOC定位语句**



使用LOC完成端口定义时，其语法如下：

NET "Top\_Module\_PORT" LOC = "Chip\_Port";

其中，“Top\_Module\_PORT”为用户设计中顶层模块的信号端口，“Chip\_Port”为FPGA芯片的管脚名。

**LOC语句中是存在优先级的，当同时指定LOC端口和其端口连线时，对其连线约束的优先级是最高的。例如，在下图中，LOC=11的优先级高于LOC=38。**

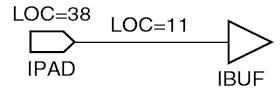
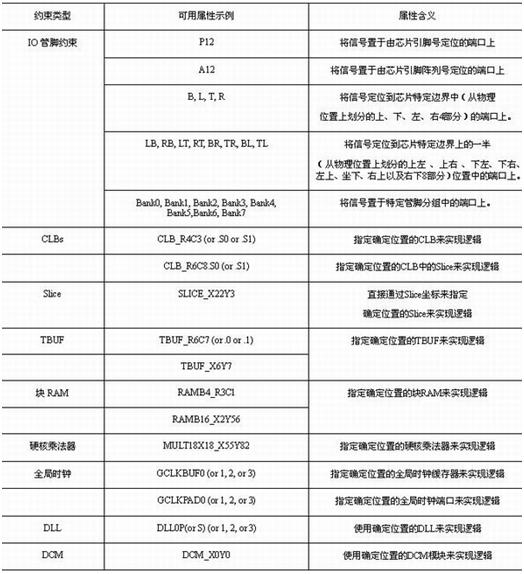
[](http://www.openhw.org/data/11-10/19200027021056/1319682511_6a0cc95e.jpg)

图 LOC优先级示意图

## LOC属性说明

LOC语句通过加载不同的属性可以约束管脚位置、CLB、Slice、TBUF、块RAM、硬核乘法器、全局时钟、数字锁相环（DLL）以及DCM模块等资源，基本涵盖了FPGA芯片中所有类型的资源。由此可见，LOC语句功能十分强大，下面列出了LOC的常用属性。

表-LOC语句常用属性列表



# 时序约束的语法（或使用constraint editor约束）

UCF文件中时序约束的语法

约束UCF文件，从**Constrains Editor直接输入是最方便、最直接的添加约束的方法了**。以下是几种常用的语法：

## 周期约束（PERIOD）

PERIOD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PERIOD约束检查时钟域内所有同步元件的时序是否满足要求，它将检查与同步时序约束端口相连接的所有路径的延迟，但是不会检查PAD到寄存器的路径。

**附加时钟周期约束的首选方法（Preferred Method）语法如下：**

**TIMESPEC “TS\_identifier” = PERIOD “TNM\_reference” period {HIGH|LOW} [high\_or\_low\_time]**

其中“[ ]”内为可选项，“{ }”为必选项，参数period为要求的时钟周期，可以使用ps、ns、us或者ms等单位，大小写都可以，缺省单位为ns。HIGH|LOW关键词指出时钟周期里的第一个脉冲是高电平还是低电平，而high\_or\_low\_time为脉冲的延续时间，缺省单位也是ns，如果不提供该项，则缺省占空比为50%。

**TIMESPEC是一个基本时序相关约束标识**，表示本约束为时序规范。TSidentifier包括字母TS和一个标识符identifier（为ASCII码字符串）共同组成一个时序规范。

例如定义时钟周期约束时，首先在时钟网线clk上附加一个TNM\_NET约束，把clk驱动的所有同步元件定义为一个名为sys\_clk的分组，然后使用TIMESPEC约束定义时钟周期。

NET “clk” TNM\_NET=”sys\_clk”;

#定义clk驱动的所有同步元件为sys\_clk的分组

TIMESPEC “TS\_sys\_clk”= PERIOD “sys\_clk” 50 HIGH 30;

#定义可引用的时序规范TS\_sys\_clk，

#这个规范规定sys\_clk组的时钟情况

而定义派生时钟的语法如下：

TIMESPEC “TSidentifier\_2”=PERIOD “timegroup\_name” “TSidentifier\_1” [\*or/] factor PHASE [+|-] phase\_value [units];

#定义第二个时序规范TSidentifier\_2

#其内容是名为timegroup\_name的分组是第一个时序规范TSidentifier\_1派生的

其中TSidentifier\_2为要定义的派生时钟，TSidentifier\_1为已经定义的时钟，factor指出两者周期的倍数关系，是一个浮点数。phase\_value指出两者之间的相位关系，为浮点数。例如：

定义主时钟clk0：

TIMESPEC “TS01” = PERIOD “clk0” 10.0 ns;

定义派生时钟clk180，其相位与主时钟相差180°：

TIMESPEC “TS02” = PERIOD “clk180” TS01 PHASE + 5.0 ns;

定义派生时钟clk180\_2,其周期为主时钟的1/2，并延迟2.5ns：

TIMESPEC “TS03” = PERIOD “clk180\_2” TS01 /2 PHASE + 2.5 ns;

## 偏移约束(OFFSET)

**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（端口）信号，不能用于内部信号**。使用该约束可以为综合实现工具指出输入数据到达的时刻，或者输出数据稳定的时刻，从而在综合实现中调整布局布线过程，使正在开发的FPGA/CPLD的输入建立时间以及下一级电路的输入建立时间满足要求。

基本语法如下：

OFFSET = {IN|OUT} “offset\_time” [units] {BEFORE|AFTER} “clk\_name” [TIMEGRP “group\_name”];

其中{IN|OUT}说明约束的是输入还是输出，offset\_time为FPGA引脚数据变化与有效时钟沿之间的时间差，BEFORE|AFTER说明该时间差在有效时钟沿的前面还是后面，TIMEGRP “group\_name”定义了约束的触发器组，缺省时约束该时钟驱动的所有触发器。

## 分组约束

使用TNM（Timing Name）约束可以选出构成一个分组的元件，并赋予一个名字，以便给它们附加约束。TNM\_NET（timing name for nets）约束只加在网线上，其作用与TNM加在网上时基本相同，即把该网线所在路径上的所有有效同步元件作为命名组的一部分。不同之处在于当TNM约束加在PAD NET上时，TNM的值将被赋予PAD，而不是该网线所在的路径上的同步元件，即TNM约束不能穿过IBUF。而用TNM\_NET约束就不会出现这种情况。

## 专门约束

附加约束的一般策略是首先附加整体约束，例如PERIOD、OFFSET等，然后对局部的电路附加专门约束，这些专门约束通常比整体约束宽松，通过在可能的地方尽量放松约束可以提高布线通过率，减小布局布线的时间。

FROM\_TO约束在两个组之间定义时序约束，对两者之间的逻辑和布线延迟进行控制，这两个组可以是用户定义的，也可以是与定义的。用户可以使用TNM\_NET、TNM和TIMEGRP定义组，而与定义组主要包括FFS、LATCHES、PADS和RAMS等。语法如下：

TIMESPEC “TSname” = FROM “group1” TO “group2” value;

其中value为延迟时间，可以使具体数值或表达式。

MAXDELAY约束定义了特定网线上的最大延迟，其语法如下：

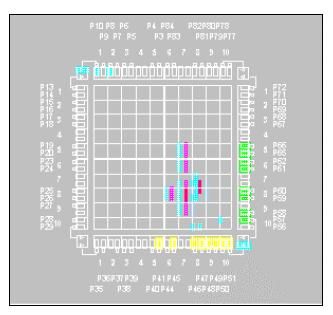
NET “net\_name” MAXDELAY = value units;

# 时序约束概念和效果(特权)

执行工具不会试图寻找达到最快速的布局&布线路径。——取而代之的是，执行工具会努力达到你所期望的性能要求。

性能要求和时序约束相关——时许约束通过将逻辑元件放置的更近一些以缩短布线资源从而改善设计性能。

**没有时序约束的例子**

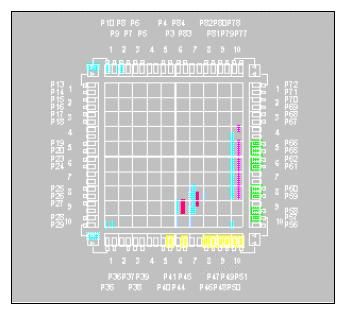


该工程没有时序约束和管脚分配

——注意它的管脚和放置

——该设计的系统时钟频率能够跑到50M

**添加时序约束的例子**



  和上面是相同的一个设计，但是加入了3个全局时序约束。

——它最高能跑到60M的系统时钟频率

——注意它**大部分的逻辑的布局更靠近器件边沿其相应管脚的位置**

## 更多关于时序约束

         时序约束应该用于界定设计的性能目标

1.       太紧的约束将会延长编译时间

2.       不现实的约束可能导致执行工具罢工

3.       查看综合报告或者映射后静态时序报告以决定你的约束是否现实

执行后，查看布局布线后静态时序报告以决定是否你的性能要求达到了——如果约束要求没有达到，查看时序报告寻找原因。

## 路径终点

         有两种类型的路径终点：

1.       I/O pads

2.       同步单元（触发器，锁存器，RAMs）

时序约束的两个步骤：

1.       路径终点生产groups（顾名思义就是进行分组）

2.       指点不同groups之间的时序要求

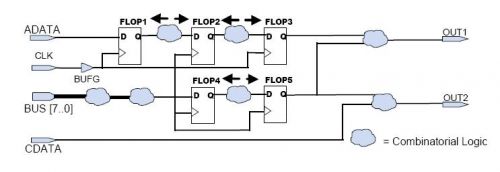
全局约束使用默认的路径终点groups——即所有的触发器、I/O pads等

**问题思考**

单一的全局约束可以覆盖多延时路径

如果箭头是待约束路径，那么什么是路径终点呢？

所有的寄存器是否有一些共同点呢？



**问题解答**

什么是路径终点呢？

         ——FLOP1,FLOP2,FLOP3,FLOP4,FLOP5。

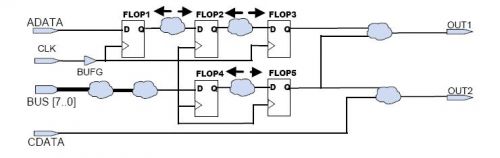
         所有的寄存器是否有一些共同点呢？

         ——它们共享一个时钟信号，约束这个网络的时序可以同时覆盖约束这些相关寄存器间的延时路径。

## 周期约束

周期约束覆盖由**参与网络钟控的的同步单元之间的路径延时**。

         周期约束不覆盖的路径有：input pads到output pads之间的路径（纯组合逻辑路径），input pads到同步单元之间的路径，同步单元到output pads之间的路径。



周期约束**使用最准确的时序信息**，使其能够自动的计算：

1.       源寄存器和目的寄存器之间的时钟偏斜（Clock Skew）

2.       负沿钟控的同步单元

3.       不等同占空比的时钟

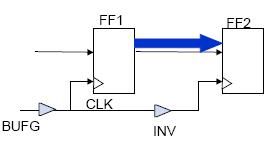
4.       时钟的输入抖动（jitter）

假设：

1.       CLK信号占空比为50%

2.       周期约束为10ns

3.       由于FF2将在CLK的下降沿触发，两个触发器之间的路径实际上将被约束为10ns的50%即5ns



### 时钟输入抖动（Clock Input Jitter）

         时钟输入抖动是源时钟的不确定性（clock uncertainty）之一

         时钟的不确定时间必须从以下路径扣除：

         ——周期约束建立时间路径

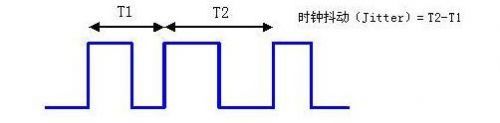
         ——OFFSET IN约束的建立时间路径

         时钟的不确定时间必须添加到以下路径中：

         ——周期约束保持时间路径

         ——OFFSET IN约束保持时间路径

         ——OFFSET OUT约束路径



## Pad-to-Pad约束

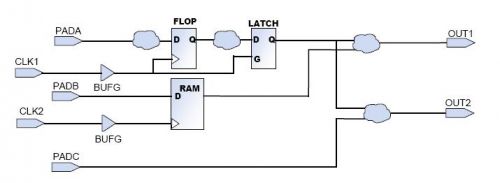
         ——不包含**任何同步单元的纯组合逻辑电路**

         ——**纯组合逻辑延时路径开始并结束于I/O pads**，所以通常会被我们遗漏而未约束

**问题思考**

         哪些路径是由CLK1进行周期约束？

         哪些路径是由pad-to-pad进行约束？

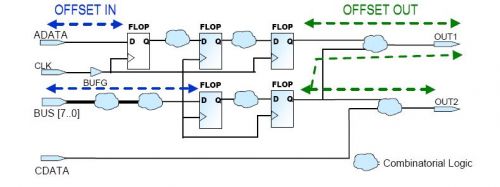


**OFFSET约束**

         OFFSET约束覆盖以下路径：

——从input pads到同步单元（OFFSET IN）

——从同步单元到output pads（OFFSET OUT）



### OFFSET约束特性

 OFFSET约束自动计算时钟分布延时

1.       提供最准确的时序信息

2.       大量增加输入信号到达同步单元的时间（时钟和数据路径并行）

3.       大量减少输出信号到达输出管脚的时间（时钟和数据路径先后）

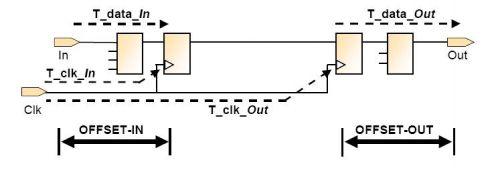
OFFSET约束也可以解释时钟输入抖动——使用抖动确定关联的周期约束

### 时钟延时

         数据路径延时和时钟分布延时都需要在OFFSET计算中使用到

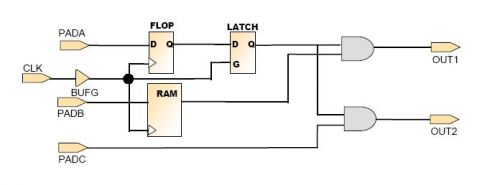
         ——OFFSET IN = T\_data\_in – T\_clk\_in

         ——OFFSET OUT = T\_data\_out + T\_clk\_out



**问题思考**

         在这个电路中哪些路径是由OFFSET IN 和 OFFSET OUT来约束的？



**问题解答：**

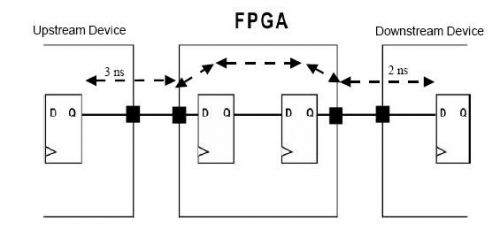
         ——OFFSET IN：PADA to FLOP and PADB to RAM

——OFFSET OUT：LATCH to OUT1, LATCH to OUT2, and RAM to OUT1

**问题思考**

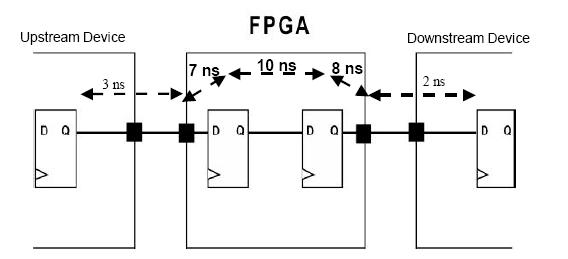
         下面给出的系统框图里，你将给出什么样的约束值以使系统能够跑到100MHz？

         ——假设在下面的器件之间没有时钟偏斜



**问题解答：**

         PERIOD = 10 ns , OFFSET IN (BEFORE) = 7 ns and OFFSET OUT (AFTER) = 8 ns



**小结**

1.       性能期望和时序约束相关联

2.      **周期约束覆盖同步单元之间的延时路径**

3.       OFFSET约束覆盖从输入管脚到同步单元和从同步单元到输出管脚之间的延时路径

## 3. 特定路径时序约束

         使用全局时序约束（PERIOD,OFFSET,PAD-TO-PDA）将约束整个设计

         仅仅使用全局约束通常会导致过约束

         ——约束过紧

——编译时间延长并且可能阻止实现时序目标

——通过综合工具或者映射后时序报告重新审视性能评估

**特定路径约束能够覆盖全局时序约束在特定路径上的约束**

——这就允许设计者放宽特定路径的时序要求

### 更多关于特定路径约束

         你的设计器件的内部面积将会从特定路径约束收益

1.       多周期路径Multi-cycle paths

2.       跨时钟域路径

3.       双向总线

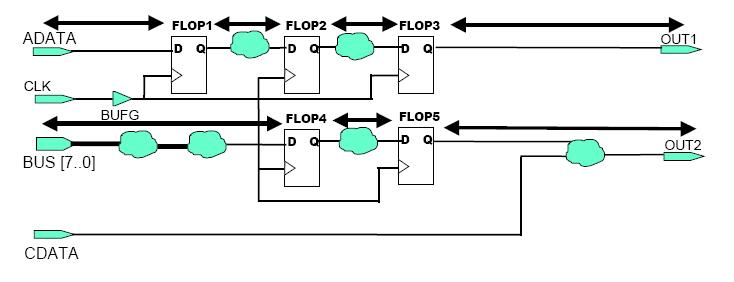
4.       I/O时序

特定路径约束应该由你的性能目标来界定，不能够不加限制的随意放置

## 全局约束回顾

         使用全局PERIOD,OFFSET IN和OFFSET OUT约束将约束所有以下的路径

         这使得控制设计的总体性能更加容易



## 特定路径约束实例

         一条特定路径约束对于路径本身的优化微乎其微

         这有助于你更好的控制设计性能，并带给执行工具更大的灵活性以达到你的性能和使用要求

         生成特定路径约束需要两个步骤：

1.       多个有共同时序要求的特定路径终点生成一个groups

2.       关联两个groups，指定它们的特定路径的时序要求

## 生成终点路径的Groups

**特定路径时序约束在终点路径较好的分组后会更加高效——否则，约束一个大的工程将极其耗时耗力。**

         约束编辑有助于你更容易的进行路径终点（pads, flip-flops, latches, and RAMs）进行Groups分组。

使用约束编辑器，终点路径的分组有以下选项：

– Group by nets

– Group by instance name

– Group by hierarchy

– Group by output net name

– Timing THRU Points option

– Group by clock edge

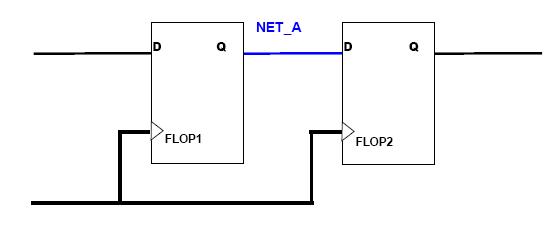
## Nets 分组与output net name分组对比

由net分组的 “NET\_A”将生成一个只包含FLOP2的group

——Group包含选择网络所驱动的寄存器

由output net name分组的“NET\_A”将生成一个只包含FLOP1的group

——Group包含选择网络的源寄存器



## 回顾全局OFFSET约束

在时钟行中使用Pad-to-Setup和Clock-to-Pad列为所有出于该时钟域的I/O路径指定OFFSETs。为大多数I/O路径进行约束的最简单方法——然而，这将会导致一个过约束的设计。

## 指定管脚的OFFSET约束

         使用Pad-to-Setup和Clock-to-Pad列为每个I/O路径指定OFFSETs。

         这种约束方法适用于只有少数管脚需要不同的时序约束。

         更常用的方法是：

1.       为Pads生成Groups

2.       对生成的指定Groups进行OFFSET IN/OUT约束

## 双沿时钟的OFFSET约束

         OFFSET约束指明了FPGA管脚的输入数据和初始时钟之间的关系。

         初始时钟沿在周期约束定义中出现关键词“高”和“低”。

                  ——高：初始时钟上升沿（默认），即上升沿锁存数据

                  ——低：初始时钟下降沿

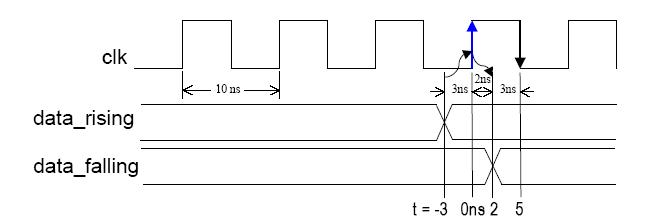
         如果所有的I/O都由时钟的一个沿控制，那么你可以使用这个关键字高或低进行周期约束。

如果两个沿都用到，你就必须进行两个OFFSET的约束。

                  ——每个OFFSET对应一个时钟沿

                  ——DDR寄存器也是这样使用的一个例子

## 双沿时钟的OFFSET IN约束



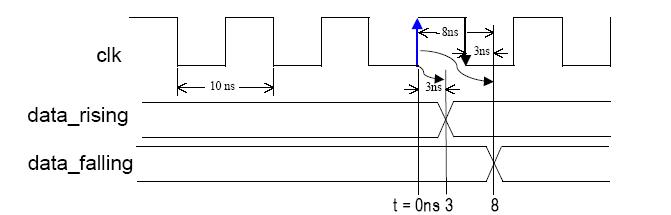
输入数据在上升沿或者下降沿之前3ns有效——周期约束为10ns，初始上升沿，占空比为50%。

         为每个时钟沿生成一个时钟Groups

                  ——输入时钟的上升沿，OFFSET = IN 3ns BEFORE CLK;

                  ——输入时钟的下降沿，OFFSET = IN -2ns BEFORE CLK;（在初始时钟的上升沿后2ns = 时钟下降沿前3ns）

## 双沿时钟的OFFSET OUT约束



输出数据必须在时钟的上升沿或者下降沿后3ns内有效——周期约束为10ns，初始上升沿，占空比为50%。

         为每个时钟沿生成一个时钟Groups

                  ——输入时钟的上升沿，OFFSET = OUT 3ns AFTER CLK;

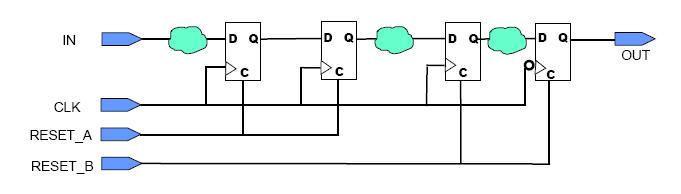
                  ——输入时钟的下降沿，OFFSET = OUT 8ns AFTER CLK;（在初始时钟的上升沿后8ns = 时钟下降沿后3ns）

**问题思考**

         特定路径时序约束如何改善了设计性能？

你如何约束这个设计使其内部时钟频率达到100 MHz？

输入（数据）将在时钟CLK的上升沿到达前3ns内有效。输出数据必须在时钟CLK的下降沿后4ns内稳定下来。写出合适的OFFSET约束？



**问题解答**

         特定路径时序约束如何改善了设计性能？

                  ——它使得执行工具更加灵活的达到你的时序要求。

你如何约束这个设计使其内部时钟频率达到100 MHz？

——给时钟信号CLK施加一个10ns的全局周期约束。

写出合适的OFFSET约束？

OFFSET = IN 3 ns BEFORE CLK;

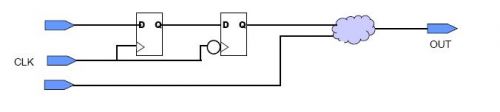
OFFSET = OUT 4 ns AFTER CLK;

## 时钟上升沿和下降沿之间的时序约束

         周期约束可以自动计算两个沿的约束——包括调整非50%占空比的时钟。

         例：一个CLK时钟周期约束为10ns，能够应用5ns的约束到两个寄存器之间。

         不需要特定路径应用到这个例子中。

[](http://maxim.eefocus.com/data/11-09/1316679158_fca764fb.jpg)

## 相关时钟域的约束

**为一个时钟进行周期约束——以这个周期约束确定相关的时钟。**

         执行工具将根据它们的关系来决定如何处理跨时钟域。

         DCM有多个输出：

                   ——确定DCM输入时钟的周期约束

                   ——执行工具将会从这个周期约束推导出其输出的约束

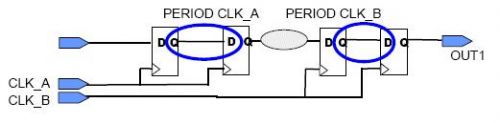
                   ——所有的约束将会和原始的周期约束相关

## 不相关时钟域的约束

         在这个例子中，周期约束不覆盖到处于两个时钟域之间的任何延时路径。——这是默认的处理方式。

         你必须添加一个约束覆盖到相关时钟域之间的路径中。——例如，频率相同，但是CLK\_B有一些相位偏移。

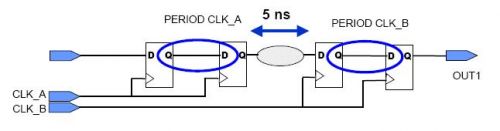
**在两个不相关的时钟域你就必须添加一些同步电路。**

[](http://maxim.eefocus.com/data/11-09/1316679161_4d62f931.jpg)

 约束两个时钟域之间的路径。

         ——使用Groups by NETs选项为CLK\_A和CLK\_B定义groups，如果你为每个时钟添加完周期约束，这个步骤将自动完成。

         ——在这个寄存器的groups之间指定快速/慢速例外约束。

[](http://maxim.eefocus.com/data/11-09/1316679165_7e14d7be.jpg)

## 多周期路径约束

         多周期约束应用在连续几个时钟周期内寄存器不需要更新的情况。

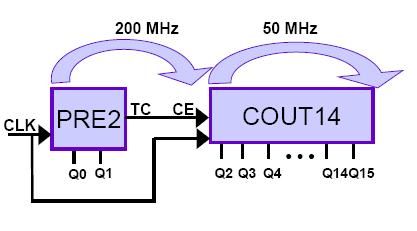
                   ——总是至少需要一个时钟周期才更新。

                   ——通常的，这样的寄存器由时钟使能信号控制。

         一个分段计数器就是这样的一个例子。

                   ——COUT14每隔4个时钟周期才更新一次。

                   ——这些寄存器间的路径就算是多周期路径。

[](http://maxim.eefocus.com/data/11-09/1316679168_e73f0f07.jpg)

## False 路径

         False路径选项将用于防止约束覆盖到特定路径

## 时序约束优先级

         从高到低为：

1.       False路径——将会覆盖任何其它的约束路径

2.       FROM THRU TO

3.       FROM TO

4.       管脚指定OFFSETs

5.       Groups OFFSETs（由寄存器或者PADS生产的groups）

6.       全局PERIOD和OFFSETs——最低优先级约束

这里特权同学提醒大家注意的是，通常类似下面这样的计数器绝对不可以归为多周期约束：

         reg[15:0] counter;

         always @ (posedge clk or negedge rst\_n)

         begin

                   if(!rst\_n) counter <= 16’d0;

                   else counter <= counter+1’b1;

         end

虽然我们想想似乎counter[1]也是2个clk变化一次，counter[2]也是4个clk变化一次……但是，我们想想看，如果从counter=1到counter=2没有在一个clk完成，那么肯定就会影响到counter=2到counter=3的变化，对吧？所以，这样的计数器不能算做多周期约束例外。

提纲里描述的多周期例外的计数器应该是这样一个模型：

         reg[15:0] counter;

         always @ (posedge clk or negedge rst\_n)

         begin

                   if(!rst\_n) counter[1:0] <= 2’d0;

                   else counter[1:0] <= counter[1:0]+1’b1;

         end

         always @ (posedge clk or negedge rst\_n)

         begin

                   if(!rst\_n) counter[15:2] <= 14’d0;

                   else if(counter[1:0] == 2’b11) counter[15:2] <= counter[15:2]+1’b1;

         end

         上面两个always块里的数据互不干扰，并且都正常工作，只有下一个always块检测到前一个always块里的counter[1:0]==2’b11时才进位加1。

写到这里，特权发现单从功能上来说，这两个例子是没有差别的，说白了，任何一个计数器都可以建模成后面的形式。或者说，我的问题其实没有说明白，反而被自己的例子给驳倒了。

呵呵，换个角度思考这个问题，到底什么时候是多周期例外，什么时候不是？这个其实还是要看情况的，如果你的系统实时性较高，可能会在某一特定的时钟周期用到16位计数器的计数值（如a = (counter == 16’hffff)?1b’1:1’b0; ），那么这个计数器的高位就不能算作多周期例外。而如果比如在我的一个工程里，有这样的计数器用法：cuonter[2:0]没16个clk的后8个clk需要自增加（从0到15），而高位counter[18:3]当然只有在16个clk变化一次，因为这里counter是用于作为一个地址产生器，也就是说，我的地址是每16个clk的后8个clk用到，那么这里的counter[18:3]就是一个8clk的多周期例外实例。

         说白了，还是要具体问题具体分析。

# 总结

题记：achieving timing closure即达到时序收敛，这是一个很具有挑战性的任务。因为实际的工程项目往往不会像我们用一个资源超大（相对于你的设计来说）的FPGA来做几个数码管串口实验那么简单。设计者往往需要达到成本、速度、资源等各个方面的平衡，即使是一个小设计，有时候也是很费神的。特权同学前几周在饱经ISE4里才有的老器件的折磨后，感慨良多。

## 关于时序报告

**ISE中的时序报告分为两种：**

**– Post-Map Static Timing Report**

**– Post-Place & Route Static Timing Report**

**所谓Post-Map是布局后（没有布线）的静态时序报告，主要用于估计设计的性能，然后提前对设计做一些必要的修改。因为设计的实现（布局布线）是很消耗时间的。Post-Place & Route就是布局布线后的一个比较接近实际板级的一个静态时序报告了，这算是设计者进行时序分析的最终依据。**

## 关于性能估计

**综合报告**

1.       准确的逻辑延时；

2.       基于扇出的布线延时估计

3.       报告的性能是实际的20%误差内

**Post-Map静态时序报告**

1.       准确的逻辑延时

2.       基于最快的可能的布线资源的布线延时估计

3.       使用了60/40规则来计算更趋近于实际的性能估计

**60/40法则：**

1.       这是一个时序约束合理性的经验法则；

2.       打开Post-Map静态时序报告，查看时序报告中关于逻辑延时的百分比

——低于60%，时序很有机会到达时序约束要求

——60%到80%，如果使用了高级选项，时序也很有机会达到时序要求

——80%以上，基本上很难（回到综合部分，或者重新优化你的代码）

**Post-Place & Route时序分析**

         找出时序违规的因素有很多：

1.       设计的综合不当或者代码风格太烂；

2.       糟糕的综合结果（路径中的逻辑太繁杂）

3.       不准确或者不完整的时序约束

4.       糟糕的逻辑映射和布局

每个问题都用不同的解决方案：

1.       重写代码

2.       添加时序约束（注意应该是一些时序例外）

3.       使用不同的软件选项重新综合或者实现

准确的定位时序报告能够解决大多数问题。

**一些可能的问题和解决**

布线延时太长——似乎有些路径扇出很低，但是延时却很大，那么很可能这个地方的布线比较拥堵。解决办法：如果是不相关的逻辑布局到一块，可以到Floorplanner中查看。（这个问题特权同学还没有完全领会也是比较头疼的，希望看到更多更好的资料或者自己在工程中有更多体会时再和大家分享）

         高扇出问题——解决办法是复制高扇出的网络。如果是组合逻辑，那么就比较难了。

         逻辑级数太高——这个问题综合工具无法做太多优化。首先查看是否该路径为多周期路径，如果是，添加多周期例外；使用retiming选项更加均匀的分配触发器之间的逻辑；确定一个比较好的代码技巧被运用到了你的设计代码中；使用流水线设计。

         I/O时序问题——使用DCM移除时钟分布延时； 将输入输出相关的寄存器放入IOB寄存器中。

另外在实际应用中，其实很是有很多可以应用的技巧的，比如实现属性选项里其实是可以设置布局布线的努力程度，还有布局布线的次数等待，对于大多数设计而言这些工具都是有用的。

# Xilinx Timing Analyzer使用札记

虽说当初刚接触FPGA的时候学的是ISE，但是真正深入学习还是在Quartus II平台上。现在项目需要，又得重新来玩转ISE了，虽说QII和ISE很多东西是相通的，不过还是有那么点操作上的差异在里头。用惯了TimeQuest的特权同学折腾起这个Xilinx Timing Analyzer也是一愣一愣的，好不适应。

         花了点时间在Help上，还好这个Help的说明还是蛮到位的。列几个常用的技巧或者说是简单的操作以及特权同学遇到的问题在这里和大家一起学习和探讨。

1.       四种时序分析查看方式。在同等条件下，其实这四种查看方式得到的时序报告结果是一致的，因为它是基于一个时序结果进行的。如图1所示，工具栏上的这四个时序分析按钮分别代表了（从左到右）：

|  |  |
| --- | --- |
| 序号 | 功能 |
| 1 | 根据时序约束产生的时序分析报告。 |
| 2 | 自动产生的时序分析报告。在同等的时序约束条件下，那么该报告的结果肯定和1的结果是大同小异的。 |
| 3 | 用户指定的路径的时序报告，主要是路径端点的时序报告。 |
| 4 | 用户指定的路径的时序报告，主要是时钟或者IO的时序报告。 |

打开Xilinx Floorplanner或Xilinx FPGA Editor后，在详细时序报告中点击任何一条路径或者net等，这里点击了Data Path:dsp\_addr<10> to uut dspc/Mtridata dsp\_out\_dbr\_5（如图2），相应的可以看到其在Floorplanner（如图3）或FPGA Editor中的位置（如图4）。

http://maxim.eefocus.com/data/11-09/1316679383_f40aa98f.jpg

图1

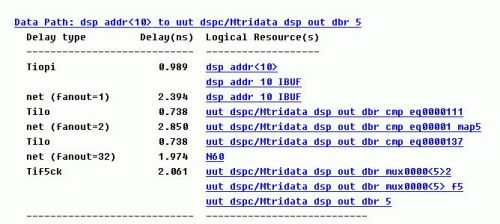
[](http://maxim.eefocus.com/data/11-09/1316679388_b870d0bc.jpg)

图2

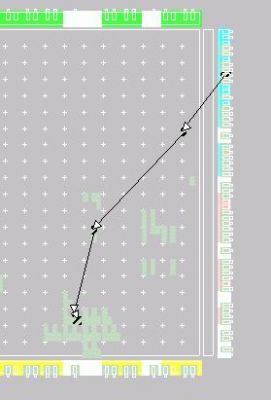
[](http://maxim.eefocus.com/data/11-09/1316679406_58c0e619.jpg)

图3

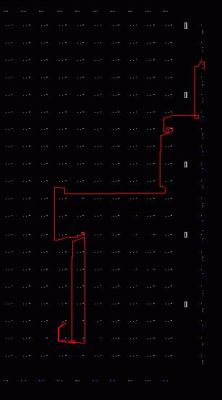
[](http://maxim.eefocus.com/data/11-09/1316679403_af4e138b.jpg)

图4

在查看unconstrained path ayalysis时，遇到了一些问题，clk是FPGA片外晶振输入的时钟，从PAD输入后直接连到DLL，而不直接钟控任何寄存器。但是在unconstrained path报告中却出现了clk到内部所有寄存器的路径。详细的查看这些路径，发现这些路径无外乎是内部DLL输出的两个使用到的时钟的路径。如图5和图6所示。

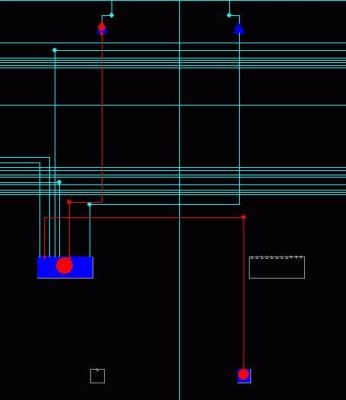
[](http://maxim.eefocus.com/data/11-09/1316679412_481a614b.jpg)

图5

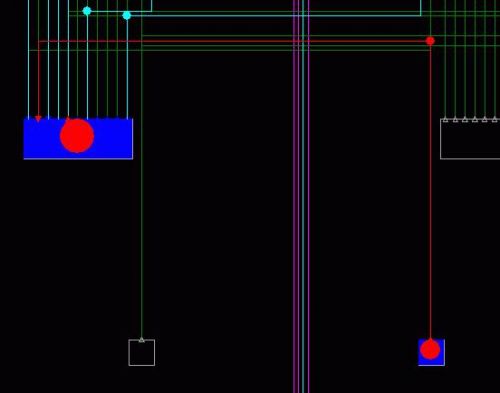
[](http://maxim.eefocus.com/data/11-09/1316679423_93853969.jpg)

图6

         而特权同学希望添加一个clk到all FFT的Flase path，但是

**TIMESPEC "TS\_CLK" = FROM "clk" TO "FFS"  TIG;**

这条语句约束得到的是约束工具的报错信息，因为之前有clk的时钟约束：

**NET "clk" TNM\_NET = "clk";**

**TIMESPEC "TS\_clk" = PERIOD "clk" 40 ns HIGH 50 %;**

         不知道他们之间是否有冲突，虽然最后的unconstrained报告大可不必介意，但是这个false路径总应该有个约束的办法，只是特权同学现在还没有搞定其中的问题所在。