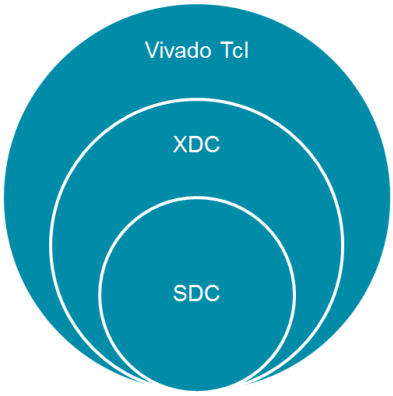
XDC 约束技巧之时钟篇

Xilinx©的新一代设计套件 Vivado 中引入了全新的约束文件 XDC，在很多规则和技巧上都跟上一代产品 ISE 中支持的 UCF 大不相同，给使用者带来许多额外挑战。Xilinx 工具专家告诉你，其实用好 XDC 很容易，只需掌握几点核心技巧，并且时刻牢记：XDC 的语法其实就是 Tcl 语言。

XDC 的优势

XDC 是 Xilinx Design Constraints 的简写，但其基础语法来源于业界统一的约束规范 SDC（最早由 Synopsys 公司提出，故名 Synopsys Design Constraints）。所以 SDC、XDC 跟 Vivado Tcl 的关系如下图所示。



XDC 的主要优势包括：

1. 统一了前后端约束格式，便于管理；
2. 可以像命令一样实时录入并执行；
3. 允许增量设置约束，加速调试效率；
4. 覆盖率高，可扩展性好，效率高；
5. 业界统一，兼容性好，可移植性强；

XDC 在本质上就是 Tcl 语言，但其仅支持基本的 Tcl 语法如变量、列表和运算符等等，对其它复杂的循环以及文件 I/O 等语法可以通过在 Vivado 中 source 一个 Tcl 文件的方式来补充。（对 Tcl 话题感兴趣的读者可以参考作者的另一篇文章《Tcl 在 Vivado 中的应用》）XDC 与 UCF 的最主要区别有两点：

1. XDC 可以像 UCF 一样作为一个整体文件被工具读入，也可以在实现过程中被当作一个个单独的命令直接执行。这就决定了 XDC 也具有 Tcl 命令的特点，即后面输入的约束在有冲突的情况下会覆盖之前输入的约束（时序例外的优先级会在下节详述）。另外，不同于 UCF 是全部读入再处理的方式，在 XDC 中，约束是读一条执行一条，所以先后**顺序很重要**，例如要设置 IO 约束之前，相对应的 clock 一定要先创建好。

2. UCF 是完全以 FPGA 的视角看问题，所以缺省认为所有的时钟之间除非预先声明是同步的，否则就视作异步而不做**跨时钟域时序分析**；XDC 则恰恰相反，ASIC 世界的血缘背景决定了在其中，所有的时钟缺省视作全同步，在没有时序例外的情况下，工具会主动分析每一条跨时钟域的路径。

XDC 的基本语法

XDC 的基本语法可以分为时钟约束、I/O 约束以及时序例外约束三大类。根据 Xilinx 的 UltraFast 设计方法学中 Baseline 部分的建议（**UG949** 中有详细介绍），对一个设计进行约束的先后顺序也可以依照这三类约束依次进行。本文对可以在帮助文档中查到的基本 XDC 语法不做详细解释，会将重点放在使用方法和技巧上。

时钟约束

时钟约束必须最早创建。对 7 系列 FPGA 来说，端口进来的时钟以及 GT 的输出 RXCLK/TXCLK 都必须由用户使用 create_clock 自主创建为主时钟。如果是差分输入的时钟，可以仅仅在差分对的 P 侧用 get_ports 获取端口，并使用 create_clock 创建。例如，

```
create_clock -name clk_200 -period 5 [get_ports clk200_p]
```

Vivado 自动推导的衍生时钟

MMCM/PLL/BUFR 的输出作为衍生时钟，可以由 Vivado 自动推导，无需用户创建。自动推导的好处在于当 MMCM/PLL/BUFR 的配置改变而影响到输出时钟的频率和相位时，用户无需改写约束，Vivado 仍然可以自动推导出正确的频率/相位信息。劣势在于，用户并不清楚自动推导出的衍生钟的名字，当设计层次改变时，衍生钟的名字也有可能改变。这样就会带来一个问题：用户需要使用这些衍生钟的名字来创建 I/O 约束、时钟关系或是时序例外等约束时，要么不知道时钟名字，要么时钟名字是错的。

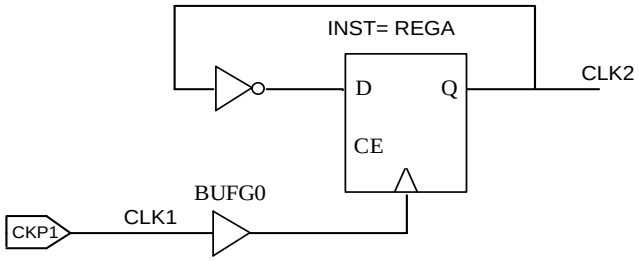
```
create_generated_clock -name my_clk_name  [get_pins  mmcm0/CLKOUT] \
                       -source [get_pins  mmcm0/CLKIN] \
                       -master_clock  main_clk
```

推荐的做法是，由用户来指定这类衍生时钟的名字，其余频率等都由 Vivado 自动推导。这样就只需写明 create_generated_clock 的三个 option，其余不写即可。如上所示。

当然，此类情况下用户也可以选择完全由自己定义衍生时钟，只需补上其余表示频率/相位关系的 option，包括-multiply_by 、-devide_by 等等。需要注意的是，一旦 Vivado 在 MMCM/PLL/BUFR 的输出检测到用户自定义的衍生时钟，就会报告一个 Warning，提醒用户这个约束会覆盖工具自动推导出的衍生时钟（例外的情况见文章下半段重叠时钟部分的描述），用户须保证自己创建的衍生钟的频率等属性正确。

用户自定义的衍生时钟

工具不能自动推导出衍生钟的情况，包括使用寄存器和组合逻辑搭建的分频器等，必须由用户使用 create_generated_clock 来创建。举例如下，




```
create_clock -name clk1 -period 4 [get_ports CKP1]
create_generated_clock   -name clk2  [get_pins REGA/Q]  \
                              -source [get_ports CKP1] -divide_by 2
```

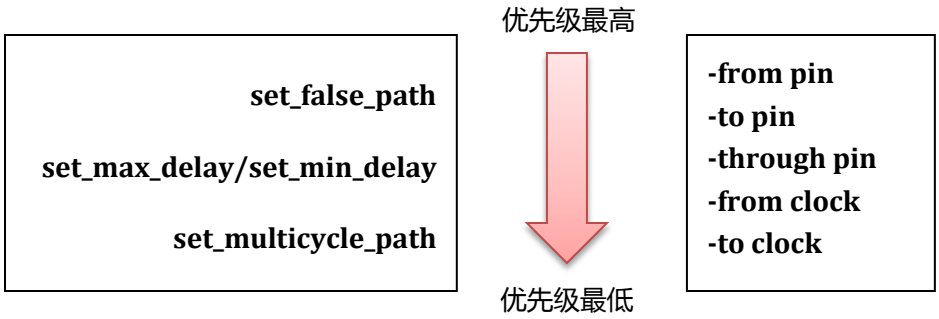
I/O 约束

在设计的初级阶段，可以不加 I/O 约束，让工具专注于满足 FPGA 内部的时序要求。当时序要求基本满足后，再加上 I/O 约束跑实现。XDC 中的 I/O 约束有以下几点需要注意：

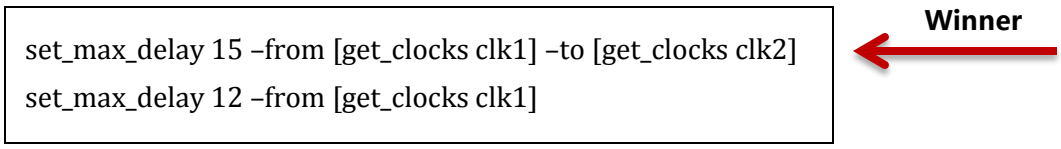
1. 不加任何 I/O 约束的端口时序要求被视作无穷大。
2. XDC 中的 set_input_delay / set_output_delay 对应于 UCF 中 OFFSET IN / OFFSET OUT，但视角相反。OFFSET IN / OFFSET OUT 是从 FPGA 内部延时的角度来约束端口时序，set_input_delay / set_output_delay 则是从系统角度来约束。
3. 典型的 I/O 时序，包括系统同步、源同步、SDR 和 DDR 等等，在 Vivado 图形界面的 XDC templates 中都有示例。2014.1 版后还有一个 Timing Constraints Wizard 可供使用。

时序例外约束

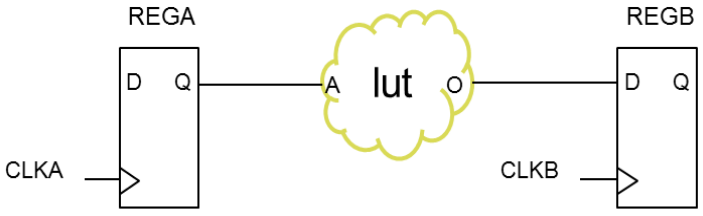
时序例外约束包括 set_max_delay/set_min_delay，set_multicycle_path，set_false_path 等，这类约束除了要满足 XDC 的先后顺序优先级外，还受到自身优先级的限制。一个总的原则就是针对同一条路径，对约束目标描述越具体的优先级越高。不同的时序例外约束以及同一约束中不同条件的优先级如下所示：

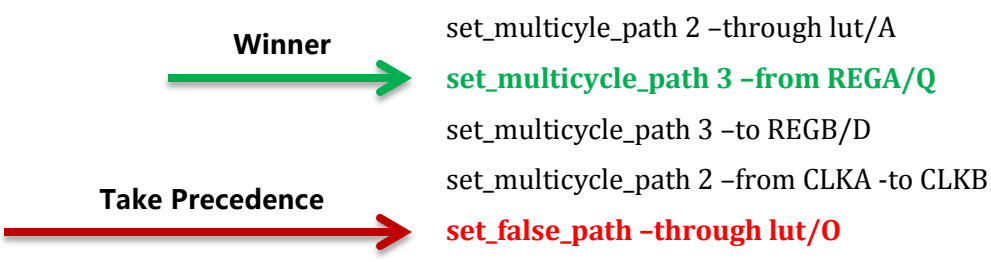


举例来说，依次执行如下两条 XDC，尽管第二条较晚执行，但工具仍然认定第一条约束设定的 15 为 clk1 到 clk2 之间路径的 max delay 值。



再比如，对图示路径依次进行如下四条时序例外约束，优胜者将是第二条。但如果再加入最后一条约束，false path 的优先级最高，会取代之前所有的时序例外约束。



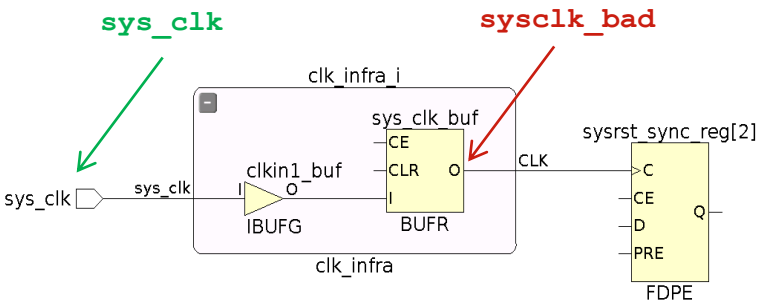


高级时钟约束

约束最终是为了设计服务，所以要用好 XDC 就需要深入理解电路结构和设计需求。接下来我们就以常见 FPGA 设计中的时钟结构来举例，详细阐述 XDC 的约束技巧。

时序的零起点

用 create_clock 定义的主时钟的起点即时序的“零起点”，在这之前的上游路径延时都被工具自动忽略。所以主时钟创建在哪个“点”很重要，以下图所示结构来举例，分别于 FPGA 输入端口和 BUFG 输出端口创建一个主时钟，在时序报告中体现出的路径延时完全不同，很明显 sysclk_bad 的报告中缺少了之前一段的延时，时序报告不可信。

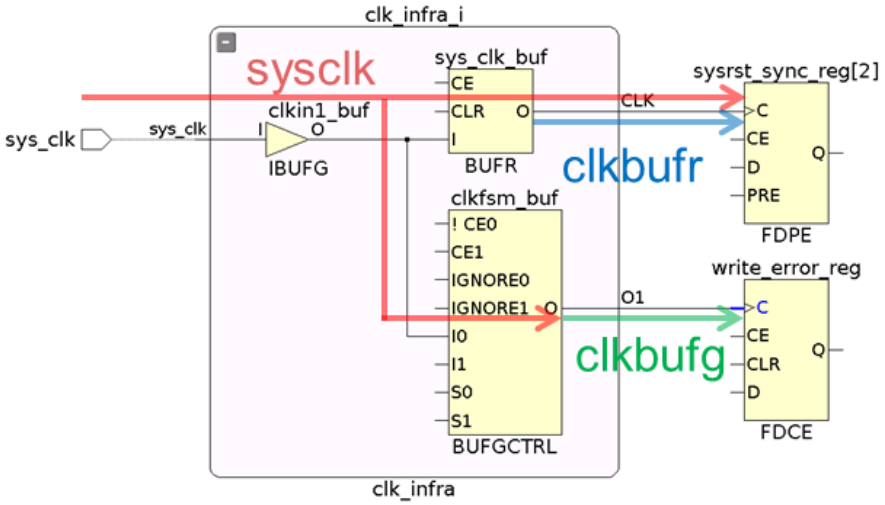



```
create_clock -name sysclk -period 10 [get_ports sys_clk]
Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
-------------------------------------------    -------------------
(clock sysclk rise edge)     0.000     0.000 r
                             0.000     0.000 r  sys_clk
net (fo=0)                   0.000     0.000    clk_infra_i/sys_clk
IBUFG (Prop_ibufg_I_O)       0.766     0.766 r  clk_infra_i/clkin1_buf/O
net (fo=3, unplaced)         1.109     1.875    clk_infra_i/clkin1
BUFR (Prop_bufr_I_O)         0.314     2.189 r  clk_infra_i/sys_clk_buf/O
net (fo=10, unplaced)        1.109     3.298    sys_clk_int
                                             r  sysrst_sync_reg[2]/C


create_clock -name sysclk_bad -period 10 [get_pins clk_infra_i/sys_clk_buf/O]
Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
-------------------------------------------    -------------------
(clock sysclk_bad rise edge)
                             0.000     0.000 r
BUFR                         0.000     0.000 r  clk_infra_i/sys_clk_buf/O
net (fo=10, unplaced)        1.109     1.109    sys_clk_int
                                             r  sysrst_sync_reg[2]/C
```

时钟定义的先后顺序

时钟的定义也遵从 XDC/Tcl 的一般优先级，即：在同一个点上，由用户定义的时钟会覆盖工具自动推导的时钟，且后定义的时钟会覆盖先定义的时钟。若要二者并存，必须使用 **-add** 选项。



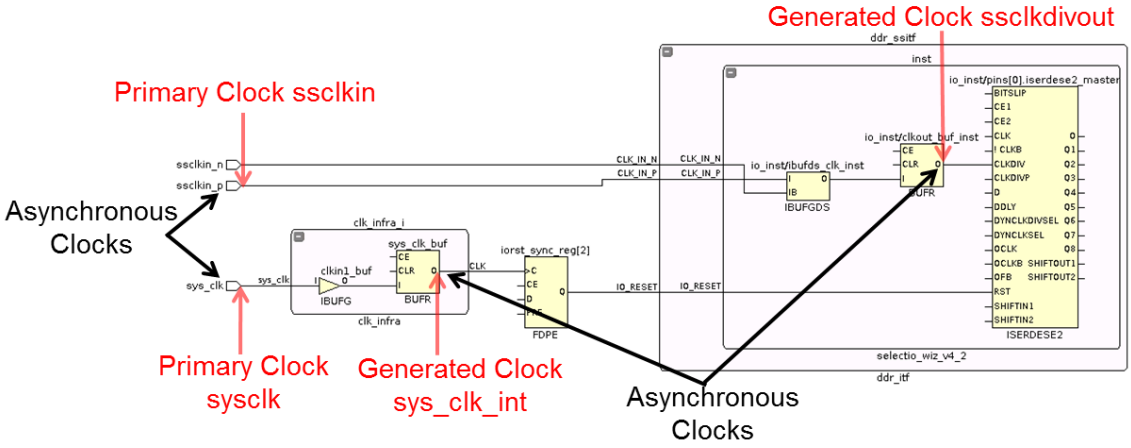

```
create_clock -name sysclk -period 10 [get_ports sys_clk]
create_generated_clock -name clkbufg -source [get_ports sys_clk] -divide_by 1 \
            [get_pins clk_infra_i/clkfsm_buf/O]
create_generated_clock -name clkbufr -source [get_ports sys_clk] -divide_by 1 \
            [get_pins clk_infra_i/sys_clk_buf/O] -add -master_clock sysclk
```

上述例子中 BUFG 的输出端由用户自定义了一个衍生钟 clkbufg，这个衍生钟便会覆盖此处原有的 sysclk。此外，图示 BUFR 工作在 bypass 模式，其输出不会自动创建衍生钟，但在 BUFR 的输出端定义一个衍生钟 clkbufr，并使用 -add 和 -master_clock 选项后，这一点上会存在 sysclk 和 clkbufg 两个重叠的时钟。如下的 Tcl 命令验证了我们的推论。

```
% get_clocks -of [get_pins sysrst_sync_reg[2]/C]
sysclk clkbufr

% get_clocks -of [get_pins write_error_reg/C]
clkbufg
```

同步时钟和异步时钟



不同于 UCF 约束，在 XDC 中，所有的时钟都会被缺省认为是相关的，也就是说，网表中所有存在的时序路径都会被 Vivado 分析。这也意味着 FPGA 设计人员必须通过约束告诉工具，哪些路径是无需分析的，哪些时钟域之间是异步的。

如上图所示，两个主时钟 ssclkin 和 sysclk 由不同的端口进入 FPGA，再经由不同的时钟网络传递，要将它们设成异步时钟，可以使用如下约束：

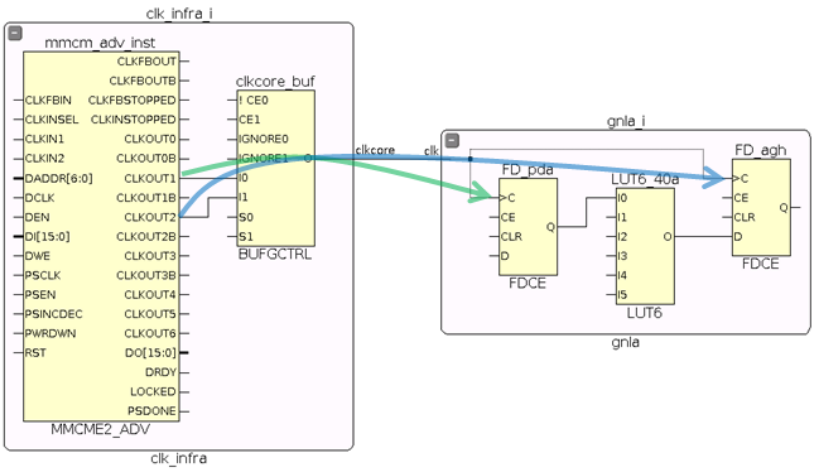
```
set_clock_groups -name sys_ss_async -asynchronous \
                 -group [get_clocks -include_generated_clocks sysclk] \
                 -group [get_clocks -include_generated_clocks ssclkin]
```

其中，**-include_generated_clocks** 表示所有衍生钟自动跟其主时钟一组，从而与其它组的时钟之间为异步关系。不加这个选项则仅仅将时钟关系的约束应用在主时钟层面。

重叠（单点多个）时钟

重叠时钟是指多个时钟共享完全相同的时钟传输网络，例如两个时钟经过一个 MUX 选择后输出的时钟，在有多种运行模式的设计中很常见。

如下图所示，clk125 和 clk250 是 clkcore_buf 的两个输入时钟，不约束时钟关系的情况下，Vivado 会对图示路径做跨时钟域（重叠时钟之间）分析。这样的时序报告即便没有违例，也是不可信的，因为 clk125 和 clk250 不可能同时驱动这条路径上的时序元件。这么做也会增加运行时间，并影响最终的实现效果。



Different clocks

Same clock tree

```
Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
-------------------------------------------    -------------------
(clock clk125_ rise edge)  0.000     0.000 r
                           0.000     0.000 r  sys_clk
...
BUFGCTRL (Prop_bufgct...)  0.093    -3.307 r  clk_infra_i/clkcore_buf/O
net (fo=4900, unplaced)    1.109    -2.198    gnla_i/clk
                                          r  gnla_i/FD_pda/C
-------------------------------------------    -------------------
FDCE (Prop_fdce_C_Q)       0.249    -1.949 r  gnla_i/FD_pda/Q
...
FDCE (Setup_fdce_C_D)     -0.003    -1.092    gnla_i/FD_agh/D
-------------------------------------------
(clock clk250_ rise edge)  4.000     4.000 r
                           0.000     4.000 r  sys_clk
...
BUFGCTRL (Prop_bufgct...)  0.083     1.416 r  clk_infra_i/clkcore_buf/O
net (fo=4900, unplaced)    1.109     2.525    gnla_i/clk
                                          r  gnla_i/FD_agh/C
clock pessimism           -0.733     1.792
clock uncertainty         -0.191     1.601
-------------------------------------------
required time                        1.601
arrival time                         1.092
-------------------------------------------
slack                                2.693
```

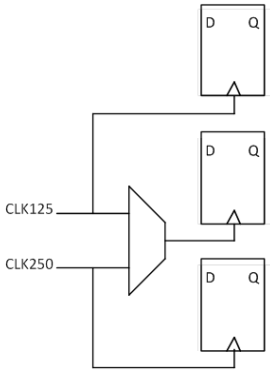
如果 clk125 和 clk250 除了通过 clkcore_buf 后一模一样的扇出外没有驱动其它时序元件，我们要做的仅仅是补齐时钟关系的约束。

```
set_clock_groups -physically_exclusive \
                 -group [get_clocks clk125] \
                 -group [get_clocks clk250]
```

在很多情况下，除了共同的扇出，其中一个时钟或两个都还驱动其它的时序元件，此时建议的做法是在 clkcore_buf 的输出端上创建两个重叠的衍生钟，并将其时钟关系约束为**-physically_exclusive** 表示不可能同时通过。这样做可以最大化约束覆盖率，也是 ISE 和 UCF 中无法做到的。

```
create_generated_clock -name clk125_bufgctrl \
            -divide_by 1 [get_pins bufgctrl_i/O] \
            -source [get_ports bufgctrl_i/I0]
create_generated_clock -name clk250_bufgctrl \
            -divide_by 1 [get_pins bufgctrl_i/O] \
            -source [get_ports bufgctrl_i/I1] \
            -add -master_clock clk250

set_clock_groups -physically_exclusive \
                 -group clk125_bufgctrl \
                 -group clk250_bufgctrl
```



其它高级约束

时钟的约束是 XDC 的基础，熟练掌握时钟约束，也是 XDC 约束技巧的基础。其它高级约束技巧，包括复杂的 CDC（Clock Domain Crossing）约束和接口时序（SDR、DDR、系统同步接口和源同步接口）约束等方面还有很多值得注意的地方。

这一系列《XDC 约束技巧》文章还会继续就上述所列方向分篇详述，敬请关注作者的后续更新，以及 Xilinx 官方网站和中文论坛上的更多技术文章。

Ally Zhou 2014-9-25 于 Xilinx 上海 Office

《XDC 约束技巧之时钟篇》 版本更新

日期	版本	作者	备注
2014-09-25	1.1	Ally Zhou	初始版本，发布于 Xilinx 中文论坛 Vivado 专区
2014-11-27	1.2	Ally Zhou	修订笔误； XDC 的基本语法－时钟约束： 增加差分时钟约束； 增加衍生时钟语法举例。