UltraFast 设计方法时序收敛快捷参考指南 (UG1292)

引言

根据《UltraFAST 设计方法指南（适用于 Vivado Design Suite）》 (UG949) 中的建议，本快捷参考指南提供了以下简化的分步骤快速收敛时序流程：

- **初始设计检查：**在实现设计前审核资源利用率、逻辑层次和时序约束。
- **时序基线：**在每个实现步骤后检查并解决时序违规，从而帮助布线后收敛时序。
- **时序违规解决：**识别建立时间违规或保持时间违规的根源，并解决时序违规。

Failfast 报告

基于 Tcl 的 Failfast 报告总结关于设计与约束的关键信息，有助于您迅速识别并解决常见的实现和性能问题。默认条件下该报告分析完整设计并输出一个表，在表中将每个分析指标与典型的指南值做对比。将每个不符合指南值的指标标记为 REVIEW。报告包括下列部分：

- 设计特性
- 关键时钟方法检查
- 根据目标 Fmax 进行保守的逻辑层次评估

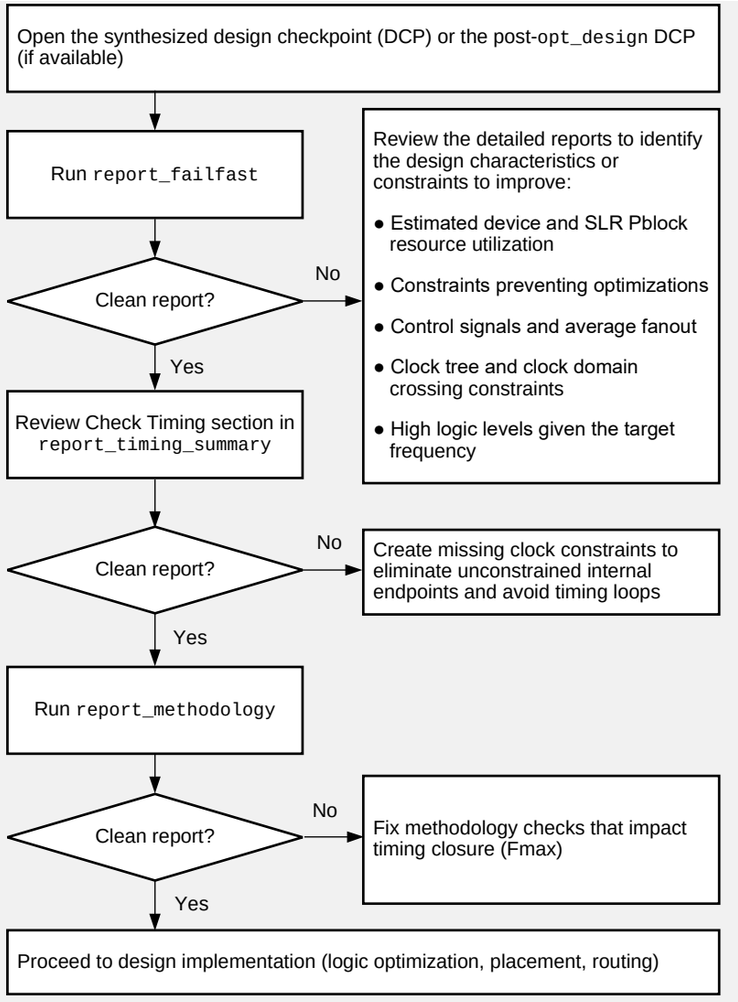
从 Vivado® 工具 2018.1 版开始，report_failfast 脚本能默认安装并以下列方式调用：

xilinx::designutils::report_failfast

从 SDx™ 工具 2018.2 版开始，report_failfast 在编译流程中使用 xocc –R 1 或 xocc –R 2 调用。

如需了解更多有关 report_failfast 的信息，请参阅 **Failfast 报告概览**（第 10 页）。

初始设计检查流程



初始设计检查详细介绍

虽然在赛灵思器件上实现设计是一个自动化程度相当高的任务，获得较高性能并解决时序或布线违规带来的编译问题，是一项复杂且耗时的工作。根据简单的日志消息或由工具生成的实现后时序报告可能难以确定故障原因。因此有必要采用逐步设计开发和编译方法，包括审核中间结果以确保设计能进行到下一个实现步骤。

第一步是确保所有的初始设计检查都已经完成。在下列层次查看这些检查：

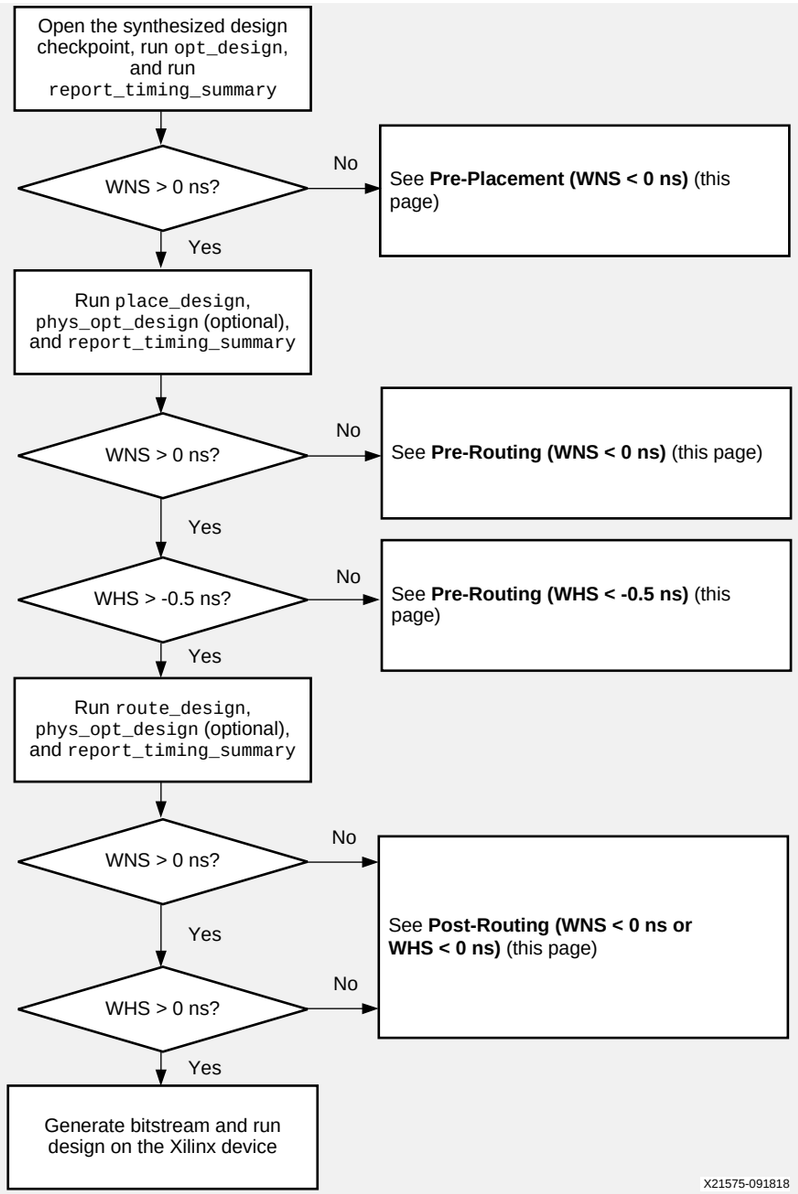
- 每个由定制 RTL 构成的内核或由 Vivado HLS 生成的内核

  注意：检查目标时钟频率约束是否现实。
- 每个与子系统对应的主要层级，例如有多个内核、IP 块和连接逻辑的 Vivado IP 集成器原理图
- 包括所有主要功能和层级、I/O 接口、完整时钟电路和物理及时序约束的完整设计

如果设计使用布局规划约束，如超级逻辑区域 (SLR) 分配或分配给 Pblock 的逻辑，为每个物理约束审核估算的资源占用，确保符合资源占用指南。在 Failfast 报告中查看默认指南。使用下列命令生成报告：

- report_utilization –pblocks <pblockName>
- report_failfast –pblock <pblockName>
- report_failfast [–slr SLR*n* | -by_slr]

时序基线流程



时序基线示例

时序基线检查的目的是通过在每个实现步骤后分析和解决时序问题，确保设计满足时序要求。在编译流程中尽早纠正设计问题和约束问题，能带来更广泛的影响并实现更高的性能。可通过创建如下中间报告，在进行下一个步骤前检查并解决时序违规：

Vivado 项目模式报告	Vivado 非项目模式报告	Sdx 工具报告
使用 UltraFast™ 设计方法或时序收敛报告策略	在每个实现步骤后添加下列报告命令： ▪ report_timing_summary ▪ report_methodology ▪ report_failfast	使用 xocc -R 1 或 xocc -R 2 选项在下列目录中生成 Failfast 报告、中间时序报告和 DCP： <runDir>/_x/link/vivado/prj/prj.runs/impl_1

预布局 (WNS < 0ns)

在 place_design 之前，时序报告反映的是假定每条逻辑路径的逻辑布局尽可能最优情况下的设计性能。必须通过采用"初始检查"建议来解决建立时间违规。

预布线 (WNS < 0ns)

在 route_design 之前，时序报告反映的是假定每个单独网络存在最理想布线延迟，存在一些扇出损失且不考虑保持时间修复的影响（网络布线绕行）或拥塞时的设计性能。建立时间违规往往是下列原因导致的欠佳布局造成的：(1) 高器件或 SLR 资源利用率；(2) 因复杂逻辑连接造成的布局拥塞；(3) 大量时序路径有过高的逻辑级数；和 (4) 不平衡时钟间的高时钟偏差或高时钟不确定性。在 Explore 模式或 AggressiveExplore 模式下运行 phys_opt_design，试图提升 post-place_design QoR。如果不成功，先重点提升布局 QoR。

预布线 (WHS < -0.5ns)

如果布线后未达成性能目标且布线前最差负时序裕量 (WNS) 为正值，尽量减少估算的最差保持时序裕量 (WHS) 重大违规。减少并减轻预布线保持时间违规有助于 route_design 重点关注 Fmax 而非修复保持时间违规。

布线后 (WNS < 0 ns 或 WHS < 0 ns)

在 route_design 后，首先通过审核日志文件或在布线后设计检查点 (DCP) 上运行 report_route_status，验证设计是否完全布线。布线违规和重大建立时间 (WNS) 或保持时间 (WHS) 违规是由高拥塞导致的结果。使用**分析建立时间违规**（第 3 页）、**解决保持时间违规**（第 4 页）以及**拥塞减少方法**（第 6 页），识别和实现解决步骤。

在 route_design 后尝试运行 phys_opt_design，解决 > -0.200 ns 的轻微建立时间违规。

在迭代设计、约束和编译策略时，保持在每个步骤后追踪 QoR，包括拥塞信息。使用 QoR 表比较运行特性，并确定在解决剩余的时序违规时的优先关注对象。

	opt_design		place_design			phys_opt_design				route_design				
Impl. Run	Directive	WNS	Directive	WNS	Congestion	Directive	WNS	WHS	THS	Directive	WNS	TNS	WHS	Congestion
Run1	ExploreWithRemap	0.034	WLDrivenBlockPlacement	-0.07	5-4-5-5	Explore	0.001	-0.409	-851.052	NoTimingRelaxation	-0.02	-1.68	0.006	5-5-4-5
Run2	Explore	0.054	AltSpreadLogic_medium	-0.368	6-5-5-5	Explore	-0.068	-0.364	-852.889	Default	-1.50	-3680.32	0.003	5-6-5-6
Run3	Default	0.054	AltSpreadLogic_high	-0.393	5-4-5-5	Explore	0.035	-0.364	-906.036	Explore	-1.37	-1495.19	0.006	4-5-5-6
Run4	Default	0.054	ExtraTimingOpt	-0.41	5-5-5-5	Explore	0.075	-0.407	-902.348	Explore	-1.23	-2896.42	0.001	5-5-5-6

提示：在 place_design 和 route_design 后使用 report_qor_suggestions，自动识别帮助新编译提升 QoR 的设计、约束和工具选项更改。

分析建立时间违规流程

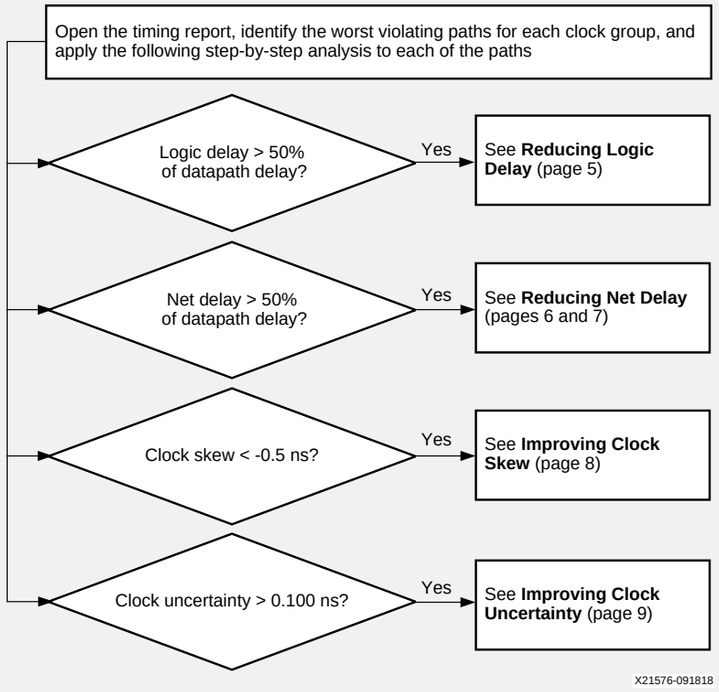
设计性能由如下因素决定：

- **时钟偏差与时钟不确定性**：时钟如何高效实现
- **逻辑延迟**：时钟周期内遍历的逻辑量
- **网络或布线延迟**：Vivado 如何高效实现设计的布局和布线

使用时序路径或设计分析报告内的信息，完成：

- 识别哪些因素是引起时序违规的主要因素
- 确定如何迭代提升 QoR

提示：需要时在每个步骤后打开 DCP 以生成更多报告。



在报告中找出建立时序路径特性

在 Vivado 项目模式中，通过下面的步骤找到建立时序路径特性：

1. 在“Design Runs”窗口中，选择要分析的实现运行。
2. 在“Run Properties”窗口中，选择**“Reports”**标签。

打开选定实现步骤的时序总结报告或设计分析报告：

- **时序总结报告**：<runName>_<flowStep>_report_timing_summary (针对文本采用 .rpt 或针对 Vivado IDE 采用 .rpx)
- **设计分析报告**：<runName>_<flowStep>_report_design_analysis

在 Vivado 非项目模式或 SDx 工具中，采取下列操作之一：

- 打开实现运行目录中的报告。
- 打开 Vivado IDE 中的实现 DCP，然后打开报告的 RPX 版本。

注意：使用 Vivado IDE 可便于您在报告、原理图和器件窗口间进行交叉探测。

对于每个时序路径，逻辑延迟、布线延迟、时钟偏差和时钟不确定性特性都位于路径的报头内：

Summary	
Name	Path 4
Slack	-0.675ns
Source	inst_209033/inst_209021/inst_200956/inst_200920/inst_
Destination	inst_209033/inst_209021/inst_200956/inst_200920/inst_
Path Group	app_clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	3.184ns (app_clk rise@3.184ns - app_clk rise@0.000ns)
Data Path Delay	3.505ns (logic 1.283ns (36.605%) route 2.222ns (63.395%))
Logic Levels	10 (CARRY8=2 LUT2=1 LUT4=1 LUT5=1 LUT6=5)
Clock Path Skew	-0.333ns
Clock Uncertainty	0.046ns

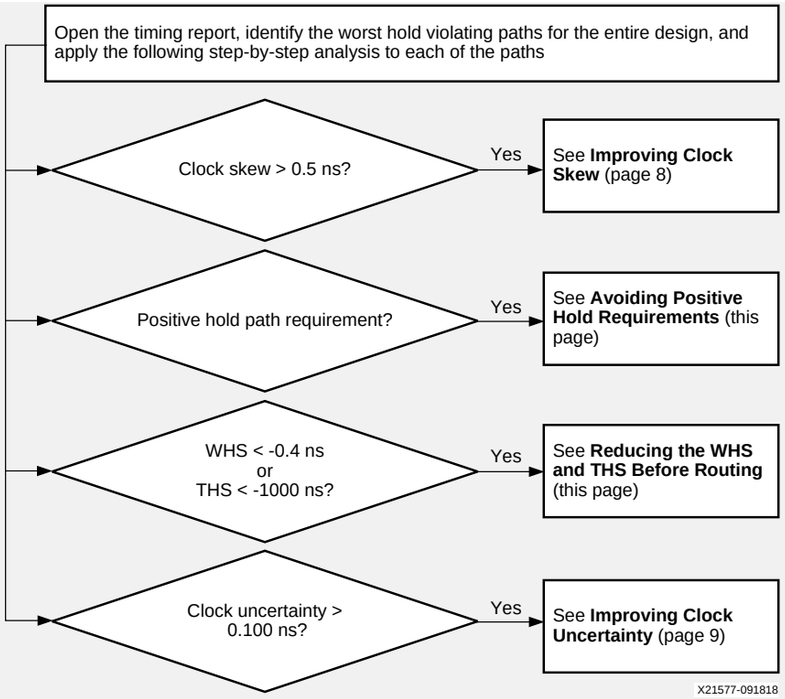
```
Slack (VIOLATED) :        -0.675ns  (required time - arrival time)
  Source:                 inst_209033/inst_209021/inst_200956/inst_200920/inst_191361/
                            (rising edge-triggered cell FDRE clocked by app_clk  {rise
  Destination:            inst_209033/inst_209021/inst_200956/inst_200920/inst_188337/
                            (rising edge-triggered cell FDRE clocked by app_clk  {rise
  Path Group:             app_clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            3.184ns  (app_clk rise@3.184ns - app_clk rise@0.000ns)
  Data Path Delay:        3.505ns  (logic 1.283ns (36.605%)  route 2.222ns (63.395%))
  Logic Levels:           10   (CARRY8=2 LUT2=1 LUT4=1 LUT5=1 LUT6=5)
  Clock Path Skew:        -0.333ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.884ns = ( 6.068 - 3.184 )
    Source Clock Delay      (SCD):    3.380ns
    Clock Pessimism Removal (CPR):    0.163ns
  Clock Uncertainty:      0.046ns  ((TSJ^2 + DJ^2)^1/2) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Discrete Jitter          (DJ):    0.060ns
    Phase Error              (PE):    0.000ns
```

除了时钟的不确定性，设计分析报告的“Setup Path Characteristics”也提供相同的时序路径特性：

Name	Slack	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Logic Levels	Routes	Logical Path
Path 1	-1.438	1.592	3.244	6%	94%	0.665	1	2	FDRE LUT4 FDRE
Path 2	-0.708	3.184	3.508	43%	57%	-0.362	5	5	RAMB18E2 LUT6 LUT6 LUT6 LUT6 LUT6 FDRE
Path 3	-0.683	3.184	3.483	42%	58%	-0.362	5	5	RAMB18E2 LUT6 LUT6 LUT6 LUT6 LUT6 FDRE
Path 4	-0.675	3.184	3.505	37%	63%	-0.333	10	8	FDRE LUT6 LUT6 LUT6 LUT5 LUT6 LUT2 CARRY8 CARRY8 LUT4 LUT6 FDRE

提示：在文本模式中，“Setup Path Characteristics”列的全部列都会显示，让表宽度非常宽。在 Vivado IDE 里，同一表中通过减少显示的列数来帮助实现可视化。右键单击表头，根据需要启用或禁用列。例如，默认条件下 DONT_TOUCH 或 MARK_DEBUG 列是不可见的。启用这些列可以查看逻辑优化分析中跳过的重要信息，而且这些信息用其他方式不便查看。

解决保持时间违规流程



下面是一个有高时钟偏差的保持时序路径示例：

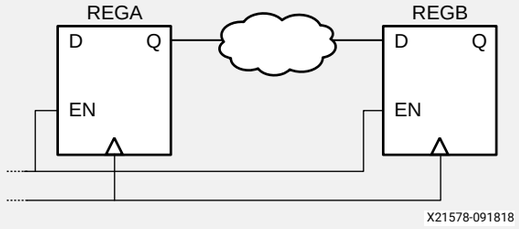
Summary	
Name	Path 259
Slack (Hold)	-1.129ns
Source	inst_209033/inst_381/inst_285879/inst_285870/inst_285584/i
Destination	inst_209033/inst_381/inst_285879/inst_285870/inst_285584/i
Path Group	app_clk
Path Type	Hold (Min at Slow Process Corner)
Requirement	0.000ns (app_clk rise@0.000ns - txoutclk_out[3]_3 rise@0.000ns)
Data Path Delay	0.180ns (logic 0.059ns (32.778%) route 0.121ns (67.222%))
Logic Levels	0
Clock Path Skew	1.247ns

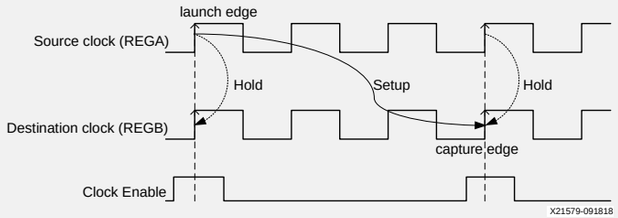
解决保持时间违规的方法

避免上升沿保持时间要求

在使用多周期路径约束放松建立时间检查时，您必须：

- 调整同一路径上的保持时间检查，让保持时间分析使用相同的启动沿和捕获沿。如果未能这样操作，会导致上升沿保持时间要求（一个或多个时钟周期）和无法实现的时序收敛。
- 指定端点引脚，而不仅仅是单元或时钟。例如，端点单元 REGB 有三个输入引脚：C、EN 和 D。只有 REGB/D 引脚需要由多周期路径例外处理进行约束（时钟使能 (EN) 引脚不用），因为 EN 引脚在每个时钟周期都会发生变化。如果将约束连接至单元而不是引脚，那么所有有效的端点引脚，包括 EN 引脚，都在约束的考虑范围内。





赛灵思建议您始终使用如下语句：

```
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -setup 3
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -hold 2
```

在布线前降低 WHS 和 THS

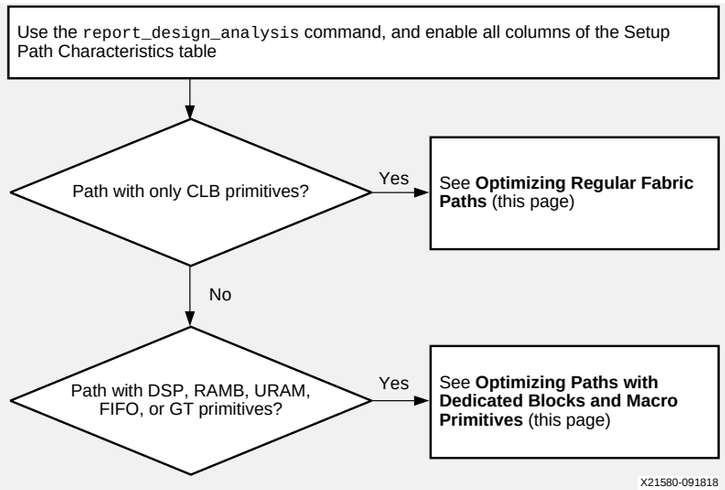
重大的预估保持时间违规会增大布线难度，并且不能总是用 route_design 解决。布局后 phys_opt_design 命令提供多种保持时间修复选项：

- 在时序元素之间插入上升沿边缘触发寄存器将时序路径分为两个半周期路径，从而显著减少保持时间违规。这种优化只能在不劣化建立时序的情况下执行。使用下列命令：

  ```
  phys_opt_design -insert_negative_edge_ffs
  ```

- 插入 LUT1 以延迟数据路径，在不造成建立时间违规的情况下减少保持时间违规。使用下列命令：
  - phys_opt_design -hold_fix：针对最大 WHS 违规在路径上插入 LUT1。
  - phys_opt_design -aggressive_hold_fix：以 LUT 利用率显著增加和编译时间延长为代价，在更多路径上插入 LUT1 以大幅降低总体保持时序裕量 (THS)。该选项可以与任何 phys_opt_design 指令结合。
    - phys_opt_design -directive ExploreWithAggressiveHoldFix：插入 LUT1，在使用一切其他设计用于提升 Fmax 的物理优化的同时修复保持时间。

减少逻辑延迟流程



Vivado 实现首先关注最关键的路径。这意味着不太困难的路径在布局之后或布线之后变为关键路径。赛灵思建议在综合之后或 opt_design 之后识别和改进最长路径，因为它对 QoR 产生最大的影响，并且通常会显著减少布局和布线迭代的次数以实现时序收敛。使用 report_design_analysis"逻辑层次分布"表，通过根据要求权衡逻辑层次分布，找出需要改进设计的时钟域。要求越低，允许的逻辑层次就越少。例如在下面的预布局逻辑层次分配报告中：

- 为 txoutclk_out[0]_4 审核有 8 个或更多逻辑层次的全部路径。
- 为 app_clk 审核有 11 个或更多逻辑层次的全部路径。

Logic Level Distribution

End Point Clock	Requirement	0	1	2	4	5	6	7	8	9	10	11	12	13	14	15	16
app_clk	3.184ns	0	0	0	1	0	0	135	16	37	30	16	16	16	16	15	7
txoutclk_out[0]_4	2.388ns	2	0	0	64	784	1677	0	9	3	0	3	4	0	0	0	0
txoutclk_out[3]_3	1.592ns	2100	5029	20	0	0	0	0	0	0	0	0	0	0	0	0	0

注释：级联 CARRY 或 MUXF 单元能够人为增加逻辑层次数量，且对延迟影响小。

提示：在 Vivado IDE 报告中，点击逻辑层次编号以选择路径，然后按 F4 生成原理图并审核逻辑。

减少逻辑延迟的方法

提示：使用 report_qor_suggestions 自动识别常见的逻辑延迟降低方法，并生成设计微调约束，便于您在您下一次的实现运行中使用。

优化常规结构路径

常规时序路径是寄存器 (FD*) 或移位寄存器 (SRL*) 之间的路径，并且穿过 LUT、MUXF 和 CARRY 的混合。如果您遇到与常规架构路径有关的问题，赛灵思建议如下。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：综合》 (UG901) 和《Vivado Design Suite 用户指南：实现》 (UG904)。

- 输入较少的级联 LUT (LUT1-LUT4) 可以相互合并来减少 LUT 使用量。但是设计层级、网络扇出、或者一些 KEEP/KEEP_HIERARCHY/DONT_TOUCH/MARK_DEBUG 属性会限制工具进行 LUT 合并。
  建议：移除属性，从综合步骤或从 opt_design -remap 开始重新运行。
- 单个 CARRY（非级联）单元限制 LUT 优化，能降低布局质量。
  建议：使用 FewerCarryChains 综合指令，或设置单元上的 CARRY_REMAP 属性为可用 opt_design 移除。
- 移位寄存器的 SRL* 延迟高于寄存器 FD* 延迟，且 SRL 布局的质量低于 FD 布局质量。
  建议：在综合后使用 RTL 中的 SRL_STYLE 属性，或是单元上的 SRL_STAGES_TO_INPUT 或 SLR_STAGES_TO_OUTPUT 属性从 SRL 输入或输出拉出寄存器。在 RTL 中必须修改动态 SRL。
- 当逻辑路径终止于以 LUT 驱动结构寄存器 (FD*) 的时钟使能 (CE) 引脚、同步建立 (S) 引脚或同步复位 (R) 引脚时，布线延迟高于寄存器数据引脚 (D)，尤其是在路径最后一个网络的扇出大于 1 时。
  建议：如果终止于数据引脚 (D) 的路径有更高裕量和更少逻辑层次，则在 RTL 内把信号上的 EXTRACT_ENABLE 或 EXTRACT_RESET 属性设置为 no。此外，也可以把单元上的 CONTROL_SET_REMAP 属性设置为在 opt_design 过程中触发相同优化。

提示：全局使用 synthesis -retiming，或在模块上使用块综合策略（如 BLOCK_SYNTH.RETIMING=1）。

使用专用块和宏原语优化路径

从专用块和宏原语（例如 DSP、块 AMB、URAM、FIFO 或 GT_CHANNEL）/到/在它们之间的逻辑路径布局难度更大，有更高的单元和布线延迟。因此，围绕宏原语添加额外的流水线或减少宏原语路径上的逻辑层次，对提升总体设计性能相当关键。

在修改 RTL 之前，通过启用所有可选的 DSP、RAMB 和 URAM 寄存器并重新运行实现，验证增添流水线给 QoR 带来的好处。在采用这一评估方法时请勿生成比特流。例如：

set_property –dict {DOA_REG 1 DOB_REG 1} [get_cells xx/ramb18_inst]

下面是一个要求增加流水线寄存器或减少逻辑层次的 RAMB18 路径示例（在 route_design 后报告）：

Name	Slack	Requirement	Path Delay	Logic Delay 1	Net Delay	Logic Levels	Routes	Logical Path	BRAM
Path 5	-0.663	3.184	3.472	48%	52%	5	5	RAMB18E2 LUT6 LUT6 LUT6 LUT6 LUT6 FDRE	No DO_REG

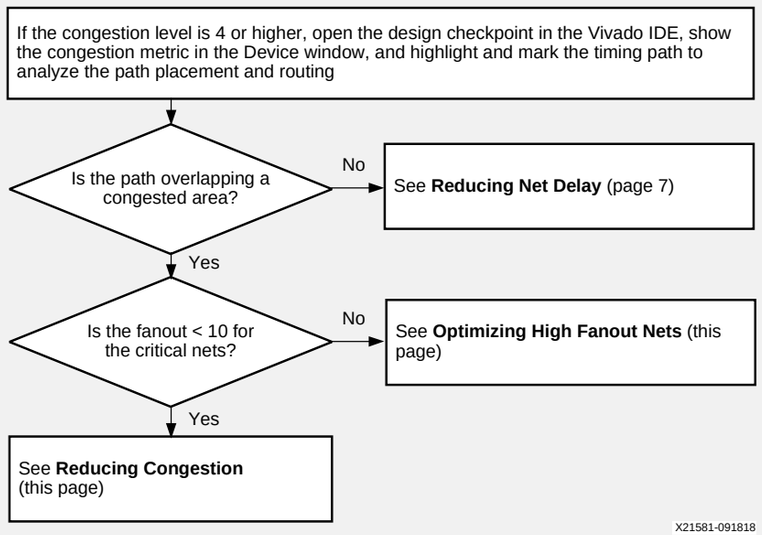
降低信号网络延迟流程 1

全局拥塞给设计性能造成如下影响：

- **4 级 (16x16)：**在 route_design 过程中的小 QoR 可变性
- **5 级 (32x32)：**欠佳布局和值得注意到 QoR 变化
- **6 级 (64x64)：**布局布线难度大，编译时间长。除非性能目标低下，时序 QoR 严重劣化。
- **7 级 (128x128) 及以上：**无法布局或布线。

route_design 命令为 4 级及以上拥塞在日志文件里输出初始估计拥塞表。为了同时报告布局器和布线器拥塞信息，使用 report_design_analysis –congestion。

提示：在 Vivado IDE 内打开布局后或布线后 DCP，以创建交互式 “report_design_analysis”窗口。通过交叉探测，高亮显示拥塞区域并显示拥塞对单个逻辑路径布局和布线的影响。请参阅 **UG949：《识别拥塞》**。

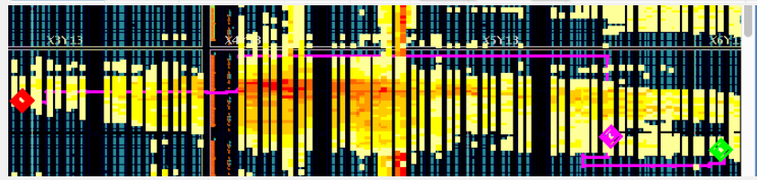


减少拥塞的方法

下面给出关键时序路径的示例，其中网络布线围绕拥塞区域绕行，导致更大的信号网络延迟。

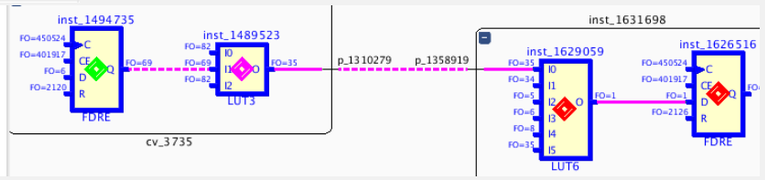
- 所有视图从设计分析报告中都能看到。
- 在“器件”窗口中按逐个 CLB 指标启用垂直和水平布线拥塞。

Name	Slack	Requirement	Path Delay	Logic Delay	Net Delay	High Fanout	Logical Path	IO Crossings
Path 5	-0.230	4	3.839	6%	94%	69	FDRE LUT3 LUT6 FDRE	1



Data Path

Delay Type	Incr (ns)	Path...	Location	Netlist Resource(s)
FDRE (Prop_EFF_SLICEM_C_Q)	(f) 0.076	4.883	Site: SLICE_X179Y744	inst_1784100/inst_174187
net (fo=69, routed)	1.175	6.058		inst_1784100/inst_174187
LUT3 (Prop_D6LUT_SLICEL_I1_O)	(r) 0.050	6.108	Site: SLICE_X163Y749	inst_1784100/inst_174187
net (fo=35, routed)	2.374	8.482		inst_1784100/inst_174187
LUT6 (Prop_C6L..._SLICEM_I0_O)	(r) 0.098	8.580	Site: SLICE_X94Y761	inst_1784100/inst_174187
net (fo=1, routed)	0.066	8.646		inst_1784100/inst_174187



减少拥塞

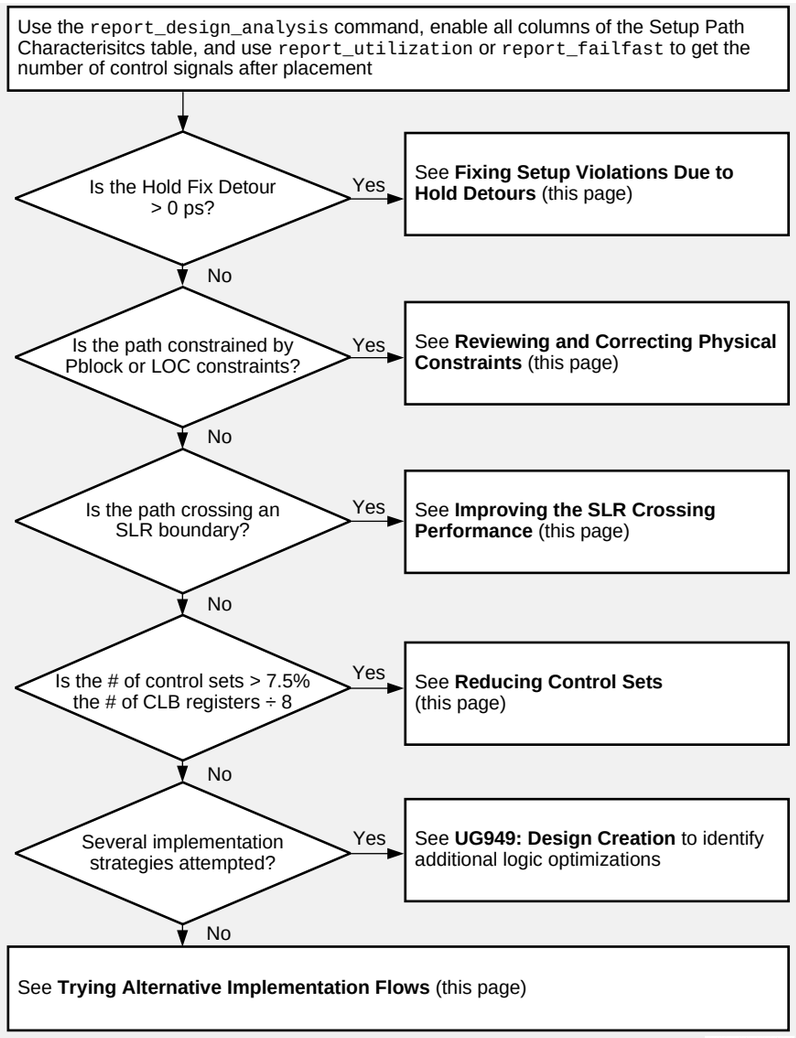
为减少拥塞，赛灵思建议按列出的顺序使用下列方法

- 如果总资源利用率大于 70-80%，可通过移除一些设计功能，或迁移部分模块或内核到另一个 SLR，从而降低器件或 SLR 的资源利用率。同时避免 LUT 和 DSP/RAMB/URAM 利用率同时都高于 80%。如果必须保持高宏原语利用率，尽可能保持 LUT 利用率低于 60%，从而在避免引入复杂的布局规划约束的条件下在拥塞区域内扩展布局。布局后使用 xilinx::designutils::report_failfast -by_slr 逐个 SLR 地审核利用率。
- 尝试多种布局指令（例如 AltSpreadLogic* 或 SSI_Spread*）或 Congestion_* implementation 运行策略。
- 使用 report_design_analysis -complexity -congestion 识别有高连接复杂性（Rent 指数> 0.65 或平均扇出 > 4）的重大拥塞模块（> 15,000 单元）。使用添加到 XDC 文件中的针对拥塞的综合设置：
  set_property BLOCK_SYNTH.STRATEGY {ALTERNATE_ROUTABILITY} [get_cells <congestedHierCellName>]
- 在拥塞区域减少使用 MUXF* 和 LUT 合并。请查看 RDA 拥塞报告中的对应列。在拥塞叶节点单元上设置 MUXF_REMAP 为 1，设置 SOFT_HLUTNM 为 “”。使用 report_qor_suggestions 获取帮助。 把拥塞区域内的非关键高扇出网络提升为全局时钟布线，具体如下：
  set_property CLOCK_BUFFER_TYPE BUFG [get_nets <highFanoutNetName>]
- 重复使用之前低拥塞实现运行中的 DSP、RAMB 和 URAM 布局约束。例如：
  read_checkpoint -incremental routed.dcp -reuse_objects [all_rams] -fix_objects [all_rams]

优化高扇出网络

- 在 RTL 内或借助如下逻辑优化明确地使用基于层级的寄存器复制：
  opt_design –merge_equivalent_drivers –hier_fanout_limit 512
- 在 route_design 前通过额外地调用物理优化步骤，在关键高扇出网络上进行强制复制：
  phys_opt_design -force_replication_on_nets <net>

降低信号网络延迟流程 2



降低信号网络延迟的方法

修复因保持时间绕行造成的建立时间违规

为了确保设计硬件功能正常，修复保持时间违规优先于修复建立时间违规（或 Fmax）。下面的示例显示了存在高偏差且有严格建立要求的两个同步时钟间的路径：

Name	Slack	Requirement	Path Delay	Clock Skew	Hold Fix Detour	Logical Path	Start Point Clock	End Point Clock	SLR Crossings
Path 1	-1.438	1.592	3.244	0.665	1181	FDRE LUT4 FDRE	txoutclk_out[3]_3	app_clk	1

说明：“保持时间修复绕行”以皮秒为单位。要解决保持时间绕行对 Fmax 的影响，请参阅**“解决保持时间违规的方法”**（第 4 页）。

审核并纠正物理约束

所有设计包含物理约束。虽然一般不能更改 I/O 位置，在做出设计更改前必须仔细验证和审核 Pblock 和位置约束。更改可能拉开逻辑间距，并引入较长的信号网络延迟。审核有超过 1 个 Pblock（Pblock 列）的路径和有位置约束（固定位置列）的路径。

提升 SLR 交汇性能

在指向堆叠硅片互联 (SSI) 技术器件时，尽早对如下设计因素进行考虑有助于提升性能：

- 在主要设计层级或内核边界添加流水线寄存器，辅助长距离布线和 SLR 交汇布线
- 验证每个 SLR 的利用率均符合指南（使用 report_failfast –by_slr）
- 使用 USER_SLR_ASSIGNMENT 约束引导实现工具。请参阅 **UG949：《使用软 SLR 布局规划约束》**。
- 如果软约束无效，使用 SLR Pblock 布局约束
- 在布局或布线后，使用 phys_opt_design -slr_crossing_opt

减少控制集

在控制集数量超过指南 (7.5%) 时，不管对整个器件还是对每个 SLR，尝试减少控制集数量：

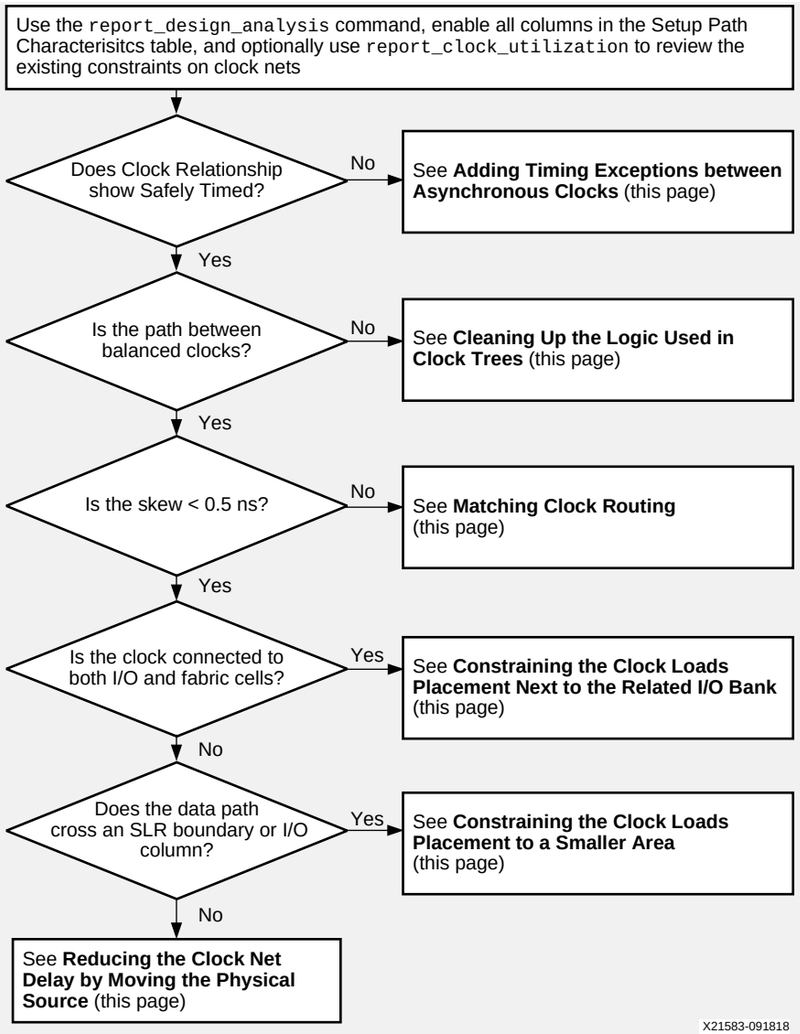
- 在 RTL 内移除时钟使能、设置或复位信号上的 MAX_FANOUT 属性
- 增加最小综合控制信号扇出（如 synth_design –control_set_opt_threshold 16）
- 使用 opt_design –control_set_merge 或 –merge_equivalent_drivers 融合复制的控制信号
- 通过设置 CLB 寄存器单元上的 CONTROL_SET_REMAP 属性，重新映射低扇出控制信号到 LUT

尝试替代性实现流程

可通过默认编译流程快速获得设计基准 (baseline)，并在未满足时序要求的情况下分析设计。如果初始实现后未能满足时序要求，尝试其他建议的实现流程：

- 尝试多个 place_design 指令（最多 10 个）和多次 phys_opt_design 迭代（Aggressive*、Alternate* 指令）
- 使用 set_clock_uncertainty，在 place_design/phys_opt_design 过程中过度约束最关键时钟（最大 0.500 ns）
- 使用 group_path –weight，在必须满足时序要求的时序时钟上提升时序 QoR 的优先级
- 在较小设计修改后使用增量编译流程，保护 QoR 并缩短运行时间

改善时钟偏差流程



改善时钟偏差的方法

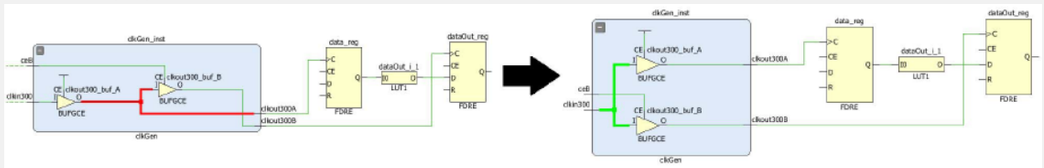
在异步时钟之间添加时序例外

必须将时序路径（其中源和目标时钟源自不同主时钟或没有公共节点）作为异步时钟处理。在这种情况下，偏差可能极其大，从而使得无法收敛时序。根据需要添加 set_clock_groups、set_false_path 和 set_max_delay –datapath_only 约束。详情，请参阅 (UG949)：《在异步时钟之间添加时序例外》。

清理时钟树中使用的逻辑

除非在时钟逻辑中使用 DONT_TOUCH 约束，否则 opt_design 命令会自动清理时钟树。选择时序路径，启用**时钟路径可视化**工具栏按钮，打开原理图 (**F4**) 审核时钟逻辑。

- 通过消除不必要的缓存或并行连接它们可以避开级联时钟缓存之间的时序路径。例如：



- 在时钟等效的情况下，把并行时钟缓存合并为单个时钟缓存。
- 在时钟路径中移除 LUT 或任何组合逻辑，因为它们可使时钟延迟和时钟偏差不可预测。

匹配时钟布线

使用 CLOCK_DELAY_GROUP 提升关键同步时钟间的时钟布线延迟，即使两个时钟网络已拥有相同的 CLOCK_ROOT。下面的示例显示了不使用 CLOCK_DELAY_GROUP 的两个同步时钟：

Global Clock Resources

Global Id	Source Id	Driver Type/Pin	Constraint	Site	Clock Region	Root	Load Clock Region	Clock Loads	Clock Period	Clock
g0	src0	BUFG_GT/O	None	BUFG_GT_X1Y212	X5Y8	CLOCK_REGION_X5Y6	30	110934	3.184	app_clk
g1	src0	BUFG_GT/O	None	BUFG_GT_X1Y215	X5Y8	CLOCK_REGION_X5Y6	2	5202	1.592	txoutclk_out[3]_3

在相关 I/O 组旁边约束时钟负载布局

对于不足 2,000 负载的 I/O 逻辑与结构单元之间的时钟，设置时钟网络上的 CLOCK_LOW_FANOUT 属性，自动布局所有负载到与时钟缓存 (BUFG*) 相同的时钟区域并保持低插入延迟和低偏差。

约束时钟负载布局到更小区域

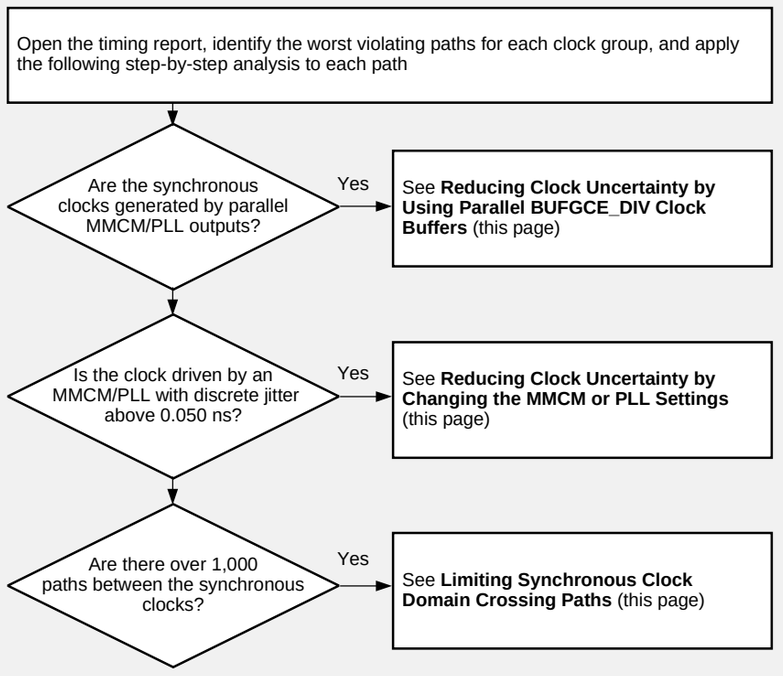
您可以使用 Pblock，强制在更小区域内（例如 1 个 SLR）布局时钟网络负载，以降低插入延迟和偏差，或避免特殊列发生交汇，例如引入偏差损失的 I/O 列。

通过迁移物理源降低时钟信号网络延迟

使用位置约束迁移源混合模式时钟管理器 (MMCM) 或锁相环 (PLL) 到时钟负载中心，以降低最大时钟插入延迟，从而降低时钟消极因素以及偏差。详情，请参阅 **(UG949)：《改善 UltraScale 和 UltraScale+ 器件的偏差》**。

改善时钟不确定性流程

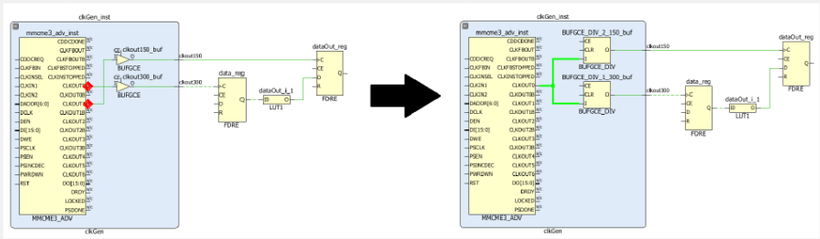
时钟不确定性指输入抖动、系统抖动、离散抖动、相位错误或用户加入的不确定性的量。用户加入部分的目的是添加到理想时钟沿上，以准确地为硬件工作条件建模。时钟不确定性同时影响建立时序路径和保持时序路径，并根据时钟树中使用的资源变化。



改善时钟不确定性的方法

使用并行 BUFGCE_DIV 时钟缓存减少时钟不确定性

对于由相同 MMCM 或 PLL 生成并受多个时钟输出驱动的周期比为 2、4 或 8 的同步时钟，仅使用 1 个 MMCM 或 PLL 输出并将其连接到并行 BUFGCE_DIV 时钟缓存（限 UltraScale™ 和 UltraScale+™ 器件）。这种时钟拓扑能消除 MMCM 或 PLL 相位偏差，大多数情况能实现 0.120 ns 的时钟不确定性。



下面给出一个示例，显示了 150 MHz 时钟和 300 MHz 时钟间时钟域交汇 (CDC) 路径的时钟不确定性降低。

- **降低前的时钟不确定性**：0.188 ns（建立）、0.188 ns（保持）
- **降低后的时钟不确定性**：0.068 ns（建立）、0.000 ns（保持）

使用时钟向导生成使用并行 BUFGCE_DIV 缓存的时钟拓扑，同时设置时钟上的 CLOCK_DELAY_GROUP 属性。

通过更改 MMCM 或 PLL 设置减小时钟不确定性

诸如 MMCM 和 PLL 等时钟修改块能以离散抖动和相位错误的形式来影响时钟不确定性。

- 在时钟向导内或使用 set_property 命令，通过修改 M（乘法器）或 D（除法器）的值提高压控振荡器 (VCO) 频率。例如，MMCM (VCO=1 GHz) 会引入 167 ps 抖动和 384 ps 相位错误，而 MMCM (VCO=1.43 GHz) 对应值是 128 ps 和 123 ps。
- 如有可能，使用 PLL 代替 MMCM，因为 PLL 引入的时钟不确定性较小。

限制同步时钟域交汇路径

由单独的时钟缓存驱动的同步时钟之间的时序路径表现出较高的偏差，因为公共时钟树节点位于时钟缓存之前，从而造成时序分析中的较高悲观因素。因此，在这些路径上更难同时满足建立和保持时间要求，尤其是对高频时钟而言（高于 500 MHz）。要想确定两个时钟间的路径数量，使用 report_timing_summary（时钟间路径部分）或 report_clock_interaction。 下面给出的示例显示了两个高速时钟间存在大量路径的设计（要求 = 1.592 ns）。其中 30% 的路径未能满足时序要求，说明它们非常难于实现。

Source Clock	Destination Clock	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	Inter-Clock Constraints
app_clk	txoutclk_out[3]_3	-0.348	-162.119	1668	5623	1.592	Timed
rxoutclk_out[0]_4	rxoutclk_out[0]_4	0.262	0.000	0	2998	2.388	Partial False Path
pcie_refclk	pcie_refclk	5.054	0.000	0	1508	7.960	Timed
txoutclk_out[3]_3	app_clk	-0.153	-0.196	2	1198	1.592	Partial False Path

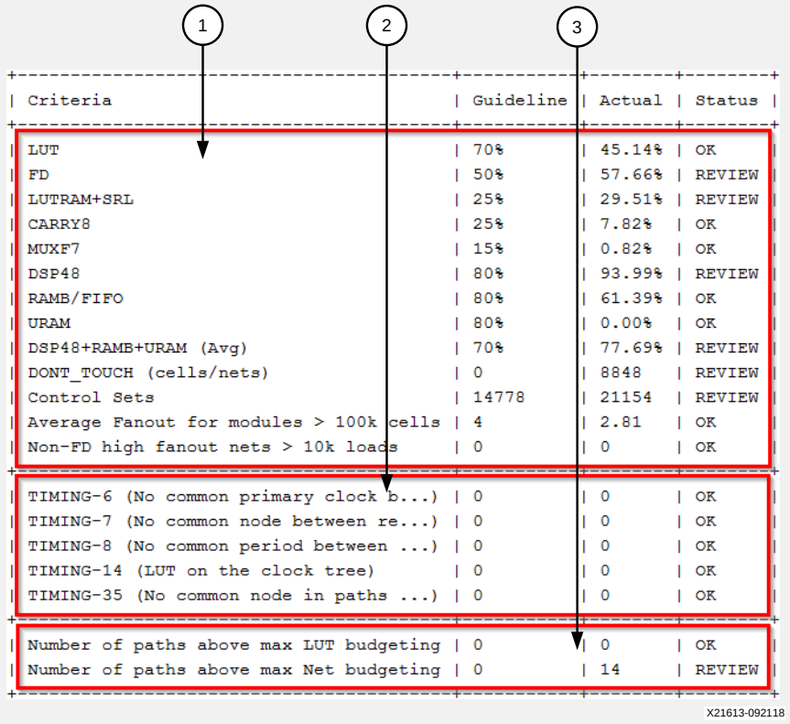
检查时钟域交汇涉及的逻辑，移除不必要的逻辑路径或尝试下列修改：

- 在受时钟使能控制的路径上添加多周期路径约束，因为新数据并非在每个周期都传输。
- 以额外的时延为代价，使用异步交汇电路和合适的时序例外替代交汇逻辑。例如使用异步 FIFO 或 XPM_CDC 参数化宏。详情，请参阅《UltraScale 架构库指南》(UG974)。

FAILFAST 报告概览

在 Failfast 报告中，解决标记为 REVIEW 的检查以改善实现与时序收敛。下面是 Failfast 报告的部分内容：

1. **设计特性**：默认利用率指南基于 SSI 技术器件，对非 SSI 技术器件可以放松。有一个或多个 REVIEW 检查的设计可行但难以实现。
2. **时钟检查**：这些检查非常重要，必须解决。
3. **LUT 和网络预算**：使用保守方法更可靠地预测在器件利用率高的情况下布局后哪些逻辑路径最可能无法满足时序要求。



X21613-092118

FAILFAST 报告使用示例

基于 Pblock 和基于 SLR 的分析

report_failfast 脚本报告特定物理区域或 SLR 的利用率情况如下：

- **布局前**：使用 -pblock <pblockName> 报告布局规划约束。当存在 SLR Pblock 时，这对于在设计周期中尽早审核 SLR 布局约束特别重要。
- **布局后**：使用 -slr <slrName> 或 -by_slr 报告每个 SLR 的利用率指标。

布局规划假设分析

使用 -top 或 -cell <hierCellName> 与 -pblock <pblockName>，报告利用率指标，在不改变 Pblock 的单元的情况下找出优质布局规划约束。

标记为“REVIEW Analysis”的 Failfast 报告检查

使用 -detailed_report <prefix> 选项，report_failfast 会为每个未能满足指南的检查生成更加详细的报告（除资源利用率检查）。审核下列每一个报告：

- <prefix>.TIMING.rpt：Detailed Methodology TIMING-* 违规
- <prefix>.AVGFO.rpt：大于 100,000 的模块的平均扇出
- <prefix>.HFN.rpt：驱动 10,000 以上负载的非 FD 高扇出网络 (HFN)
- <prefix>.DONT_TOUCH.rpt：带 DONT_TOUCH 属性集的单元/网络清单
- <prefix>.timing_budget_LUT.rpt：不满足 LUT 预算要求的详细时序路径
- <prefix>.timing_budget_LUT.rpx：不满足 LUT 预算的详细时序路径（Vivado IDE 交互报告）
- <prefix>.timing_budget_Net.rpt：不满足网络预算的详细时序路径
- <prefix>.timing_budget_Net.rpx：不满足网络预算的详细时序路径（Vivado IDE 交互报告）

内核层面或模块层面分析

以无关联模式综合每个内核或主要设计层级。然后，验证时序是否满足估计的延迟和实际时钟约束：

- 审核时序报告并解决任何不满足的路径
- 审核 report_failfast 的逻辑层面预算部分，确认在布局后最可能不满足时序要求的路径。通过修改您的设计，优化这一分析标记的路径。

实现前设计分析

在所有内核、子模块和顶层设计完成组装和综合后，审核并解决所有标记为 REVIEW 的检查。

实现前布局规划约束分析

对于大型设计和 SDx 设计，验证设计架构和层级是否正确地符合器件布局规划。

布局后 SLR 利用率分析

使用 report_failfast -by_slr 验证每个 SLR 的资源利用率是否符合建议的指南。

提示：在项目模式下使用下列 Tcl 钩子添加 Failfast 报告：

set_property STEPS.OPT_DESIGN.TCL.POST <path>/postopt_failfast.tcl [get_runs impl_*]

下面给出一个实例：postopt_failfast.tcl：

xilinx::designutils::report_failfast -file failfast_postopt.rpt -detailed_reports postopt