INTRODUCTION

This quick reference guide presents the following simplified, step-by-step flows for quickly closing timing, based on the recommendations in the *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949):

- **Initial Design Checks:** Review utilization, logic levels, and timing constraints before implementing the design.
- **Timing Baselining:** Review and address timing violations after each implementation step to help close timing after routing.
- **Timing Violation Resolution:** Identify the root cause of setup or hold violations, and resolve the timing violations.

Failfast Report

The Tcl-based failfast report summarizes key information about the design and constraints, which allows you to quickly identify and address common implementation and performance issues. By default, the report analyzes the entire design and outputs a table with each analyzed metric compared to a typical guideline. Metrics that do not comply with guidelines are marked as REVIEW. The report includes the following sections:

- Design characteristics
- Critical clock methodology checks
- Conservative logic-level assessments based on a target Fmax

Starting in the Vivado® tools 2018.1 release, the `report_failfast` script is installed by default and can be called as follows:

`xilinx::designutils::report_failfast`

Starting in the SDx™ tools 2018.2 release, `report_failfast` is called during the compilation flow when using `xocc -R 1` or `xocc -R 2`.

For more information on `report_failfast`, see **Failfast Report Overview** (page 10).

INITIAL DESIGN CHECKS FLOW
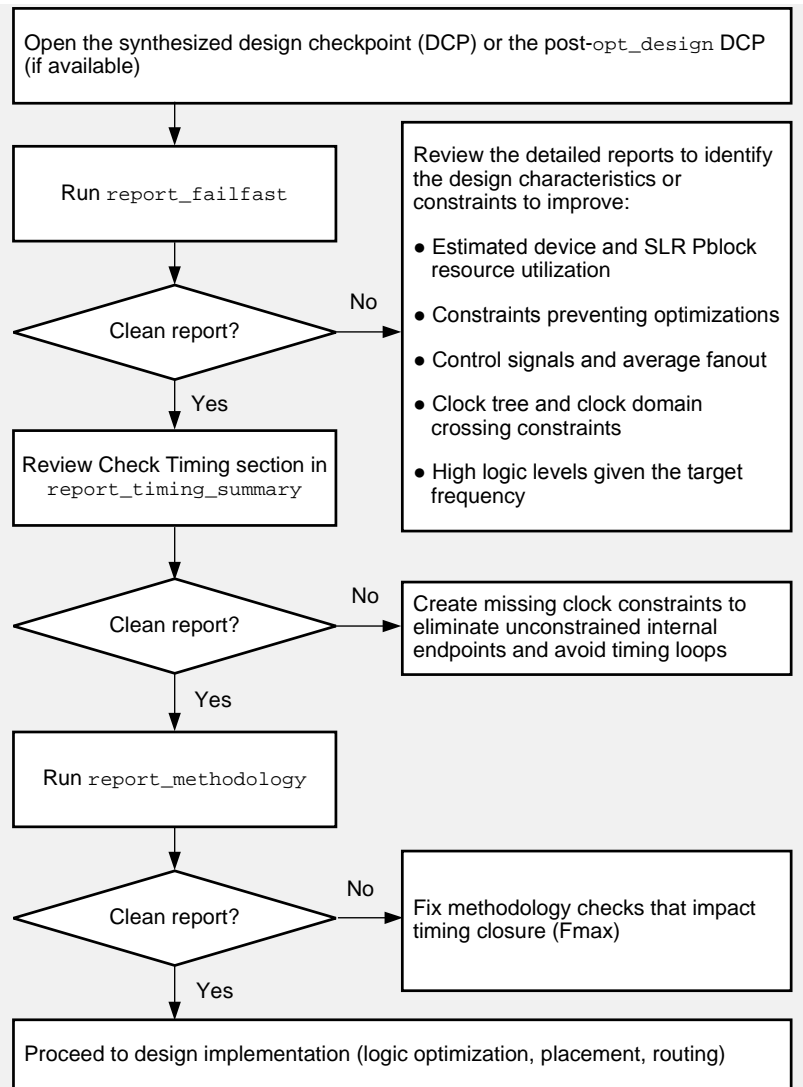
X21574-091818

INITIAL DESIGN CHECKS DETAILS

Although implementing a design on a Xilinx® device is a fairly automated task, achieving higher performance and resolving compilation issues due to timing or routing violations can be a complex and time-consuming activity. It can be difficult to identify the reason for a failure based on simple log messages or post-implementation timing reports generated by the tools. Therefore, it is essential to adopt a step-by-step design development and compilation methodology, including the review of intermediate results to ensure the design can proceed to the next implementation step.

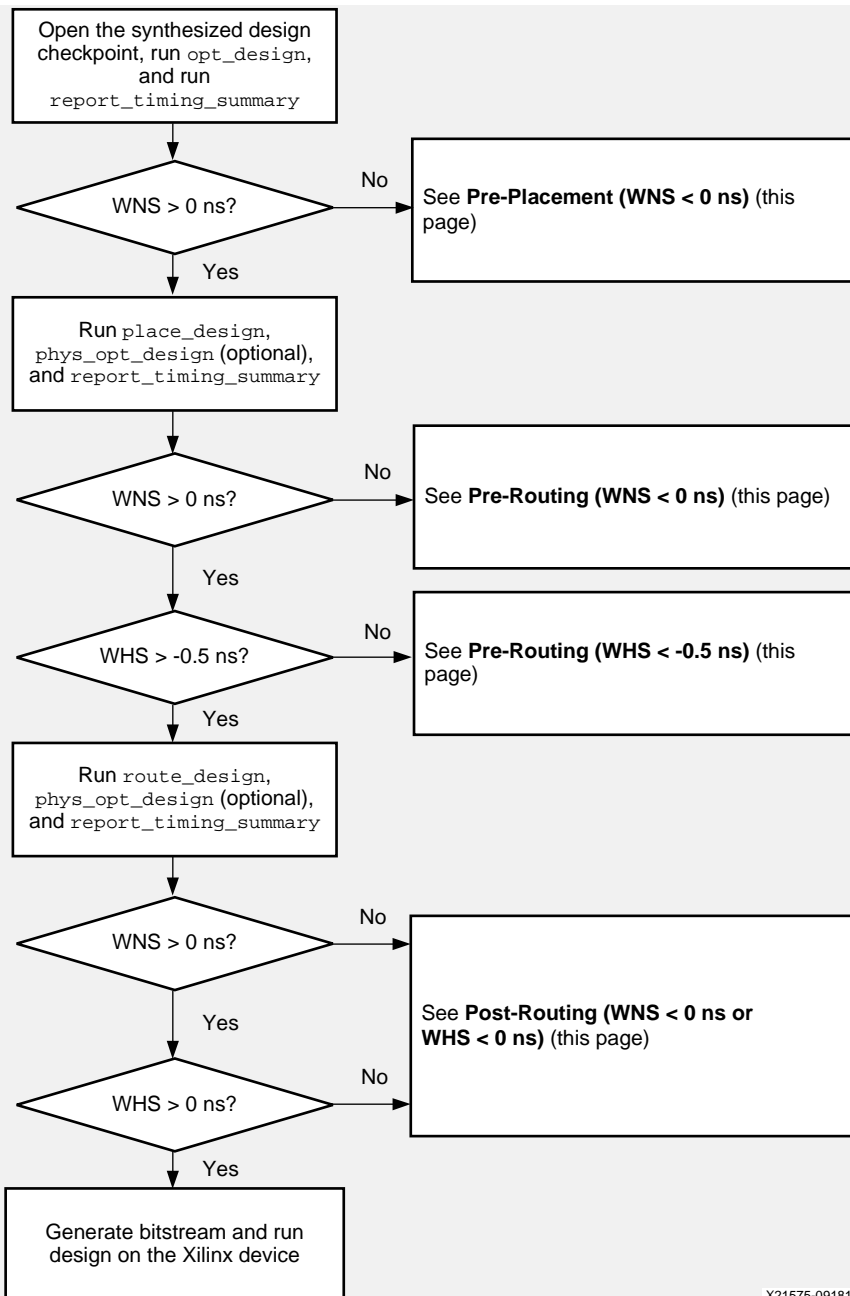The first step is to make sure all initial design checks are addressed. Review these checks at the following levels:

- Each kernel made of custom RTL or generated by Vivado HLS
  Note: Check that target clock frequency constraints are realistic.
- Each major hierarchy corresponding to a subsystem, such as a Vivado IP integrator block diagram with several kernels, IP blocks, and connectivity logic
- Complete design with all major functions and hierarchies, I/O interfaces, complete clocking circuitry, and physical and timing constraints

If the design uses floorplanning constraints, such as super logic region (SLR) assignments or logic assigned to Pblocks, review the estimated resource utilization for each physical constraint, and make sure that the utilization guidelines are met. See the default guidelines in the failfast report. To generate reports, use the following commands:

- `report_utilization -pblocks <pblockName>`
- `report_failfast -pblock <pblockName>`
- `report_failfast [-slr SLRn | -by_slr]`

 XILINX.

TIMING BASELINING FLOW

```
┌─────────────────────────────────┐
│ Open the synthesized design     │
│ checkpoint, run opt_design,     │
│ and run                         │
│ report_timing_summary           │
└─────────────────────────────────┘
              │
         WNS > 0 ns?  ──No──→  See Pre-Placement (WNS < 0 ns) (this page)
              │
             Yes
              │
┌─────────────────────────────────┐
│ Run place_design,               │
│ phys_opt_design (optional),     │
│ and report_timing_summary       │
└─────────────────────────────────┘
              │
         WNS > 0 ns?  ──No──→  See Pre-Routing (WNS < 0 ns) (this page)
              │
             Yes
              │
        WHS > -0.5 ns?  ──No──→  See Pre-Routing (WHS < -0.5 ns) (this page)
              │
             Yes
              │
┌─────────────────────────────────┐
│ Run route_design,               │
│ phys_opt_design (optional),     │
│ and report_timing_summary       │
└─────────────────────────────────┘
              │
         WNS > 0 ns?  ──No──→  See Post-Routing (WNS < 0 ns or WHS < 0 ns) (this page)
              │
             Yes
              │
         WHS > 0 ns?  ──No──→
              │
             Yes
              │
┌─────────────────────────────────┐
│ Generate bitstream and run      │
│ design on the Xilinx device     │
└─────────────────────────────────┘
```

X21575-091818

TIMING BASELINING EXAMPLE

The objective of timing baselining is to ensure that the design meets timing by analyzing and resolving timing challenges after each implementation step. Fixing the design and constraints issues earlier in the compilation flow ensures a broader impact and higher performance. Review and address timing violations before moving onto the next step by creating intermediate reports as follows:

Reports in Vivado Project Mode	Reports in Vivado Non-Project Mode	Reports in the SDx Tools
Use the Ultrafast™ design methodology or timing closure report strategies	Add the following report commands after each implementation step: ▪ report_timing_summary ▪ report_methodology ▪ report_failfast	Use the xocc -R 1 or xocc -R 2 option to generate failfast reports, intermediate timing reports, and DCPs in the following directory: <runDir>/_x/link/vivado/prj/prj.runs/impl_1

Pre-Placement (WNS < 0 ns)

Before place_design, the timing report reflects the design performance assuming the best possible logic placement for each logic path. Setup violations must be addressed by adopting the Initial Checks recommendations.

Pre-Routing (WNS < 0 ns)

Before route_design, the timing report reflects the design performance assuming the best possible routing delays for each individual net with some fanout penalty and without considering hold fixing impact (net routing detours) or congestion. Setup violations are often due to sub-optimal placement caused by (1) high device or SLR utilization, (2) placement congestion due to complex logic connectivity, (3) many paths with many logic levels, and (4) high clock skew between unbalanced clocks or high clock uncertainty. Run phys_opt_design in Explore or AggressiveExplore mode to try improving the post-place_design QoR. If unsuccessful, focus on improving the placement QoR first.

Pre-Routing (WHS < -0.5 ns)

When the performance goal is not met after routing and worst negative slack (WNS) is positive before routing, try to reduce large estimated worst hold slack (WHS) violations. Fewer and smaller pre-route hold violations help route_design focus on Fmax rather than fixing hold time violations.

Post-Routing (WNS < 0 ns or WHS < 0 ns)

After route_design, first verify that the design is fully routed by reviewing the log files or running report_route_status on the post-route design checkpoint (DCP). Routing violations and large setup (WNS) or hold (WHS) violations are the result of high congestion. Use the **Analyzing Setup Violations** (page 3), **Resolving Hold Violations** (page 4), and **Congestion Reduction Techniques** (page 6) to identify and implement the resolution steps. Try running phys_opt_design after route_design to address small setup violations > -0.200 ns.

When iterating the design, constraints, and compilation strategies, keep track of the QoR after each step, including the congestion information. Use the QoR table to compare run characteristics and determine what to focus on first when addressing the remaining timing violations.

Impl. Run	opt_design		place_design			phys_opt_design				route_design				
	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

TIP: Use report_qor_suggestions after place_design and after route_design to automatically identify design, constraints, and tool option changes that can help improve the QoR for new compilations.
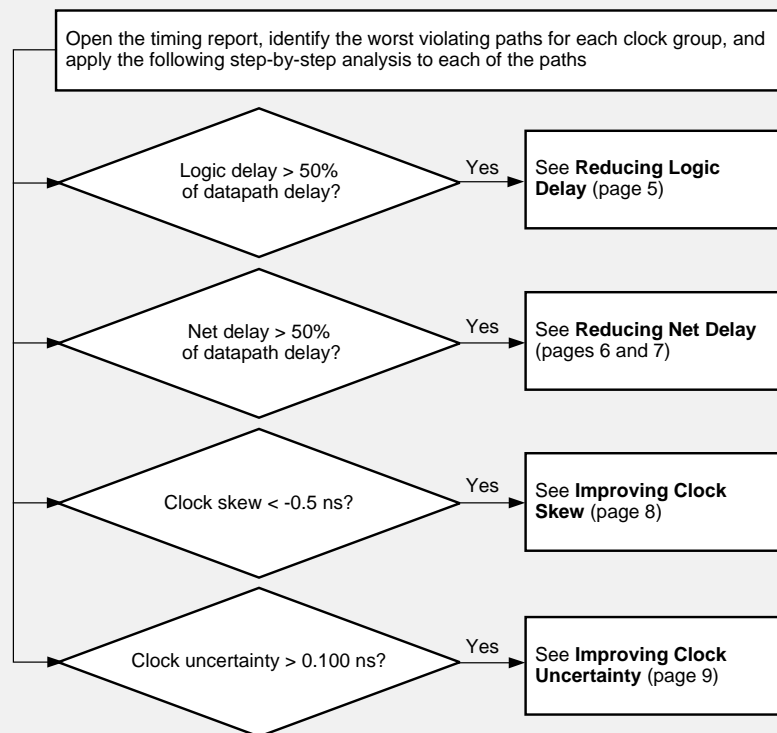
XILINX

ANALYZING SETUP VIOLATIONS FLOW

Design performance is determined by the following:

- **Clock skew and clock uncertainty**: How efficiently the clocks are implemented
- **Logic delay**: Amount of logic traversed during a clock cycle
- **Net or route delay**: How efficiently Vivado implementation places and routes the design

Use the information in the timing path or design analysis reports to:

- Identify which of these factors contributes most to timing violations
- Determine how to iteratively improve the QoR

TIP: If needed, open the DCP after each step to generate additional reports.

Open the timing report, identify the worst violating paths for each clock group, and apply the following step-by-step analysis to each of the paths

Logic delay > 50% of datapath delay? — Yes → See **Reducing Logic Delay** (page 5)

Net delay > 50% of datapath delay? — Yes → See **Reducing Net Delay** (pages 6 and 7)

Clock skew < -0.5 ns? — Yes → See **Improving Clock Skew** (page 8)

Clock uncertainty > 0.100 ns? — Yes → See **Improving Clock Uncertainty** (page 9)

X21576-091818

FINDING SETUP TIMING PATH CHARACTERISTICS IN THE REPORTS

In Vivado project mode, find setup timing path characteristics as follows:

1. In the Design Runs window, select the implementation run to analyze.
2. In the Implementation Run Properties window, select the **Reports** tab.
3. Open the timing summary report or design analysis report for the selected implementation step:
   - **Timing summary report**: `<runName>_<flowStep>_report_timing_summary` (.rpt for text or .rpx for the Vivado IDE)
   - **Design analysis report**: `<runName>_<flowStep>_report_design_analysis`

In Vivado non-project mode or in the SDx tools, do either of the following:

- Open the reports in the implementation run directory.
- Open the implementation DCP in the Vivado IDE, and open the RPX version of the report.
  Note: Using the Vivado IDE allows you to cross-probe between the reports, schematics, and Device window.

For each timing path, the logic delay, route delay, clock skew, and clock uncertainty characteristics are located in the header of the path:
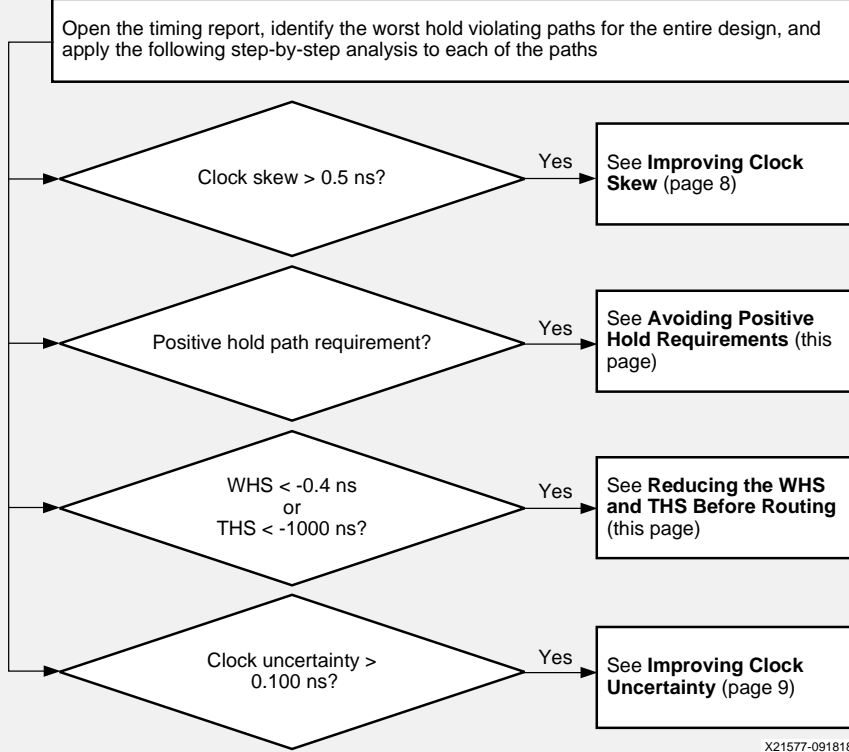
Summary	
Name	↳ Path 4
Slack	−0.675ns
Source	▷ inst_209033/inst_209021/inst_200956/inst_200920/inst_
Destination	▷ inst_209033/inst_209021/inst_200956/inst_200920/inst_
Path Group	app_clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	3.184ns (app_clk rise@3.184ns – app_clk rise@0.000ns)
Data 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
```

Except for the clock uncertainty, the same timing path characteristics are located in the Setup Path Characteristics of the design analysis report:

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 FDRE
↳ Path 3	−0.683	3.184	3.483	42%	58%	−0.362	5	5	RAMB18E2 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

TIP: In text mode, all columns of the Setup Path Characteristics column appear, making the table very wide. In the Vivado IDE, the same table shows a reduced number of columns to help with visualization. Right-click the table header to enable or disable columns as needed. For example, the DONT_TOUCH or MARK_DEBUG columns are not visible by default. Enable these columns to view important information skipped logic optimization analysis, which is difficult to identify otherwise.

≡ XILINX

RESOLVING HOLD VIOLATIONS FLOW

Open the timing report, identify the worst hold violating paths for the entire design, and apply the following step-by-step analysis to each of the paths

Clock skew > 0.5 ns? — Yes → See **Improving Clock Skew** (page 8)

Positive hold path requirement? — Yes → See **Avoiding Positive Hold Requirements** (this page)

WHS < -0.4 ns or THS < -1000 ns? — Yes → See **Reducing the WHS and THS Before Routing** (this page)

Clock uncertainty > 0.100 ns? — Yes → See **Improving Clock Uncertainty** (page 9)

X21577-091818

Following is an example of a hold timing path with high clock skew:

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

RESOLVING HOLD VIOLATIONS TECHNIQUES

Avoiding Positive Hold Requirements

When using multicycle path constraints to relax setup checks, you must:

- Adjust hold checks on the same path so the same launch and capture edges are used in the hold time analysis. Failure to do so leads to a positive hold requirement (one or multiple clock periods) and impossible timing closure.
- Specify the endpoint pin instead of just the cell or clock. For example, the endpoint cell REGB has three input pins: C, EN, and D. Only the REGB/D pin should be constrained by the multicycle path exception, *not* the clock enable (EN) pin because the EN pin can change at every clock cycle. If the constraint is attached to a cell instead of a pin, all of the valid endpoint pins are considered for the constraints, including the EN pin.

X21578-091818

X21579-091818
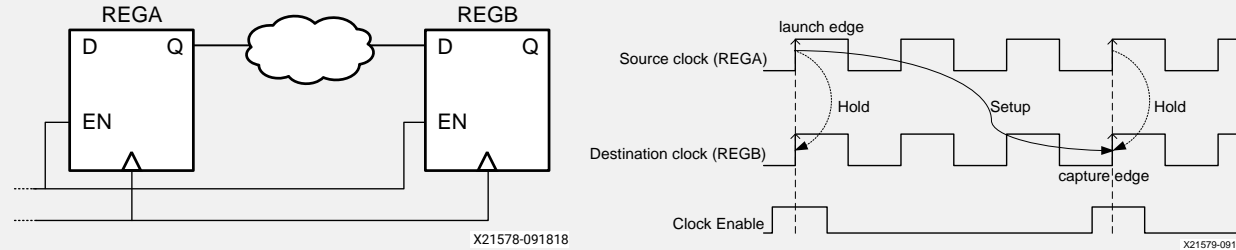
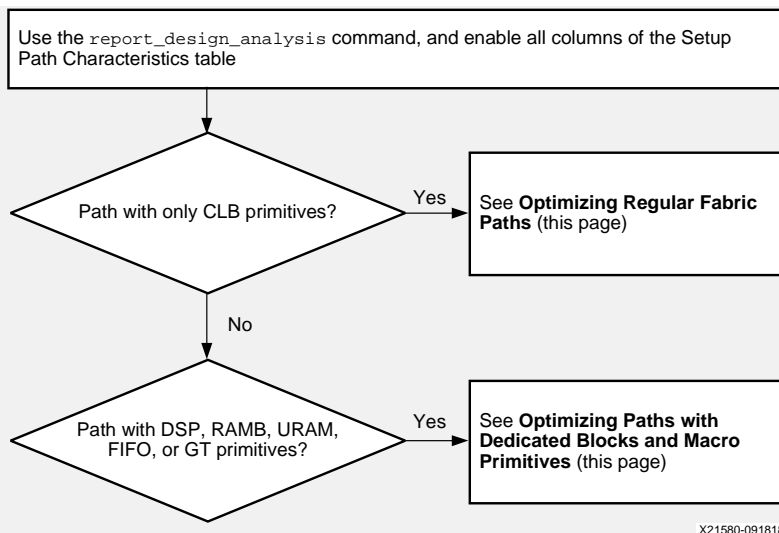Xilinx recommends that you always use the following syntax:

```
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
```

Reducing the WHS and THS Before Routing

Large estimated hold violations increase the routing challenge and cannot always be resolved by `route_design`. The post-placement `phys_opt_design` command provides several hold fixing options:

- The insertion of opposite-edge triggered registers between sequential elements splits a timing path into two half period paths and significantly reduces hold violations. This optimization is only performed if setup timing does not degrade. Use the following command:
  `phys_opt_design -insert_negative_edge_ffs`
- The insertion of LUT1 buffers delays the datapath to reduce hold violations without introducing setup violations. Use the following commands:
  - `phys_opt_design -hold_fix`: Performs LUT1 insertion on paths with the largest WHS violations only.
  - `phys_opt_design -aggressive_hold_fix`: Performs LUT1 insertion on more paths to significantly reduce the total hold slack (THS) at the expense of a noticeable LUT utilization increase and longer compile time. This option can be combined with any `phys_opt_design` directive.
  - `phys_opt_design -directive ExploreWithAggressiveHoldFix`: Performs LUT1 insertion to fix hold in addition to all other physical optimizations designed to improve Fmax.

XILINX

REDUCING LOGIC DELAY FLOW

Use the `report_design_analysis` command, and enable all columns of the Setup Path Characteristics table

↓

Path with only CLB primitives? — **Yes** → See **Optimizing Regular Fabric Paths** (this page)

No

↓

Path with DSP, RAMB, URAM, FIFO, or GT primitives? — **Yes** → See **Optimizing Paths with Dedicated Blocks and Macro Primitives** (this page)

X21580-091818

Vivado implementation focuses on the most critical paths first. This means less difficult paths often become critical after placement or after routing. Xilinx recommends identifying and improving the longest paths after synthesis or after `opt_design`, because this has the biggest impact on QoR and usually dramatically reduces the number of place and route iterations to reach timing closure. Use the `report_design_analysis` Logic Level Distribution table to identify the clock domains that require design improvements by weighing the logic level distribution against the requirement. The lower the requirement, the fewer logic levels are allowed. For example, in the following pre-placement logic level distribution report:

- Review all paths with 8 logic levels or more for txoutclk_out[0]_4.
- Review all paths with 11 logic levels or more for app_clk.

🔍 ℹ️ **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

Note: Cascaded CARRY or MUXF cells can artificially increase the logic level number and have a low impact on delay.

TIP: In the Vivado IDE report, click the logic level number to select the paths, and press **F4** to generate the schematics and review the logic.

REDUCING LOGIC DELAY TECHNIQUES

TIP: Use `report_qor_suggestions` to automatically identify common logic delay reduction techniques and to generate design tuning constraints that you can use in your next implementation run.

Optimizing Regular Fabric Paths

Regular fabric paths are paths between registers (FD*) or shift registers (SRL*) that traverse a mix of LUTs, MUXFs, and CARRYs. If you encounter issues with regular fabric paths, Xilinx recommends the following. For more information, see the *Vivado Design Suite User Guide: Synthesis* (UG901) and *Vivado Design Suite User Guide: Implementation* (UG904).

- Small cascaded LUTs (LUT1-LUT4) can be merged into fewer LUTs unless prevented by the design hierarchy, by intermediate nets with some fanout (10 and higher), or by the use of KEEP, KEEP_HIERARCHY, DONT_TOUCH, or MARK_DEBUG properties.
  Recommended: Remove the properties, and rerun starting from the synthesis step or from `opt_design -remap`.
- Single CARRY (non-cascaded) cells limit LUT optimizations and can make placement less optimal.
  Recommended: Use the FewerCarryChains synthesis directive, or set the CARRY_REMAP property on the cells to be removed by `opt_design`.
- The shift register SRL* delay is higher than the register FD* delay, and SRL placement might be less optimal than FD placement.
  Recommended: Pull a register from the input or output of the SRL using the SRL_STYLE attribute in RTL or the SRL_STAGES_TO_INPUT or SLR_STAGES_TO_OUTPUT property on the cell after synthesis. Dynamic SRLs must be modified in the RTL.
- When the logic path ends with a LUT driving a clock enable (CE), synchronous set (S), or synchronous reset (R) pin of a fabric register (FD*), the routing delay is higher than register data pin (D), especially when the fanout of the last net of the path is greater than 1.
  Recommended: If the path ending at the data pin (D) has a higher slack and fewer logic levels, set the EXTRACT_ENABLE or EXTRACT_RESET attribute to no on the signal in RTL. Alternatively, set the CONTROL_SET_REMAP property on the cell to trigger the same optimization during `opt_design`.

TIP: Use synthesis `-retiming` globally, or use the block synthesis strategy on a module (e.g., BLOCK_SYNTH.RETIMING=1).

Optimizing Paths with Dedicated Blocks and Macro Primitives

Logic paths from/to/between dedicated blocks and macro primitives (e.g., DSP, RAMB, URAM, FIFO, or GT_CHANNEL) are more difficult to place and have higher cell and routing delays. Therefore, adding extra pipelining around the macro primitives or reducing the logic levels on the macro primitive paths is critical for improving the overall design performance.

Before modifying the RTL, validate the QoR benefit of adding pipelining by enabling all optional DSP, RAMB, and URAM registers and rerunning implementation. Do *not* generate a bitstream when adopting this evaluation technique. For example:

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

Following is an example of a RAMB18 path that requires additional pipeline registers or logic level reduction (reported after `route_design`):

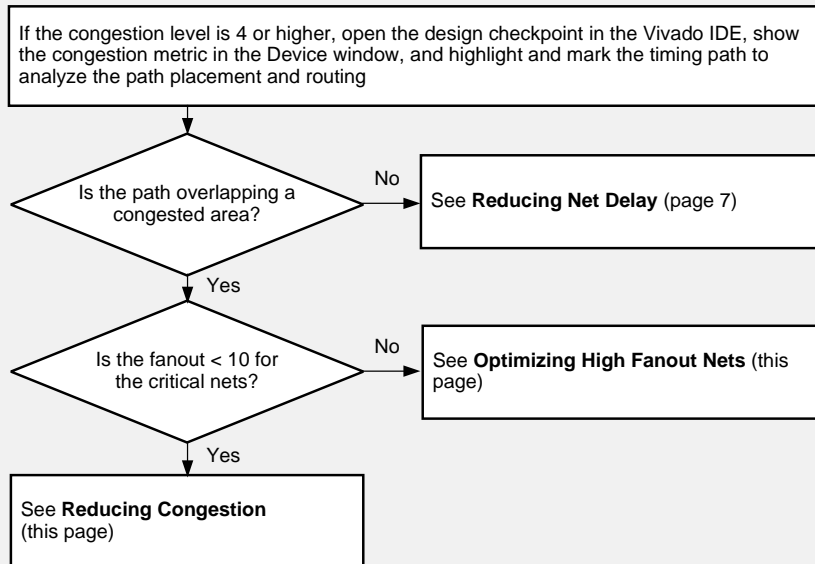| Name	Slack	Requirement	Path Delay	Logic Delay	1	Net Delay	Logic Levels	Routes	Logical Path	BRAM
↳ Path 5	-0.663	3.184	3.472	48%		52%	5	5	RAMB18E2 LUT6 LUT6 LUT6 LUT6 LUT6 FDRE	No DO_REG

📧 **XILINX**

REDUCING NET DELAY FLOW 1

Global congestion impacts the design performance as follows:

- **Level 4 (16x16)**: Small QoR variability during `route_design`
- **Level 5 (32x32)**: Sub-optimal placement and noticeable QoR variations
- **Level 6 (64x64)**: Difficult placement and routing and long compilation time. Timing QoR is severely degraded unless the performance goal is low.
- **Level 7 (128x128) and above**: Impossible to place or route.

The `route_design` command outputs the Initial Estimated Congestion table in the log file for congestion Level 4 or above.

To report both placer and router congestion information, use `report_design_analysis –congestion`.

TIP: Open the post-place or post-route DCP to create an interactive `report_design_analysis` window in the Vivado IDE. Highlight the congested areas and visualize the impact of congestion on individual logic path placement and routing by cross-probing. See **UG949: Identifying Congestion**.

If the congestion level is 4 or higher, open the design checkpoint in the Vivado IDE, show the congestion metric in the Device window, and highlight and mark the timing path to analyze the path placement and routing

```
Is the path overlapping a congested area?
   --No--> See Reducing Net Delay (page 7)
   |
   Yes
   |
Is the fanout < 10 for the critical nets?
   --No--> See Optimizing High Fanout Nets (this page)
   |
   Yes
   |
See Reducing Congestion (this page)
```
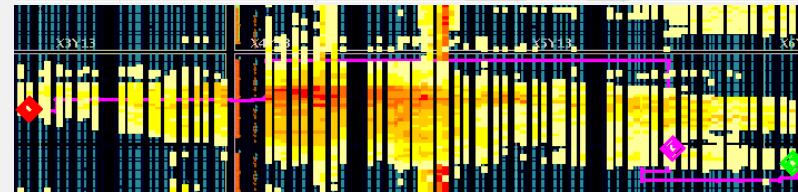
X21581-091818
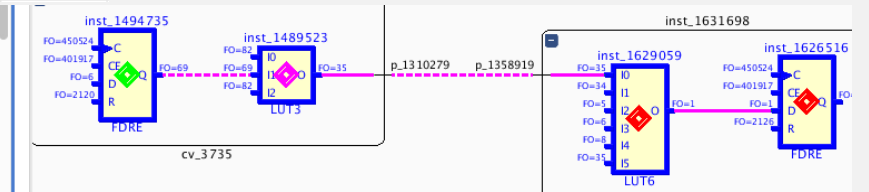
REDUCING CONGESTION TECHNIQUES

Following is an example of a critical timing path where net routing is detoured around the congested area, leading to higher net delays:

- All views are accessible from the design analysis report
- Enable Vertical and Horizontal routing congestion per CLB metrics in the Device window

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

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

Reducing Congestion

To reduce congestion, Xilinx recommends using the following techniques in the order listed:
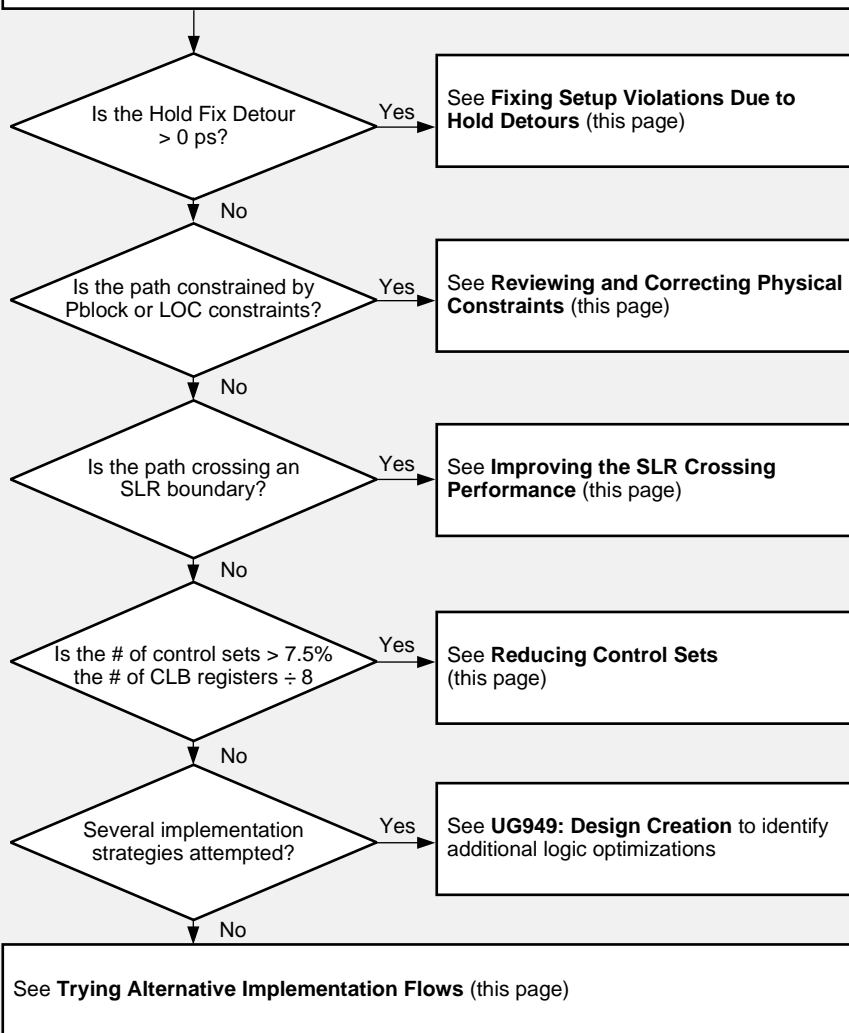
- When the overall resource utilization is above 70-80%, lower the device or SLR utilization by either removing some design functions or moving some modules or kernels to a different SLR. Avoid LUT and DSP/RAMB/URAM utilization that is above 80% at the same time. If the macro primitive utilization percentage must be high, try keeping LUT utilization below 60% to allow placement spreading in the congested area, without introducing complex floorplanning constraints. Use `xilinx::designutils::report_failfast -by_slr` to review the utilization per SLR after placement.
- Try several placer directives (e.g., AltSpreadLogic* or SSI_Spread*) or the Congestion_* implementation run strategies.
- Use `report_design_analysis -complexity -congestion` to identify large, congested modules (> 15,000 cells) with high connectivity complexity (Rent Exponent > 0.65 or Average Fanout > 4). Use the congestion-oriented synthesis settings, which are added to the XDC file: `set_property BLOCK_SYNTH.STRATEGY {ALTERNATE_ROUTABILITY} [get_cells <congestedHierCellName>]`
- Reduce MUXF* and LUT combining usage in the congested region. See the corresponding columns in the RDA congestion report. Set MUXF_REMAP to 1 and SOFT_HLUTNM to "" on the congested leaf cells. Use `report_qor_suggestions` for help.
- Promote non-critical high fanout nets in the congested region to global clock routing as follows: `set_property CLOCK_BUFFER_TYPE BUFG [get_nets <highFanoutNetName>]`
- Reuse DSP, RAMB, and URAM placement constraints from previous implementation runs with low congestion. For example: `read_checkpoint -incremental routed.dcp -reuse_objects [all_rams] -fix_objects [all_rams]`

Optimizing High Fanout Nets

- Use hierarchy-based register replication explicitly in RTL or with the following logic optimization: `opt_design –merge_equivalent_drivers –hier_fanout_limit 512`
- Force replication on critical high fanout nets with additional calls of physical optimization steps before `route_design`: `phys_opt_design -force_replication_on_nets <net>`

XILINX

REDUCING NET DELAY FLOW 2

Use the `report_design_analysis` command, enable all columns of the Setup Path Characterisitcs table, and use `report_utilization` or `report_failfast` to get the number of control signals after placement

↓

Is the Hold Fix Detour > 0 ps? — Yes → See **Fixing Setup Violations Due to Hold Detours** (this page)

↓ No

Is the path constrained by Pblock or LOC constraints? — Yes → See **Reviewing and Correcting Physical Constraints** (this page)

↓ No

Is the path crossing an SLR boundary? — Yes → See **Improving the SLR Crossing Performance** (this page)

↓ No

Is the # of control sets > 7.5% the # of CLB registers ÷ 8 — Yes → See **Reducing Control Sets** (this page)

↓ No

Several implementation strategies attempted? — Yes → See **UG949: Design Creation** to identify additional logic optimizations

↓ No

See **Trying Alternative Implementation Flows** (this page)

X21582-091818

REDUCING NET DELAY TECHNIQUES

Fixing Setup Violations Due to Hold Detours

To ensure the design is functional in hardware, fixing hold violations has higher priority than fixing setup violations (or Fmax). The following example shows a path between two synchronous clocks with high skew with a tight setup requirement:

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

Note: The Hold Fix Detour is in picoseconds. To address the hold detour impact on Fmax, see **Resolving Hold Violations Techniques** (page 4).

Reviewing and Correcting Physical Constraints

All designs include physical constraints. Although I/O locations cannot usually be changed, Pblock and location constraints must be carefully validated and reviewed when making design changes. Changes can move the logic farther apart and introduce long net delays. Review the paths with more than 1 Pblock (PBlocks column) and with location constraints (Fixed Loc column).

Improving the SLR Crossing Performance

When targeting stacked silicon interconnect (SSI) technology devices, making the following early design considerations helps to improve the performance:

- Add pipeline registers at the boundary of major design hierarchies or kernels to help long distance and SLR crossing routing
- Verify that each SLR utilization is within the guidelines (use `report_failfast -by_slr`)
- Use USER_SLR_ASSIGNMENT constraints to guide the implementation tools. See **UG949: Using Soft SLR Floorplan Constraints**.
- Use SLR Pblock placement constraints if the soft constraints do not work
- Use `phys_opt_design -slr_crossing_opt` after placement or after routing

Reducing Control Sets

Try reducing the number of control sets when their number is over the guideline (7.5%), either for the entire device or per SLR:
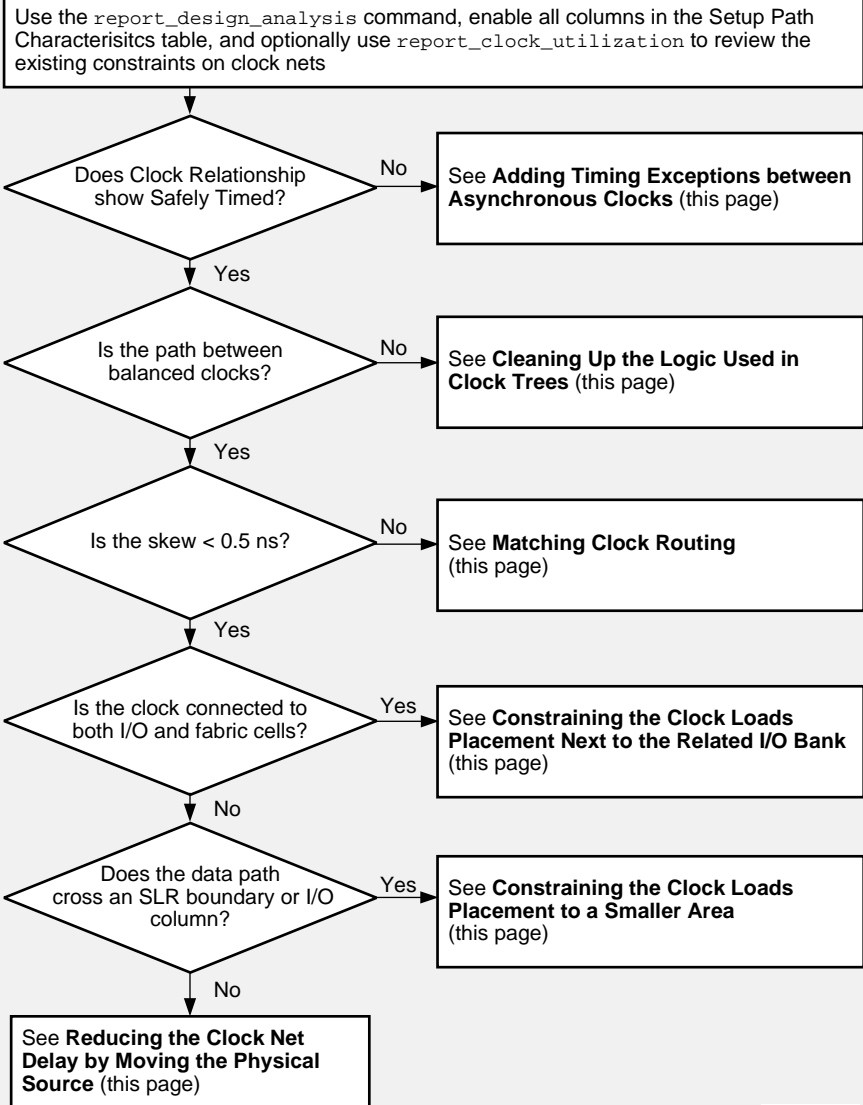
- Remove MAX_FANOUT attributes on clock enable, set, or reset signals in RTL
- Increase the minimum synthesis control signal fanout (e.g., `synth_design -control_set_opt_threshold 16`)
- Merge the replicated control signals with `opt_design -control_set_merge` or `-merge_equivalent_drivers`
- Remap low fanout control signals to LUTs by setting the CONTROL_SET_REMAP property on CLB register cells

Trying Alternative Implementation Flows

The default compilation flow provides a quick way to obtain a baseline of the design and start analyzing the design if timing is not met. If timing is not met after initial implementa1tion, try some of the other recommended flows:

- Try several `place_design` directives (up to 10), and several `phys_opt_design` iterations (Aggressive*, Alternate* directives)
- Overconstrain the most critical clocks (up to 0.500 ns) during `place_design/phys_opt_design` using `set_clock_uncertainty`
- Increase the timing QoR priority on timing clocks that must meet timing using `group_path -weight`
- Use the incremental compilation flow after minor design modifications to preserve QoR and reduce runtime

XILINX.

IMPROVING CLOCK SKEW FLOW

Use the `report_design_analysis` command, enable all columns in the Setup Path Characterisitcs table, and optionally use `report_clock_utilization` to review the existing constraints on clock nets

Does Clock Relationship show Safely Timed? — No → See **Adding Timing Exceptions between Asynchronous Clocks** (this page)

↓ Yes

Is the path between balanced clocks? — No → See **Cleaning Up the Logic Used in Clock Trees** (this page)

↓ Yes

Is the skew < 0.5 ns? — No → See **Matching Clock Routing** (this page)

↓ Yes

Is the clock connected to both I/O and fabric cells? — Yes → See **Constraining the Clock Loads Placement Next to the Related I/O Bank** (this page)

↓ No

Does the data path cross an SLR boundary or I/O column? — Yes → See **Constraining the Clock Loads Placement to a Smaller Area** (this page)

↓ No

See **Reducing the Clock Net Delay by Moving the Physical Source** (this page)

X21583-091818

IMPROVING CLOCK SKEW TECHNIQUES

Adding Timing Exceptions Between Asynchronous Clocks

Timing paths in which the source and destination clocks originate from different primary clocks or have no common node must be treated as asynchronous clocks. In this case, the skew can be extremely high, making it impossible to close timing. Add `set_clock_groups`, `set_false_path` and `set_max_delay –datapath_only` constraints as needed. For details, see **UG949: Adding Timing Exceptions Between Asynchronous Clocks**.

Cleaning Up the Logic Used in Clock Trees

The `opt_design` command automatically cleans up clock trees unless DONT_TOUCH constraints are used on the clocking logic. Select the timing path, enable the **Clock Path Visualization** toolbar button 🔛, and open the schematic (**F4**) to review the clock logic.

- Avoid timing paths between cascaded clock buffers by eliminating unnecessary buffers or connecting them in parallel. For example:
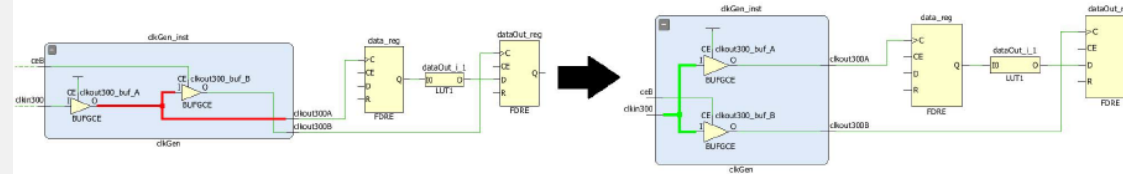
- Combine parallel clock buffers into a single clock buffer unless the clocks are not equivalent.
- Remove LUTs or any combinatorial logic in clock paths, which can make clock delays and clock skew unpredictable.

Matching Clock Routing

Use the CLOCK_DELAY_GROUP to improve clock routing delay matching between critical synchronous clocks, even when the two clock nets already have the same CLOCK_ROOT. The following example shows two synchronous clocks without the CLOCK_DELAY_GROUP:

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

Constraining the Clock Loads Placement Next to the Related I/O Bank

For clocks between I/O logic and fabric cells with less than 2,000 loads, set the CLOCK_LOW_FANOUT property on the clock net to automatically place all the loads in the same clock region as the clock buffer (BUFG*) and keep insertion delay and skew low.

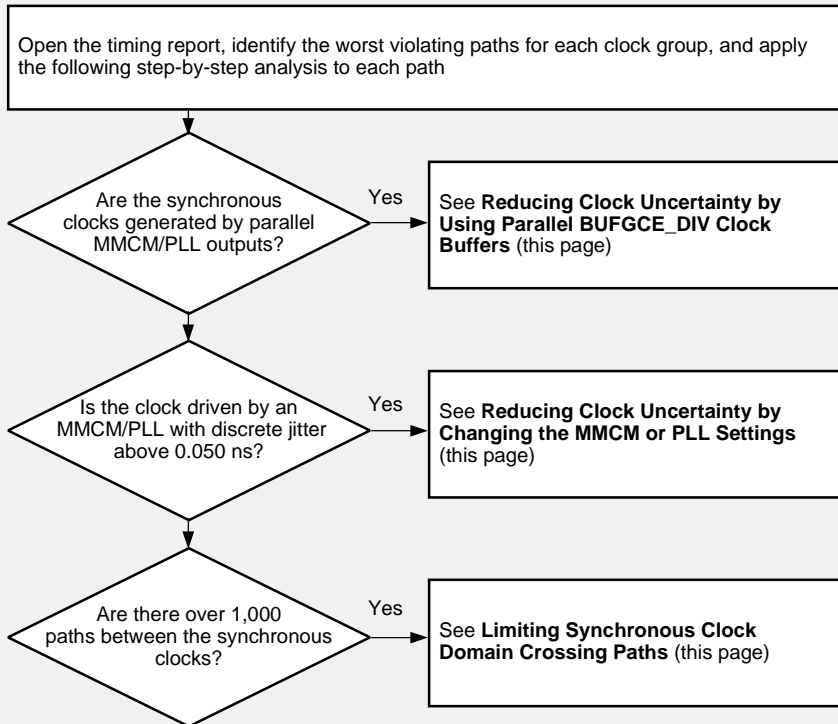### Constraining the Clock Loads Placement to a Smaller Area

You can use Pblocks to force the placement of clock net loads in a smaller area (e.g., 1 SLR) to reduce insertion delay and skew or to avoid crossing special columns, such as I/O columns that introduce a skew penalty.

Reducing the Clock Net Delay by Moving the Physical Source

Use a location constraint to move the source mixed-mode clock manager (MMCM) or phase-locked loop (PLL) to the center of the clock loads to reduce the maximum clock insertion delay, which results in lower clock pessimism and skew. For details, see **UG949: Improving Skew in UltraScale and UltraScale+ Devices**.

XILINX.

IMPROVING CLOCK UNCERTAINTY FLOW

Clock uncertainty is the amount of input jitter, system jitter, discrete jitter, phase error, or user-added uncertainty, which is added to the ideal clock edges to model the hardware operating conditions accurately. Clock uncertainty impacts both setup and hold timing paths and varies based on the resources used in the clock trees.

Open the timing report, identify the worst violating paths for each clock group, and apply the following step-by-step analysis to each path

Are the synchronous clocks generated by parallel MMCM/PLL outputs? — Yes → See **Reducing Clock Uncertainty by Using Parallel BUFGCE_DIV Clock Buffers** (this page)

Is the clock driven by an MMCM/PLL with discrete jitter above 0.050 ns? — Yes → See **Reducing Clock Uncertainty by Changing the MMCM or PLL Settings** (this page)

Are there over 1,000 paths between the synchronous clocks? — Yes → See **Limiting Synchronous Clock Domain Crossing Paths** (this page)

X21581-091818

IMPROVING CLOCK UNCERTAINTY TECHNIQUES

Reducing Clock Uncertainty by Using Parallel BUFGCE_DIV Clock Buffers

For synchronous clocks with a period ratio of 2, 4, or 8 generated by the same MMCM or PLL and driven by several clock outputs, use only 1 MMCM or PLL output and connect it to parallel BUFGCE_DIV clock buffers (UltraScale™ and UltraScale+™ devices only). This clock topology eliminates the MMCM or PLL phase error that results in 0.120 ns clock uncertainty in most cases.
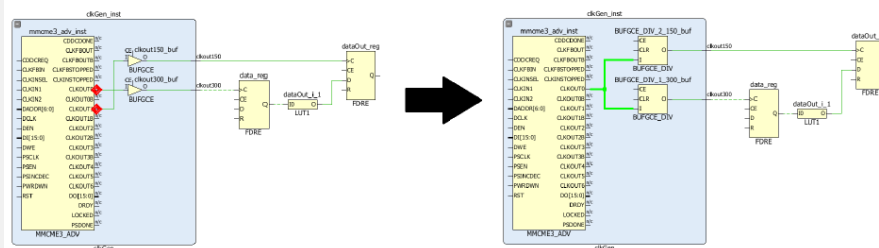
Following is an example of a clock uncertainty reduction for clock domain crossing (CDC) paths between a 150 MHz clock and a 300 MHz clock:

- **Clock Uncertainty Before**: 0.188ns (setup), 0.188ns (hold)
- **Clock Uncertainty After**: 0.068ns (setup), 0.000ns (hold)

Use the Clocking Wizard to generate the clock topology with parallel BUFGCE_DIV buffers, and set the CLOCK_DELAY_GROUP property on the clocks.

Reducing Clock Uncertainty by Changing the MMCM or PLL settings

Clock modifying blocks, such as the MMCM and PLL, contribute to clock uncertainty in the form of discrete jitter and phase error.

- In the Clocking Wizard or using the `set_property` command, increase the voltage-controlled oscillator (VCO) frequency by modifying the M (multiplier) and D (divider) values. For example, MMCM (VCO=1 GHz) introduces 167 ps jitter and 384 ps phase error versus 128 ps and 123 ps for MMCM (VCO=1.43 GHz).
- If possible, use a PLL instead of an MMCM, because PLLs introduce less clock uncertainty.

Limiting Synchronous Clock Domain Crossing Paths

Timing paths between synchronous clocks that are driven by separate clock buffers exhibit higher skew, because the common clock tree node is located before the clock buffers, resulting in higher pessimism in the timing analysis. As a result, it is more challenging to meet both setup and hold requirements at the same time on these paths, especially for high frequency clocks (over 500 MHz). To identify the number of paths between two clocks, use `report_timing_summary` (Inter-Clock Paths section) or `report_clock_interaction`. The following example shows a design that contains many paths between two high speed clocks (requirement = 1.592 ns). 30% of these paths fail timing, which indicates that they are particularly difficult to implement.

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

Review the logic involved in the clock domain crossings and remove unnecessary logic paths, or try the following modifications:

- Add multicycle path constraints on the paths controlled by clock enable, because new data are not transferred every cycle.
- Replace the crossing logic with asynchronous crossing circuitry and appropriate timing exceptions at the expense of extra latency. For example, use asynchronous FIFOs or XPM_CDC parameterized macros. For details, see the *UltraScale Architecture Libraries Guide* ([UG974](#)).

XILINX

FAILFAST REPORT OVERVIEW

In the failfast report, address the checks marked as REVIEW to improve implementation and timing closure. Following are the different sections of the failfast report:

1. **Design Characteristics**: The default utilization guidelines are based on SSI technology devices and can be relaxed for non-SSI technology devices. Designs with one or more REVIEW checks are feasible but are difficult to implement.
2. **Clocking Checks**: These checks are critical and must be addressed.
3. **LUT and Net Budgeting**: Use a conservative method to better predict which logic paths are unlikely to meet timing after placement with high device utilization.

```
                                          1              2              3

+-----------------------------------+-----------+--------+--------+
| Criteria                          | Guideline | Actual | Status |
+-----------------------------------+-----------+--------+--------+
| LUT                               | 70%       | 45.14% | OK     |
| FD                                | 50%       | 57.66% | REVIEW |
| LUTRAM+SRL                        | 25%       | 29.51% | REVIEW |
| CARRY8                            | 25%       | 7.82%  | OK     |
| MUXF7                             | 15%       | 0.82%  | OK     |
| DSP48                             | 80%       | 93.99% | REVIEW |
| RAMB/FIFO                         | 80%       | 61.39% | OK     |
| URAM                              | 80%       | 0.00%  | OK     |
| DSP48+RAMB+URAM (Avg)             | 70%       | 77.69% | REVIEW |
| DONT_TOUCH (cells/nets)           | 0         | 8848   | REVIEW |
| Control Sets                      | 14778     | 21154  | REVIEW |
| Average Fanout for modules > 100k cells | 4   | 2.81   | OK     |
| Non-FD high fanout nets > 10k loads     | 0   | 0      | OK     |
+-----------------------------------+-----------+--------+--------+
| TIMING-6 (No common primary clock b...) | 0   | 0      | OK     |
| TIMING-7 (No common node between re...) | 0   | 0      | OK     |
| TIMING-8 (No common period between ...) | 0   | 0      | OK     |
| TIMING-14 (LUT on the clock tree)       | 0   | 0      | OK     |
| TIMING-35 (No common node in paths ...) | 0   | 0      | OK     |
+-----------------------------------+-----------+--------+--------+
| Number of paths above max LUT budgeting | 0   | 0      | OK     |
| Number of paths above max Net budgeting | 0   | 14     | REVIEW |
+-----------------------------------+-----------+--------+--------+
                                                      X21613-092118
```

FAILFAST REPORT USAGE EXAMPLES

Pblock-Based and SLR-Based Analysis

The `report_failfast` script reports the utilization of the specified physical area or SLR as follows:

- **Before placement**: Use `-pblock <pblockName>` to report on floorplanning constraints. This is especially important for reviewing SLR placement constraints early in the design cycle when SLR Pblocks exist.
- **After placement**: Use `-slr <slrName>` or `-by_slr` to report utilization metrics for each SLR.

Floorplanning What-If Analysis

Use `-top` or `-cell <hierCellName>` with `-pblock <pblockName>` to report utilization metrics and identify good floorplanning constraints without changing the cells to the Pblock.

Failfast Report Checks Marked as REVIEW Analysis

When you use the `-detailed_report <prefix>` option, `report_failfast` generates additional detailed reports for each check that does not meet the guideline (except for resource utilization checks). Review each of the following reports:

- `<prefix>.TIMING.rpt`: Detailed Methodology TIMING-* violations
- `<prefix>.AVGFO.rpt`: Average Fanout for modules bigger than 100,000
- `<prefix>.HFN.rpt`: Non-FD high fanout net (HFN) driving more than 10,000 loads
- `<prefix>.DONT_TOUCH.rpt`: List of cells/nets with DONT_TOUCH property set
- `<prefix>.timing_budget_LUT.rpt`: Detailed timing paths failing the LUT budgeting
- `<prefix>.timing_budget_LUT.rpx`: Detailed timing paths failing the LUT budgeting (Vivado IDE interactive report)
- `<prefix>.timing_budget_Net.rpt`: Detailed timing paths failing the net budgeting
- `<prefix>.timing_budget_Net.rpx`: Detailed timing paths failing the net budgeting (Vivado IDE interactive report)

Kernel-Level or Module-Level Analysis

Synthesize each kernel or major design hierarchy in an out-of-context mode. Then, verify that timing is met with estimated delays and with a realistic clock constraint:

- Review the timing reports and address any failing paths
- Review the logic level budgeting section of `report_failfast` to identify the paths that are more likely to fail timing after placement. Optimize the paths flagged by this analysis by modifying your design.

Pre-Implementation Design Analysis

After all kernels, sub-modules, and the top-level design are assembled and synthesized, review and address all checks flagged as REVIEW.

Pre-Implementation Floorplan Constraints Analysis

For large designs and for SDx designs, verify that the design architecture and hierarchies fit the device floorplan appropriately.

Post-Placement SLR Utilization Analysis

Use `report_failfast -by_slr` to verify that the resource utilization in each SLR is within the recommended guidelines.

TIP: In Project Mode, add the failfast report with the following Tcl hook:

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

Following is an example of `postopt_failfast.tcl`:

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