



VIVADO EXPERT SERIES | 2017



High Speed Design Closure Techniques

Balachander Krishnamurthy

VIVADO[®]

HLx Editions

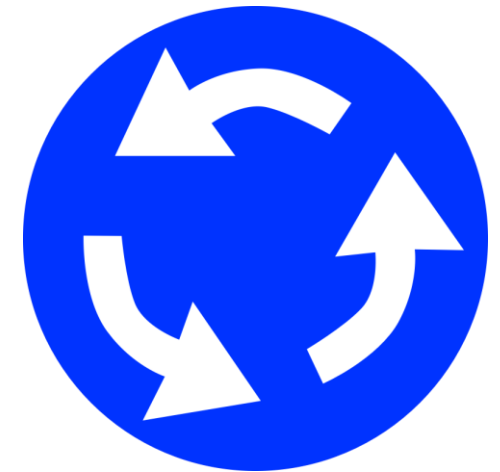
Agenda

- High speed design challenges
- Design analysis
- Design guidelines
- Complexity and congestion analysis
- Summary

VIVADO EXPERT SERIES | 2017


High Speed Design Challenges

- High performance requirement (>400 MHz)
- Issues detected too late
- RTL coding style
- Constraints not clean
- Too many logic levels
- Too many high fanout nets and too many control sets
- Sub-optimal clocking and big blocks (BRAM, DSP)
 - Trade-off for power vs performance
- Complexity and Congestion
- Many timing closure iterations



UltraFast Design Methodology (UG949)



- Iterate after synthesis for faster analysis and predictable results
 - RTL Coding style and examples
- Recommendation with checklists in all areas:
 - Board and device planning
 - Design creation
 - Implementation
 - Design Closure
-  ■ Run Report Methodology to identify and fix issues early
 - Synthesis, Timing, XDC and Clocking methodology checks



UG1231: UltraFast Design Methodology Quick Reference Guide

UltraFast Design Methodology Quick Reference Guide (UG1231)

INTRODUCTION

The UltraFast™ Design Methodology is a set of best practices recommended by Xilinx to maximize productivity and reduce design iterations of complex systems, including embedded processor subsystems, analog and digital processing, high-speed connectivity, and network processing. See the *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949) for more information.

The UltraFast Design Methodology Checklist (XTP301) includes common questions that highlight typical areas where design decisions have downstream consequences and draws attention to potential problems that are often unknown or ignored. It provides easy access to related collateral. The checklist is available within the Xilinx Documentation Navigator tool (DocNav).

This quick reference guide highlights key design methodology steps to achieve quicker system integration and design implementation and to derive the greatest value from Xilinx® devices and tools. Pointers to related collateral are also provided. The main design tasks covered in this guide include:

- Board and Device Planning
- Design Entry and Implementation
- Top-Level Design Validation
- Design Analysis
- Design Closure

Refer to the UltraFast Design Methodology – System-Level Design Flow available within the Xilinx Documentation Navigator tool (DocNav) for pointers to all design hubs and specific collateral.



UG1231 (v2018.3) November 11, 2018

BOARD AND DEVICE PLANNING

PCB Designer

Examine Key Interfaces

- Validate part orientation and key interfaces

Examine the PCB Layout

- Perform the Memory Interface and Transceiver Checklists
- Follow PCB layout recommendations
- Ensure final FPGA pinout is signed off by FPGA designer

Review the Schematic

- Complete PCB Checklist review
- Check PDS, configuration, and power supplies
- Validate I/O state before, during, and after configuration

Manufacture and Test

- Verify the configuration sequence, power supplies, and I/O performance with the test I/O project

See Also:
[UG949: Board and Device Planning PCB Design Checklist](#)
[Memory Interface IP Design Checklists](#)
[Schematic Design Checklists](#)

FPGA Designer

Analyze Device for Pinout

- Examine transceiver and bonded I/O locations
- Examine SSI technology I/O planning
- Validate part orientation and key interfaces

Define I/O Pinouts for Key Interfaces

- Create I/O planning projects
- Define and validate memory controllers, GTs, and PCIe® technology locations
- Establish a clocking skeleton
- Minimize floorplan distance between connected IP

Define Final Pinout

- Merge interface projects into a final I/O project
- Validate DRCs and SSN analysis
- Implement design to check clocking and I/O rules
- Use the final I/O project for production test

Estimate Power

- Determine power budget and thermal margin using Xilinx Power Estimator (XPE)
- Apply toggle rates using knowledge of prior designs

See Also:
[UG949: Board and Device Planning Power Estimation and Optimization Design Hub](#)
[I/O and Clock Planning Design Hub](#)

DESIGN ENTRY AND IMPLEMENTATION

Logic Designer

Define a Good Design Hierarchy

- Define relevant hierarchies to help global placement and floorplanning
- Insert I/O and clock components near the top level
- Add registers at main hierarchical boundaries
- Generate IP and review target device utilization

Build and Validate RTL Submodules

- Ensure design adheres to RTL coding guidelines
- Add sufficient registers around DSP and memories
- Use control signals only when absolutely necessary
- Use synthesis attributes to control final logic mapping
- Create simple timing constraints to review estimated timing and address paths with too many logic levels
- Review synthesis log files, utilization report, and elaborated view to identify sub-optimal mapping
- Run Methodology and RTL checks and review issues
- Implement the submodule in out-of-context (OOC) mode to validate implemented performance
- Review utilization and power against original budget
- Simulate the design to validate functionality

Assemble and Validate Top-Level Design

- Synthesize the top-level RTL design and resolve all connectivity issues
- Review top-level utilization and clocking guidelines
- Create and validate top-level constraints
- Iterate the RTL and constraints to fix Methodology and DRC issues and meet timing
- Proceed to implementation

See Also:
[UG949: Design Creation and Implementation Designing with IP Design Hub](#)
[Using IP Integrator Design Hub](#)
[Logic Synthesis Design Hub](#)
[Applying Design Constraints Design Hub](#)
[Implementation Design Hub](#)

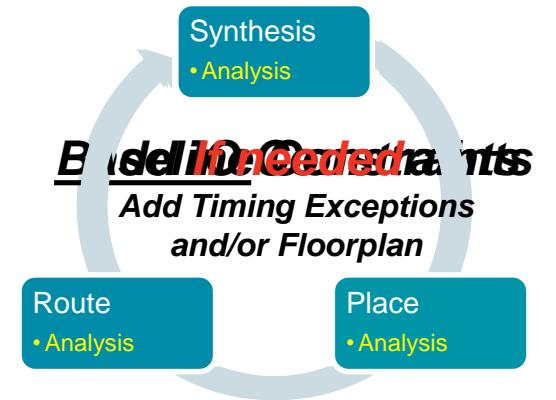
Design Analysis

Constraints, logic levels, control sets and high fanout nets

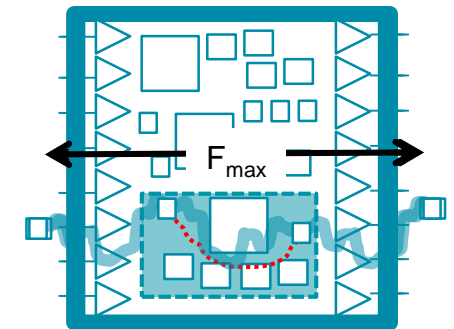
Baselining – Clean Constraints

- **Baselining: Ensure timing constraints are clean**

- All clocks and clock relationships are defined
- Asynchronous clock domain crossings are constrained correctly
 - 1-bit CDC's: false path or clock group constraints
 - Multi-bit CDCs: Use set_max_delay –datapath_only and/or set_bus_skew
- IO constraints
- Other point-to-point exception constraints



Optimize Entire Design



Baselined CDC

Logic Levels

- Review number of logic levels

- Guideline: 500 ps per logic level (1 LUT + 1 net)
- Analyze with:

```
report_design_analysis -logic_level_distribution -extend
```

- Recommendations:

- Modify RTL to reduce the number of logic levels
- Use instance based synthesis to optimize modules
 - Synthesis Options: retiming, area, performance, routability, etc.

Speedgrade	-1	-2	-3
7 Series	575ps	500ps	425ps
UltraScale	490ps	425ps	360ps
UltraScale+	350ps	300ps	250ps
UltraScale+ LV	490ps	425ps	360ps

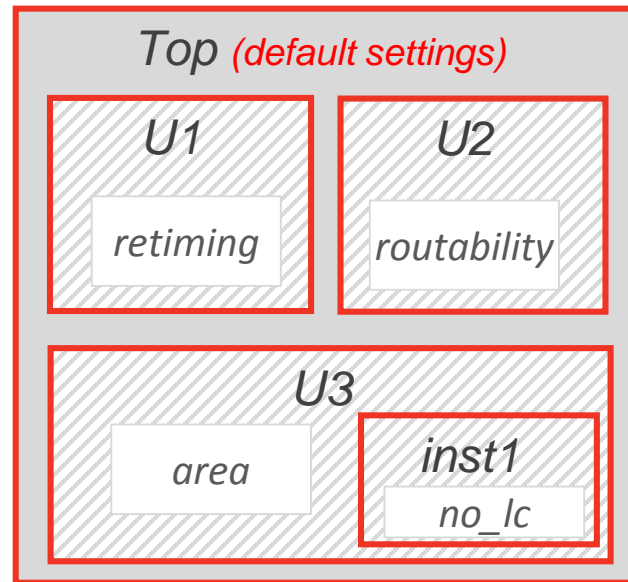
Instance Based Synthesis Options

■ Instance-based optimization using XDC

```

set_property BLOCK_SYNTH.RETIMING 1 [get_cells U1]
set_property BLOCK_SYNTH.STRATEGY {ALTERNATE_ROUTABILITY} [get_cells U2]
set_property BLOCK_SYNTH.STRATEGY {AREA_OPTIMIZED} [get_cells U3]
set_property BLOCK_SYNTH.LUT_COMBINING [get_cells U3/inst1]
    
```

- Nested options supported
- Strategies are presets
- Multiple logic options allowed



Instance Based Synthesis Settings

Strategies
(bundled preset)

Default
Area
Performance
Routability
retiming
adder_threshold
comparator_threshold
shreg_min_size
fsm_extraction
lut_combining
flatten_inside_partition
extract_partition
flatten_hierarchy
control_set_threshold
max_lut_input
muxf_mapping
keep_equivalent_register

Logic Options

Strategies and logic options applicable to instances

Control Sets

Analyze with `report_utilization` and `report_control_sets -verbose`

▪ Control sets can impact placement and timing

▪ Guideline

- Acceptable if unique control sets < 7.5% of total slices
- Analysis needed if unique control sets > 15% total slices

▪ Recommendations

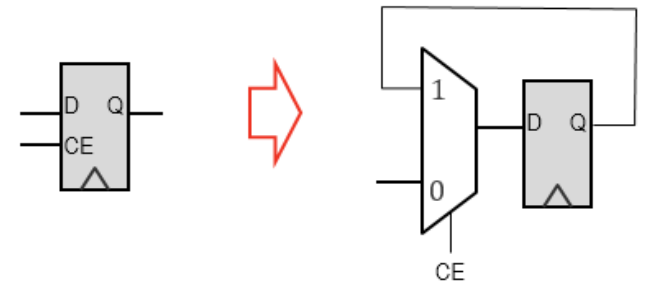
- Manually replicate registers with high fanout or PhysOpt
- Global synthesis option to change defaults (`-control_set_opt_threshold`)
 - Different thresholds based on architecture
 - ✓ 4 for 7-series and 2 for UltraScale and UltraScale+
- Implement timing critical low fan-out (<8) control signals in fabric
 - By using `extract_enable` or `extract_reset` in RTL
 - In XDC file:

```
set_property extract_enable/reset "no" [get_cells U1/tmp_reg]
```

```
module top (input clk,
            input [9:0] din, a, b
            (* extract_enable = "no" *)
            output reg [9:0] dout);

    reg [9:0] dinr, ar, br;
    wire en = ar = br;

    always @(posedge clk)
    begin
        ar <= a;
        br <= b;
        dinr <= din;
        if(en)
            dout <= dinr;
    end
endmodule
```



CE emulation with logic on D input side

High Fanout Nets

Analyze by running `report_high_fanout_nets`

▪ RTL Recommendations:

- If timing is not critical and high fanout nets feed FFs directly

- Promote nets with fanout >25k to global clocking

```
set_property CLOCK_BUFFER_TYPE BUFG [get_nets netName]
```

- Timing is Critical

- Replicate registers in RTL with KEEP attribute

- Replicate in each SLR with pblock constraints

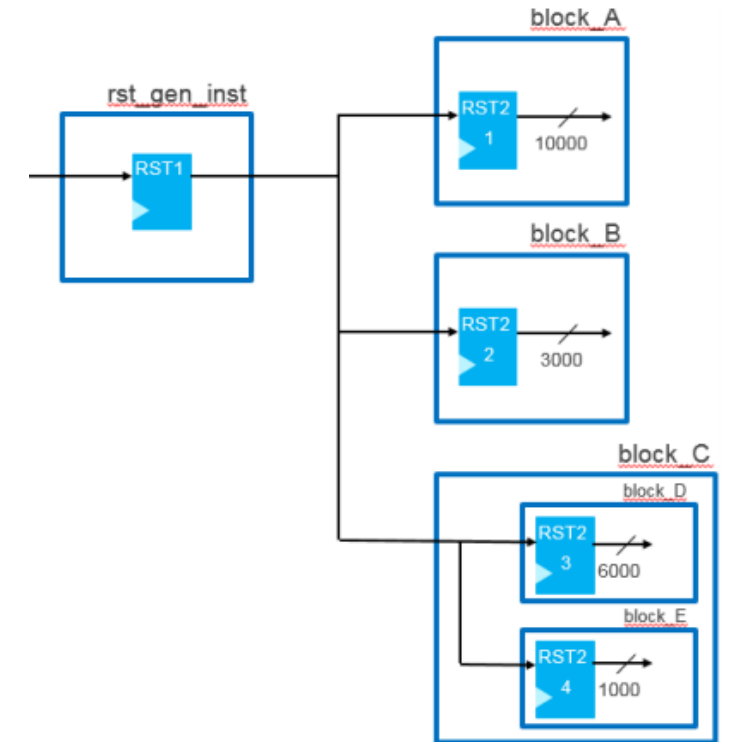
- Avoid high fanout from/to DSP/BRAM

▪ Synthesis Recommendations:

- Avoid MAX_FANOUT for global control signals
- Use MAX_FANOUT only inside small module if needed

▪ Replicate high fanout net drivers with PhysOpt

- `Phys_opt_design -force_replication_on_nets`
- `Phys_opt_design -directive AggressiveExplore`



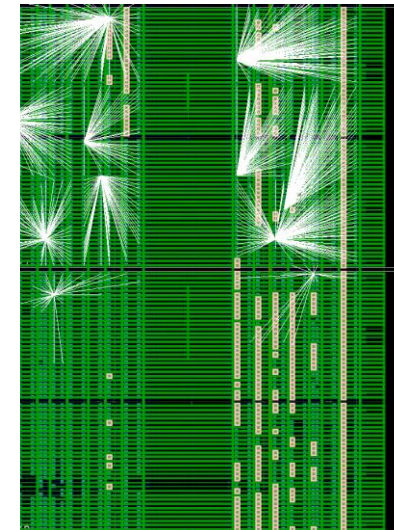
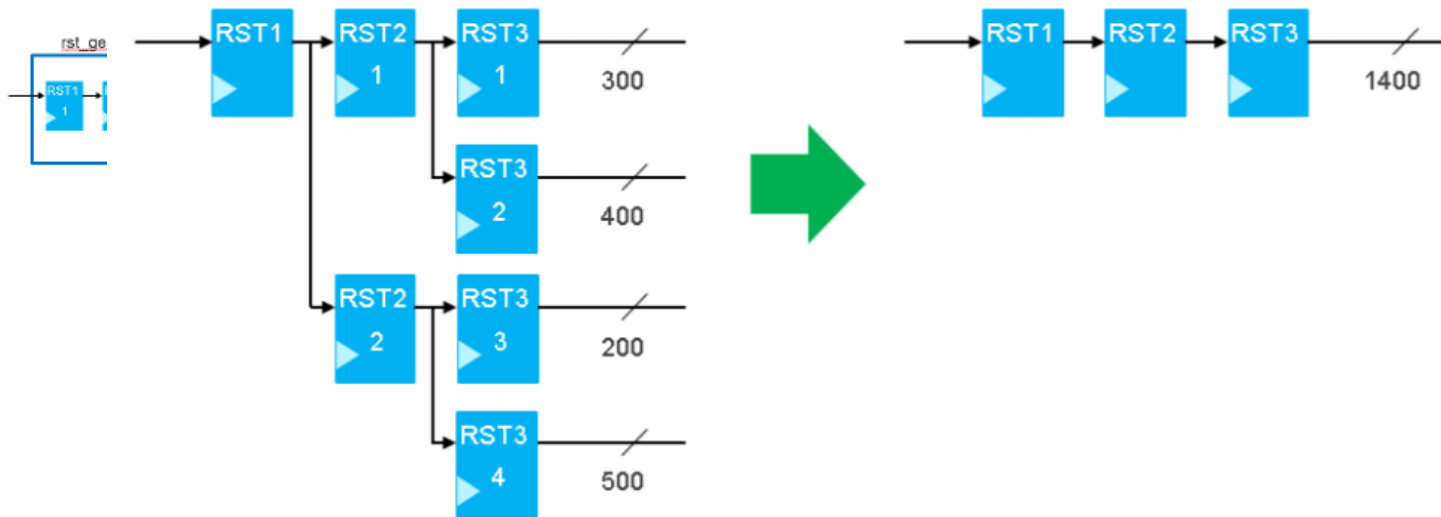
High Fanout Nets Optimization Flow

- **Synthesis: Limited Replication**
 - Avoid MAX_FANOUT for global control signals
 - Use MAX_FANOUT only inside small module if needed
 - Use KEEP on manually replicated cells
- **Opt Design: Coarse-grained Replication**
 - Based on large design hierarchies
- **Place Design: Mid-grained Replication**
 - Based on early placement and timing information
- **Phys Opt Design: Fine-grained Replication**
 - Based on accurate timing information

High Fanout Nets Optimization in Opt Design

■ Opt Design

- Insert, merge and split BUFGs on high fanout nets (Done by default)
- Merge low fanout control signals in opt_design with '-control_set_merge'
- Merge all LUT and Flop equivalent drivers with '-merge_equivalent_drivers'
- Control module based replication with '-hier_fanout_limit <number>'



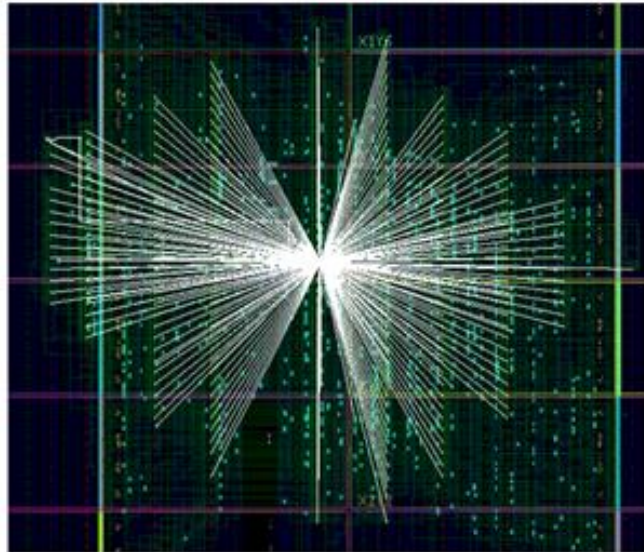
High Fanout Nets Optimization in Placer

- Physical Synthesis in Placer (PSIP)
 - Automatically inserts BUFGs during global placement based on resource availability
 - Replicates high fanout nets during placement with '-fanout_opt'
 - Replicates low/medium fanout nets connected to DSP/BRAM
- Advantages
 - No guess needed work at RTL/Synthesis level
 - Optimal control set usage
 - Placement is known and replication is based on driver and load placement
 - Addresses timing critical HFN early in placer
 - Reduces need for post placement physical optimization

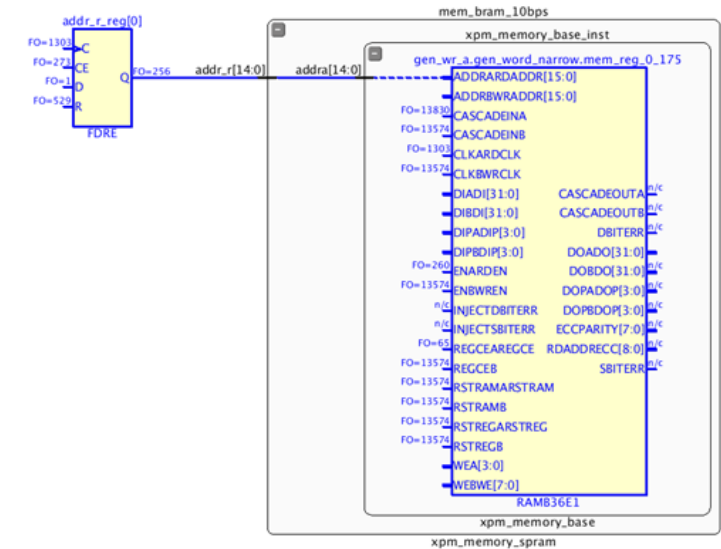
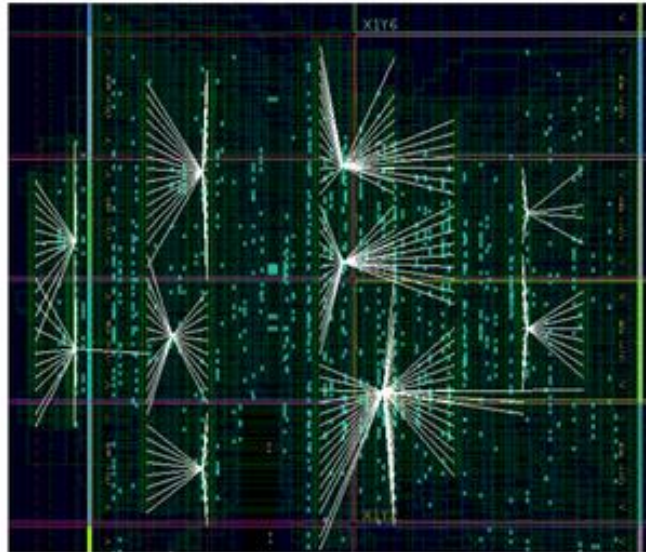
PSIP Example : Control Signal

- FF driving address bits of large BRAM

Without replication in placer
WNS = -0.312 ns



With replication in placer
WNS = 0.146 ns



Design Guidelines

Clocking, Synchronous CDCs, BRAM and DSPs

Clocking Guidelines

- Sub-optimal clocking hurts timing
 - Review and fix clocking methodology checks
- Reduce clock buffer utilization, where possible
 - Reduce clocks by combining identical synchronous clocks
 - Remove buffer from MMCM feedback path (no delay compensation support)
 - Remove cascaded buffers unless absolutely required
 - Remove buffer from MMCM feedback path when in INTERNAL mode
- Floorplan clocks to keep
 - Clock with low fanout close to the driver
 - Away from highly utilized areas
- Avoid clocks routed on fabric resources
 - Unpredictable skew, competes with routing resources with other nets
 - Insert clock buffers, if possible (loads >30)

Guidelines for synchronous CDC

■ Analyze critical crossings

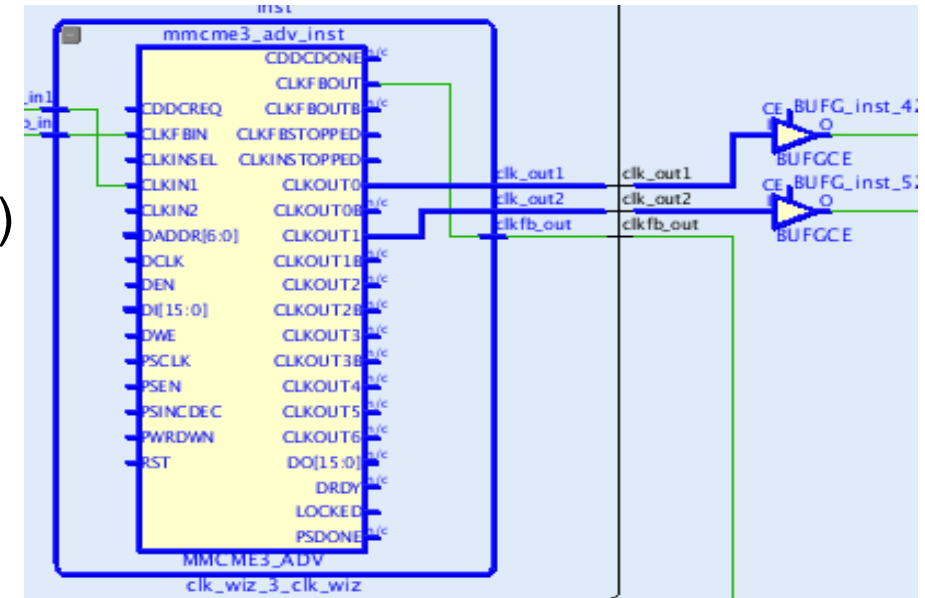
- Run report_timing_summary or report_clock_interaction
- Tight setup requirements (more sensitive to skew)
- Highest number of paths (largest impact on TNS & THS)

■ Review and improve CDC clock topologies

- Configure VCO of MMCM to run at highest frequency
 - Reduces clock uncertainty
- Avoid common node before MMCM
- Avoid cascaded buffers
- Avoid crossing IO/SLR boundaries to minimize skew

■ Reduce/relax CDC paths

- Treat as asynchronous (requires proper synchronization circuit)
- Use multicycle path constraints (relax setup) (requires CE logic)



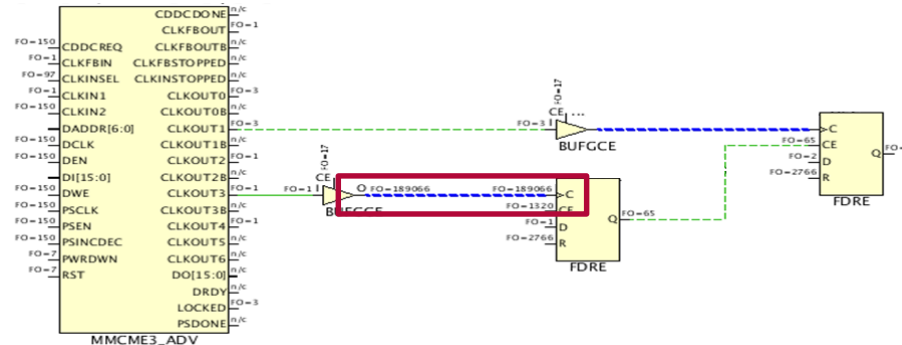
Clock Skew in Synchronous CDCs

CLOCK_DELAY_GROUP constraint to limit skew on synchronous clocks

▪ Match clock net delays in a group

- Useful for constraining synchronous CDC skew
- Better accuracy than using USER_CLOCK_ROOT constraint

▪ Example:



```
set_property CLOCK_DELAY_GROUP group_0 [get_nets {u1/clk_50M u1/clk_100M}]
```

▪ Guidelines

- Apply to net segment directly connected to buffer output
- Buffers must have same driving cell => ensures balanced topology
 - Driver examples: MMCM, PLL, IBUFDS, GT_CHANNEL

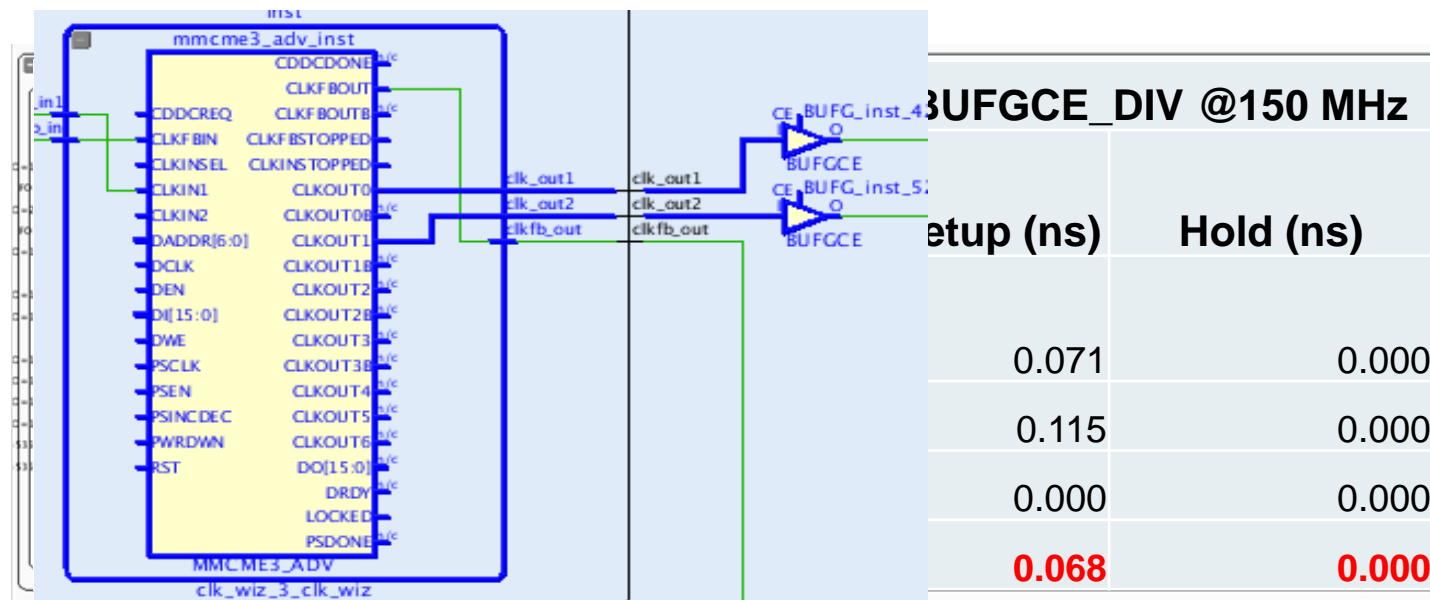
Leverage BUFGCE_DIV in UltraScale devices

- **Use BUFGCE_DIV to perform division**

- Removes phase error ($\sim 120\text{ps}$) \Rightarrow helps both setup and hold on CDC paths
- Clocks with simple period ratio (/1 /2 /4 /8)
- Only 4 BUFGCE_DIV per clock region

- **Beware of cell delay difference between BUFGCE and BUFGCE_DIV**

- Best if both clocks use same buffer type (BUFGCE_DIV can divide by 1)



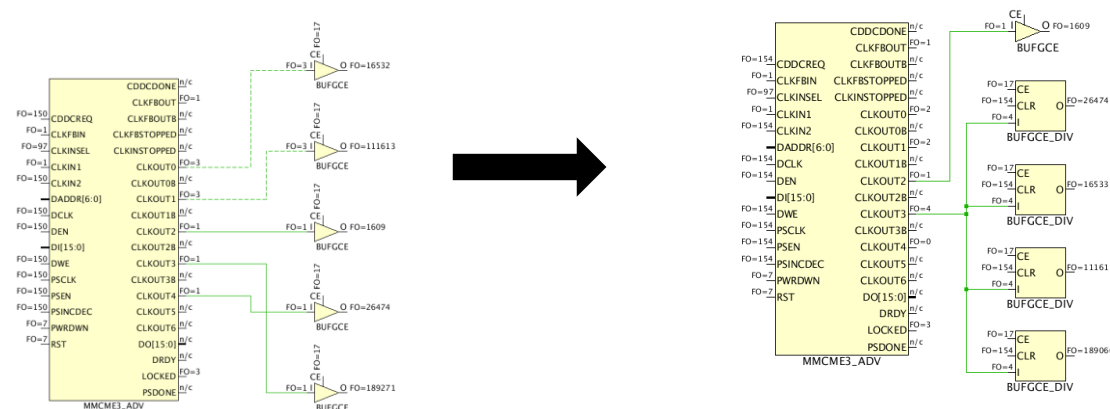
Real customer design - Case Study

- Design with high speed clocks and many synchronous CDCs
- Identify critical CDC paths with report_clock_interaction (pre-route)

Source Clock	Destination Clock	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	WHS (ns)	THS (ns)	Failing Endpoints (THS)	Total Endpoints (THS)	Path Req (WHS)	Common Primary Clock	Inter-Clock Constraints
sys_clk_491	sys_clk_491	0.165	0.000	0	430480	2.034	-0.253	-289.459	10007	430480	0.000	Yes	Partial False Path
sys_clk_245	sys_clk_491	0.916	0.000	0	3292	2.034	-0.383	-271.798	2022	3292	0.000	Yes	Partial False Path
sys_clk_491	sys_clk_245	0.420	0.000	0	3751	2.034	-0.206	-254.735	2687	3751	0.000	Yes	Partial False Path
sys_clk_245	sys_clk_122	0.461	0.000	0	1125	4.069	-0.224	-73.057	619	1125	0.000	Yes	Partial False Path
sys_clk_245	sys_clk_61_44	10.444	0.000	0	3134	16.276	-0.234	-70.723	868	3134	0.000	Yes	Partial False Path
sys_clk_245	sys_clk_245	0.255	0.000	0	238146	4.069	-0.234	-46.686	1802	238146	0.000	Yes	Partial False Path
sys_clk_30_72	sys_clk_61_44	9.222	0.000	0	5348	16.276	-0.430	-41.222	545	5348	0.000	Yes	Partial False Path
sys_clk_30_72	sys_clk_122	27.186	0.000	0	12908	32.552	-0.356	-33.333	415	12908	0.000	Yes	Partial False Path
sys_clk_122	sys_clk_245	0.540	0.000	0	10781	4.069	-0.185	-21.341	580	10781	0.000	Yes	Timed
sys_clk_122	sys_clk_61_44	10.212	0.000	0	1083	16.276	-0.196	-14.464	172	1083	0.000	Yes	Timed

- 4 clocks identified with tightest setup requirement and highest THS violations

- Improve clock tree and apply **CLOCK_DELAY_GROUP**



Block RAM

Guidelines for Paths from/to BRAM

- **Implement smaller memories in the LUTRAM (<8kb)**
- **Avoid high fanout signals from/to big blocks**
 - Replicate in RTL based on hierarchy with KEEP attribute
 - Advanced replication features coming in 2016.3 across the flow
- **Use low-logic-level paths to drive control signals**
 - Replicate address bus as needed or use PhysOpt
 - Recommend cascade_height attribute for performance/power
- **Enable output register for high performance designs**
 - Register should be either inside or directly connected to output pin
- **Analyze block RAM paths with report_design_analysis**

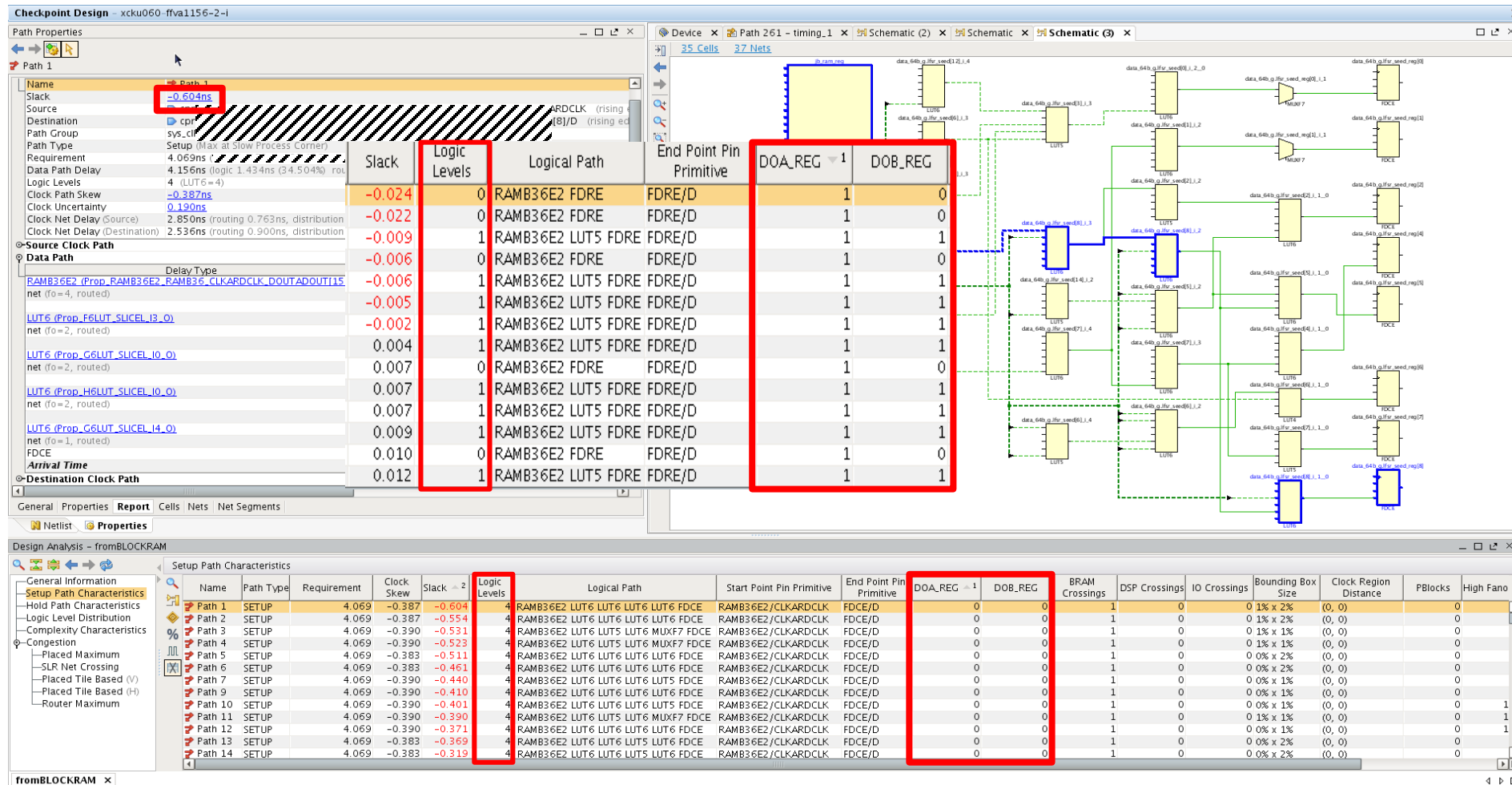
```
...  
(* ram_style = "distributed" *)  
reg [31:0] mem [(2**8)-1:0];  
reg [7:0] addr_reg;  
...
```

```
...  
(* ram_style = "block",  
  cascade_height = 1/8/32 *)  
reg [31:0] mem [(2**15)-1:0];  
reg [14:0] addr_reg;  
...
```

```
report_design_analysis -of_timing_paths [get_timing_paths \  
-from/to [get_cells -hier -filter {PRIMITIVE_GROUP==BLOCKRAM}] \  
-max 1000 -nworst 16 -unique_pins] -file <from><to>BLOCKRAM.rpt
```

Real Customer Case Study - Unregistered RAMB output

- Many paths from same RAMB had 4+ levels of logic @ 250MHz



Complexity and Congestion

Design Complexity

Analyze with report_design_analysis -complexity

- **Complexity is measure of logic interconnection**
- **Complex modules may be down a few levels of hierarchy**
 - Change the 'Hierarchical depth' in the GUI

High Rent, average fanout on larger instances

High LUT6%, MUXF* utilization

Instance	Module	Rent	Average Fanout	Total Instances	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	Memory LUT	DSP	RAMP	MUXF
utop	utop	0.51	2.73	38637	701	3507	2932	3016	3957	6128	1585	68	80	1025
mgtEngine (mgtTop)	mgtTop	0.87	1.80	1198	48	48	154	144	72	11	11	0	0	0
cpuEngine (or1200_top)	or1200_top	0.52	3.78	10308	137	598	879	792	928	2461	416	4	22	312
fftEngine (fftTop)	fftTop	0.68	1.34	3315	34	1151	56	42	12	116	769	64	0	0
usbEngine0 (usbf_top)	usbf_top	0.70	4.12	11414	241	845	913	1015	1374	1717	200	0	29	331
usbEngine1 (usbf_top_0)	usbf_top_0	0.71	4.12	11414	241	845	913	1015	1374	1717	200	0	29	331

- **Goal**
 - Identify difficult modules (>15k cells) with high rent (>0.65) and/or high average fanout (>4.0) before placement
- **Solutions**
 - Option 1: Try different synthesis options, adopt a bottom-up synthesis flow (OOC)
 - Option 2: Floorplan the difficult module

Congestion

Analyze with `report_design_analysis -congestion`

- **Physical regions with**

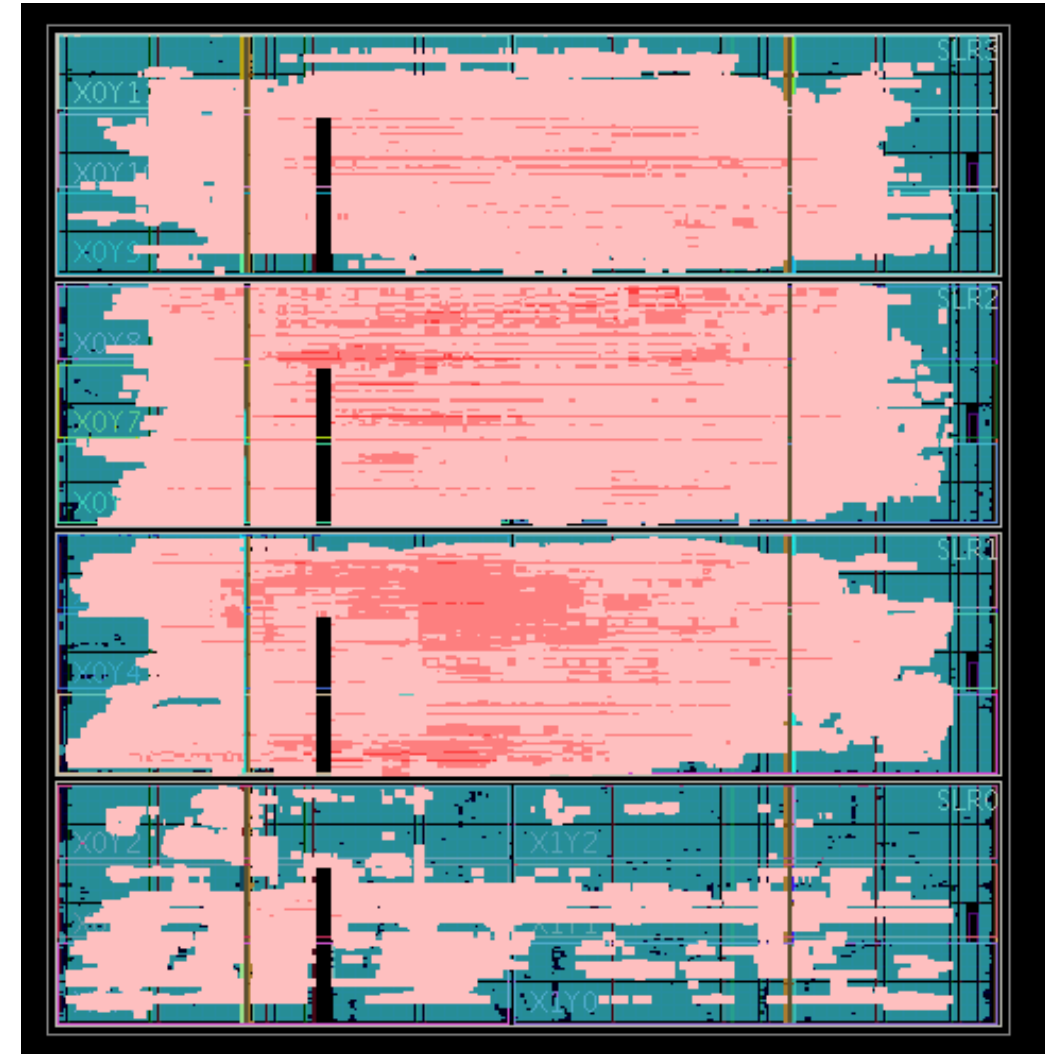
- High pin density
- High utilization of routing resources

- **Placer is Congestion-aware:**

- Balances congestion vs. wire length vs. timing
 - Cannot always eliminate congestion
 - Cannot anticipate congestion due to hold fixing
 - Timing estimation impacted by detours
- Reports congested areas seen by placer

- **Router congestion**

- Routing detours to handle congestion but impacts timing
- Reports areas with routing utilization close to 100%



“Smear” Maps

Placer congestion tends to be more conservative than router XILINX ALL PROGRAMMABLE™

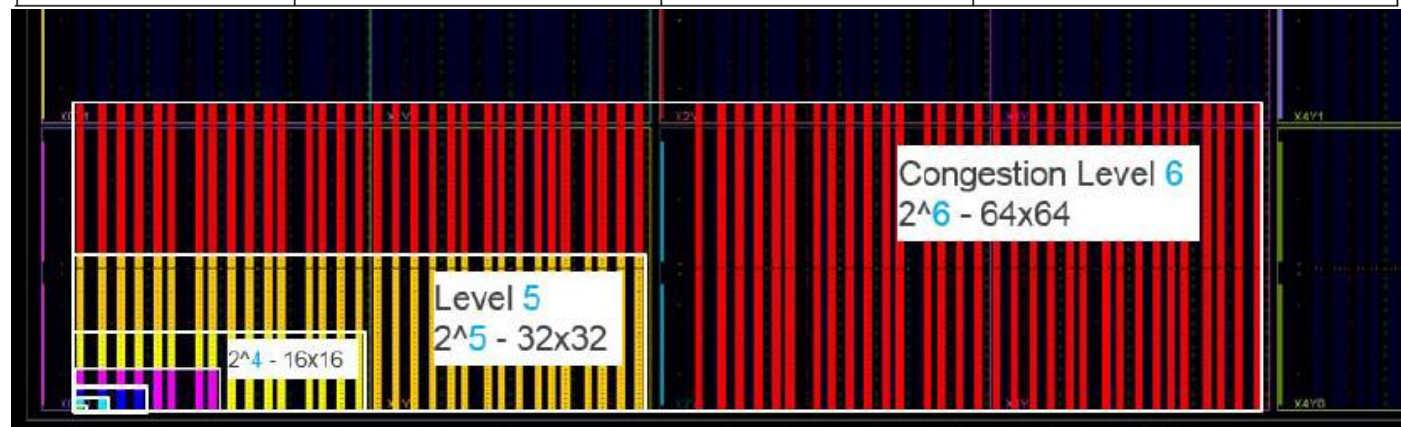
Congestion Levels

■ Definition

- Level corresponds to size of square area that contains congested routing
- Square size based on interconnect (INT_XnYm) or CLB (CLE_M_XnYm)

■ Congestion Level Ranges

Level	Area	Congestion	QoR Impact
1, 2	2x2, 4x4	None	None
3, 4	8x8, 16x16	Mild	Possible QoR degradation
5	32x32	Moderate	Likely QoR degradation
6	64x64	High	Difficulty routing
7,8	128x128, 256x256	Impossible	Likely unroutable



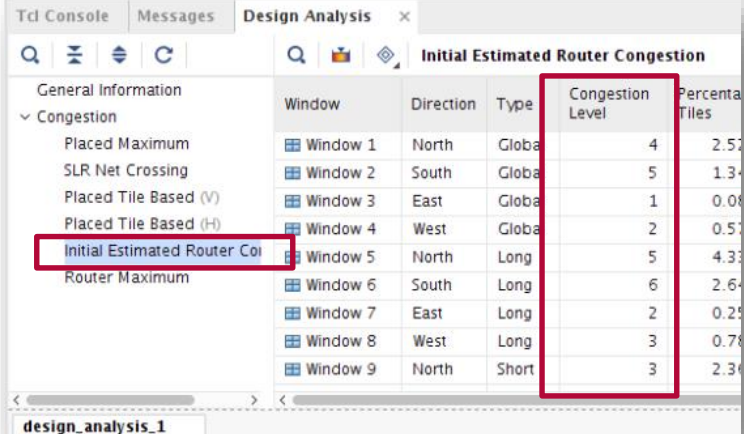
Types of Congestion

- Global: region of high utilization of routing resources
 - Demand for connections into/out of the region is high compared to supply
 - Common causes: High degree of LUT-combining, too many control sets, massive busses, poor floorplanning
 - UltraScale and UltraScale+ analysis often benefit from more detailed models: [Short](#) and [Long](#)
- Short: shortage of short-distance routes: SINGLES, DOUBLES, QUADS
 - More common in UltraScale
 - Common causes: dense concentration of MUXF or CARRY primitives
- Long: shortage of long-distance routes: HLONGs, VLONGs
 - More common in UltraScale+
 - Common causes: module with high Rent/average fanout, high big-block utilization with heavy connectivity, excessive SLR crossing nets

Congestion Analysis: Recommended Approach

WARNING: [Route 35-447] Congestion is preventing the router from routing all nets.

- Check router warning messages
- Run Report Design Analysis (IDE) or report_design_analysis -congestion
 - Focus on **Initial Estimated Router Congestion**
 - Most detailed report: **Global**, **Long**, and **Short** congestion
 - Congestion picture seen by the router after initial routing
 - Best opportunity to incorporate effective changes
 - Placed reports confirm congestion but are not as accurate
 - Router Maximum indicates whether congestion was resolved
- Check **Initial Estimated Router** congestion level
 - Level 4 and below - usually not a problem
 - **Level 5** - some difficulty, can be resolved if isolated region
 - **Level 6** - major difficulty for routing, compile time and Fmax impacted
 - **Level 7** and above - usually impossible



Window	Direction	Type	Congestion Level	Percentage of Tiles
Window 1	North	Global	4	2.5%
Window 2	South	Global	5	1.3%
Window 3	East	Global	1	0.0%
Window 4	West	Global	2	0.5%
Window 5	North	Long	5	4.3%
Window 6	South	Long	6	2.6%
Window 7	East	Long	2	0.2%
Window 8	West	Long	3	0.7%
Window 9	North	Short	3	2.3%

Congestion Analysis for UltraScale and UltraScale+

- Identify hierarchical cells contributing to the congested region

The screenshot shows the 'Initial Estimated Router Congestion' window. The table below represents the data shown in the window:

Window	Direction	Type	Congestion Level	Percentage Tiles	Cell Names	Top Cell 1	Top Cell 2	Top Cell 3	Combined LUTs	LUT6	LUT5	Flop	MUXF
Window 1	North	Global	4	2.526%	sub_top_1_7 (95%)				54%	13%	0%	64%	0%
Window 2	South	Global	5	1.340%	sub_top_1_2 (79%)				50%	10%	0%	77%	0%
Window 3	East	Global	1	0.085%	sub_top_1_2 (78%)	sub_top_1_3 (14%)			70%	11%	1%	92%	0%
Window 4	West	Global	2	0.577%	sub_top_1_0 (55%)	sub_top_1_12 (42%)			94%	5%	0%	67%	0%
Window 5	North	Long	5	4.336%	sub_top_1_7 (51%)	sub_top_1_4 (25%)	sub_top_1_8 (18%)		49%	10%	0%	58%	0%
Window 6	South	Long	6	2.647%	sub_top_1_12 (45%)	sub_top_1_2 (38%)	sub_top_1_11 (8%)		48%	10%	0%	67%	0%
Window 7	East	Long	2	0.257%	sub_top_1_2 (80%)	sub_top_1_3 (15%)			78%	21%	0%	88%	0%
Window 8	West	Long	3	0.789%	sub_top_1_8 (19%)	sub_top_1_4 (18%)	sub_top_1_7 (17%)		0%	0%	0%	71%	0%
Window 9	North	Short	3	2.366%	sub_top_1_3 (86%)				61%	14%	0%	91%	0%

- Check for congestion “symptoms” in those cells
 - report_design_analysis -congestion: check utilization columns
 - report_design_analysis -complexity: use -hierarchical_depth to report cell complexity metrics
 - report_qor_suggestions: evaluate analysis and apply suggestions on problem areas
 - report_utilization [-cells] : review utilization of problematic cells

Congestion Culprits

- Congested regions may be due to netlist structure
 - Heavy MUXF usage can create shortage of connections
 - Long CARRY chains attract dense logic and connections
 - Large numbers of low-fanout control signals (reset, enable) consume routing resources
 - Excessive amount of complex logic cones (high Rent Exponent)
- Resynthesize the hierarchical cells contributing to congestion
 - Caution using strategies and options globally, possible wider effect on timing
 - Use block-level synthesis strategies to generate more suitable netlist
 - Out-of-context (OOC) flow can be used for solutions not yet handled by block-level synthesis

Congestion Solutions (1/2)

- Disable LUT combining on a problem instance

- Override existing LUT properties with empty value at beginning of Implementation

```
set_property HLUTNM "" get_cells -hier -filter {ref_name =~ LUT* && name =~ */inst/*}
```

- High utilization of MUXF and CARRY primitives may cause congestion

- Use opt_design to remap MUXF and CARRY primitives to LUTs
- Global options `-muxf_remap` and `-carry_remap`: use with caution, may affect timing
- Focus on cells in congested regions using cell properties `MUXF_REMAP` and `CARRY_REMAP`

```
set property MUXF_REMAP 1 [get_cells inst_a]; # opt_design only remaps MUXF cells in inst_a
```

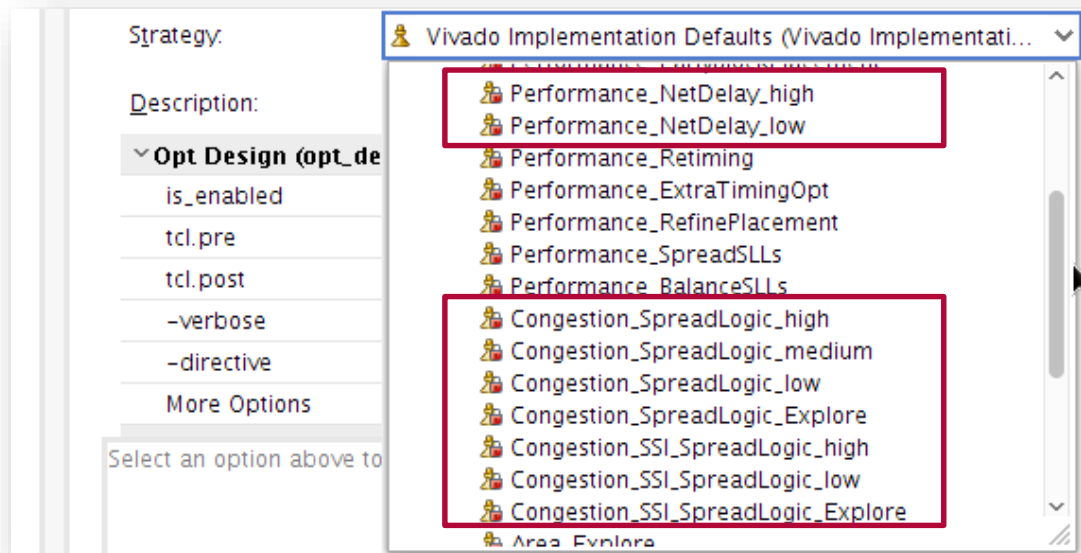
- Further reduce logic levels to minimize timing impact

- `opt_design -remap`: affects entire design
- Can use DONT_TOUCH for surgical optimization

```
set orig_dont_touch_cells [get_cells -hier -filter DONT_TOUCH] ; # remember which cells have DONT_TOUCH
set DONT_TOUCH 1 <all cells except those to be remapped> ; # isolate the cells for optimization
opt_design -remap ; # only optimize the isolated cells
set DONT_TOUCH 0 [get_cells -hier] ; # removed DONT_TOUCH from all cells
set DONT_TOUCH 1 $orig_dont_touch_cells ; # restore original DONT_TOUCHed cells
```


Congestion Solution (2/2)

- Suggested Implementation Strategies



UltraScale+: **NetDelay*** are good choices
Less Short congestion, more Long congestion

UltraScale: **SpreadLogic*** are good choices

See UG904 Appendix C for directives used
in each strategy

report_qor_suggestions (RQS)

New

- Analyzes design QoR issues and recommends solutions
 - Can be run at any design stage
 - Reports design issues against categories of different QoR checks
 - Suggests actions to take for each design issue
 - Provides turnkey XDC and Tcl files containing suggested fixes

- Next phase in progressing toward automated timing closure solutions
 - Current format: a text report with detailed descriptions and supplementary files
 - Future release: interactive GUI report
 - Eventual goal: Design Closure Wizard

Summary



timing_summary
timing
locks (Note: Tcl only)
lock_networks
lock_interaction
dc
methodology
design_analysis
control_sets
high_fanout_nets

Additional Resources

➤ UG 949

➤ UG 903

➤ UG 906

➤ UG 1231

**All available now in the
resource widget!**

➤ [UltraFast Design Methodology for the Vivado Design Suite – Introduction and Overview](#)

➤ [Constraint Explosion Quick Take Video](#)

➤ [UltraFast Vivado Design Methodology for Timing Closure](#)

➤ www.xilinx.com/vivado

Q&A



facebook.com/XilinxInc



twitter.com/#!/XilinxInc



youtube.com/XilinxInc



linkedin.com/company/xilinx



plus.google.com/+Xilinx



xilinx.com/about/app-download.html