Top 5 Timing Closure Techniques

Greg Daughtry

XILINX
ALL PROGRAMMABLE™


CLUB VIVADO
users group
Correct Timing Constraints
Analyze Before Doing
Implementation Strategies and Directives
Congestion and Complexity
Advanced Physical Optimization

Create Good Timing Constraints

- Create constraints: Four key steps
  1. Create clocks
  2. Define clocks interactions
  3. Set input and output delays
  4. Set timing exceptions

  } Baseline Constraints

- Use Timing Constraint Wizard
  - Powerful Constraint Creation Tool

- Validate constraints at each step
  - Monitor unconstrained objects
  - Validate timing
  - Debug constraint issue post-synthesis
    - Analysis will be faster

- ☑ report_timing_summary
- ☑ check_timing
- ☑ report_clocks *(Note: Tcl only)*
- ☑ report_clock_networks
- ☑ report_clock_interaction
- ☑ XDC and TIMING DRCs
- ☑ Report CDC

Establish a Good Starting Point

Baseline with Timing Constraint Wizard

- Disable user XDC file(s)
  - Leave IP XDC files as is
- Create baseline XDC file, set as target
- Run Timing Constraints Wizard
  - Constrain all clocks and clock interactions
  - Flag CDC issues by running Report CDC
- Skip IO constraints in first pass
- Iterate through P&R stages, validate timing at every stage
  - Add exception constraints where necessary
  - Core Flop-to-Flop timing can be met
- Add IO & other exception constraints in subsequent passes
  - Iterate through P&R stages, validate timing at every stage of flow

- Correct Timing Constraints
- Analyze Before Doing
- Implementation Strategies and Directives
- Congestion and Complexity
- Advanced Physical Optimization

World Class Analysis
Make Sense of Your Design Data

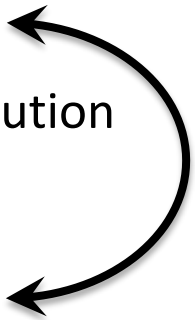
- 45 Reports Give Critical Design Info
  - Clocks and clock interaction
  - Timing Analysis and Constraints
  - Design Complexity
  - Utilization
  - Power
  - Placer/Router/Optimization Status
  - DRC
  - Control Sets
  - IP Upgrade Status

  Vivado% help report_*
- Log files have Context-sensitive Information
  - Every action in order of execution
  - Severity levels: Info, Warning, Critical Warning, and Errors
- Progressive Estimation Accuracy
  - As stages progress from pre-synth to final route “signoff”

Report Design Analysis

Report Types

- **Timing**
  - Key netlist, timing and physical critical path characteristics
  - Combination of characteristics that lead to timing violations
  - Logic levels distribution per destination clock
- **Complexity**
  - Logical netlist complexity
  - Metrics and problematic cell distribution
- **Congestion**
  - Congestion seen by placer, router
  - Top contributors to SLR crossings



Complexity may lead to Congestion

Extended Timing Report

- Setup analysis: show the paths before and after the critical path

report_design_analysis **-extend** -setup

Characteristics	WorstPath to Src	Path #1	WorstPath from Dst
PATH_TYPE	SETUP	SETUP	SETUP
REQUIREMENT	3.100	3.100	3.100
PATH_DELAY	1.066	3.283	1.227
LOGIC_DELAY	0.391(37%)	0.422(13%)	0.303(25%)
NET_DELAY	0.675(63%)	2.861(87%)	0.924(75%)
CLOCK_SKEW	[illegible]	[illegible]	[illegible]
SLACK	1.927	-0.379	1.638
CLOCK_RELATIONSHIP	Safely Timed	Safely Timed	Safely Timed
TIMING_EXCEPTION			
LOGIC_LEVELS	0	3	1
LOGICAL_PATH	RAMB36E2 FDRE	FDRE LUT6 MUXF7 MUXF8 FDRE	FDRE LUT3 FDRE
STARTPOINT_CLOCK	txusrclk_int	txusrclk_int	txusrclk_int
ENDPOINT_CLOCK	txusrclk_int	txusrclk_int	txusrclk_int
STARTPOINT_PIN	RAMB36E2/CLKARDCLK	FDRE/C	FDRE/C
ENDPOINT_PIN	FDRE/D	FDRE/D	FDRE/D
COMB_DSP	0	0	0
DOA_REG	1	0	0
DOB_REG	1	0	0
MREG	0	0	0
PREG	0	0	0
BRAM_CROSSINGS	1	1	4
DSP_CROSSINGS			
IO_CROSSINGS			
CONFIG_CROSSINGS			
SLR_CROSSINGS	0	0	0
BOUNDING_BOX_SIZE	0% x 1%	2% x 13%	17% x 0%
CLOCK_REGION_DISTANCE	(0, 0)	(0, 2)	(1, 0)
RBLOCKS	0	0	0

...

See how much slack is available from surrounding paths

Logic Level Distribution

report_design_analysis

- Number of logic levels in top 5000 critical paths
  - Default number of paths cannot be changed (2015.3 will fix this)
  - Table can be generated for specific paths using -of_timing_paths

3. Logic Level Distribution

End Point Clock	0	1	2	3	4	5	6	7	8	9	10	11-15	16-20	21-25	26-30	31+
async_default	279	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rxusrclk_int	5	0	3	5	0	0	0	0	0	0	0	0	0	0	0	0
txusrclk_int	1059	701	266	1622	186	192	487	195	0	0	0	0	0	0	0	0

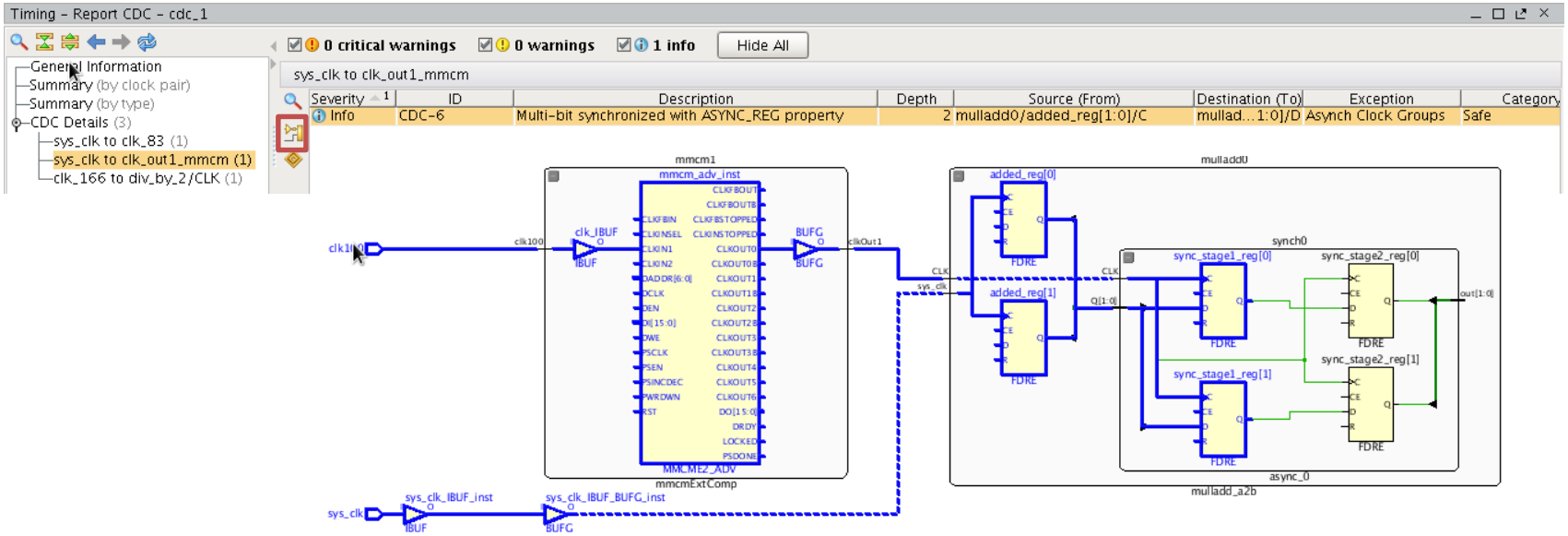
* Columns represents the logic levels per end point clock
** distribution is for top worst 5000 paths

- Identify longest paths (outliers) and modify the RTL
  - Reduces placer focus on few difficult paths only
  - Expands placer solutions and optimization range

Clock Domain Crossing Report

report_cdc

- Identifies CDC topologies
  - Reports unsafe crossings and constraint issues
- Structural issues reported even if exception constraints exist
- Excellent cross-probing support
  - View schematics and exact line number in RTL



CLUB VIVADO users group

- Correct Timing Constraints
- Analyze Before Doing
- Implementation Strategies and Directives
- Congestion and Complexity
- Advanced Physical Optimization

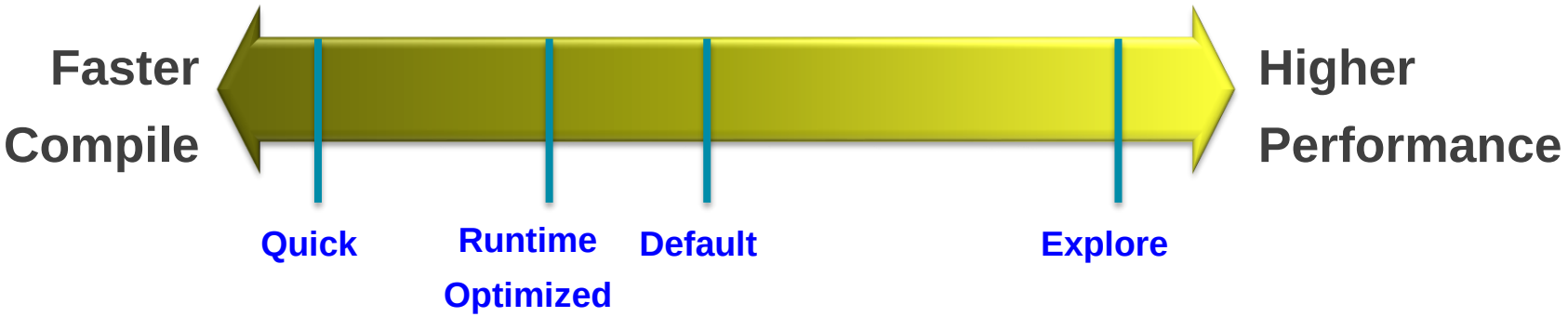
Try All The Tool Options
SmartXplorer Style

- Launch a run for every strategy
  - Easy To Try
  - Pick the best one from design runs table
- Runs Infrastructure Supports “Grid” Computing
  - Built-in parallel runs on different hosts (Linux)
  - LSF and Sun Grid Engine
- Don’t Expect This Will Solve All Your Problems

Vivado Implementation Strategies and Directives

- Directive: "directs" command behavior to try alternative algorithms
  - Enables wider exploration of design solutions
  - Applies to opt_design, place_design, phys_opt_design, route_design

- Strategy: combination of implementation commands with directives
  - **Performance**-centric: all commands use directives for higher performance
  - **Congestion**-centric: all commands use directives that reduce congestion
  - **Flow**-centric: modifies the implementation flow to add steps to Defaults
    - power_opt_design
    - post-route phys_opt_design



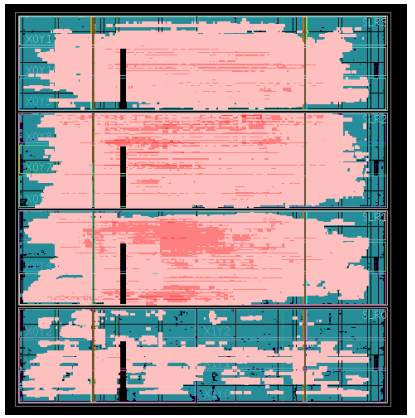
Implementation Strategies

Strategy Name	Objectives
Defaults	Balance between timing closure effort and compile time
Performance_Explore Performance_ExplorePostRoutePhysOpt	Multiple passes of opt_design and phys_opt_design, advanced placement and routing algorithms, and post-route placement optimization. Optionally add post-route phys_opt_design.
Performance_NetDelay_*	Makes delays more pessimistic for long distance and higher fanout nets with the intent to shorten their overall wirelength. Low, medium, and high settings (high = high pessimism).
Performance_WLBlockPlacement	Prioritize wirelength minimization for BRAM/DSPs
Congestion_SpreadLogic_*	Spread logic to aggressively avoid congested regions (low, medium, and high settings control degree of spreading)
Performance_ExploreSLLs	Timing-driven optimization of SLR partitioning
Congestion_BalanceSLLs Congestion_BalanceSLRs Congestion_SpreadLogicSLLs Congestion_CompressSLR	Algorithms for alleviating congestion in SSI designs: Balance SLLs between SLRs, balance utilization in each SLR, spread logic (SSI-tailored algorithms), compress logic in SLRs to reduce SLLs


CLUB VIVADO
users group
Correct Timing Constraints
Analyze Before Doing
Implementation Strategies and Directives
Congestion and Complexity
Advanced Physical Optimization

Congestion

- Physical regions with
  - High pin density
  - High utilization of routing resources



"Smear" Maps

- Placer congestion
  - Congestion-aware: balances congestion vs. wirelength vs. timing slack
    - Cannot always eliminate congestion
    - Cannot anticipate potential congestion introduced by hold fixing
    - Timing estimation does not reflect detours due to congestion
  - Reports congested areas seen by placer algorithms
- Router congestion
  - Routing detours are used to handle congestion at the expense of timing
  - Reports largest square areas with routing utilization close to 100%

Placer congestion tends to be more conservative than router

Complexity Report

- Complex modules in lower hierarchy

report_design_analysis -complexity [-hierarhcial_depth N]

Rent's Rule:

$$N_p = K_p N_g^{\beta}$$

Report Design Analysis

Table of Contents

1. Complexity Characteristics

1. Complexity Characteristics

High Rent (β), Avg fanout on larger instances

High LUT6%, MUXF* utilization

Instance	Module	Rent	Average Fanout	Total Instances	LUT1	...	LUT5	LUT6	Memory LUT	DSP	RAMB	MUXF
(top)	top	0.40	4.54	331191	320(0.2%)		4131(2.2%)	172519(90.8%)	960	0	16	104555
genblk1[0].x_inst	xge_mac	0.57	5.90	5335	38(1.1%)		455(12.8%)	1669(47.0%)	120	0	2	9
rx_eq0	rx_enqueue_340	0.70	4.02	1776	0(0.0%)		132(10.2%)	735(56.8%)	0	0	0	0
tx_dq0	tx_dequeue_346	0.67	3.78	2305	32(1.9%)		276(16.3%)	825(48.7%)	0	0	0	0
genblk1[0].xbar_inst	xbar_top	0.72	4.41	35625	2(0.0%)		55(0.3%)	19666(98.8%)	0	0	0	13056
decode_inst	xbar_dec_335	0.73	4.67	33940	1(0.0%)		28(0.1%)	19338(99.1%)	0	0	0	12864
xbar_inst	xbar_core_336	0.37	1.59	1685	1(0.3%)		27(6.8%)	328(82.4%)	0	0	0	192
genblk1[1].x_inst	xge_mac_2	0.57	4.23	5321	38(1.1%)		454(12.8%)	1661(46.9%)	120	0	2	9
rx_eq0	rx_enqueue_290	0.66	4.02	1776	0(0.0%)		132(10.2%)	735(56.8%)	0	0	0	0
tx_dq0	tx_dequeue_296	0.65	3.78	2305	32(1.9%)	...	276(16.3%)	825(48.7%)	0	0	0	0
genblk1[1].xbar_inst	xbar_top_3	0.72	4.41	35625	2(0.0%)		55(0.3%)	19666(98.8%)	0	0	0	13056
decode_inst	xbar_dec_285	0.74	4.67	33940	1(0.0%)		28(0.1%)	19338(99.1%)	0	0	0	12864
xbar_inst	xbar_core_286	0.34	1.59	1685	1(0.3%)		27(6.8%)	328(82.4%)	0	0	0	192
genblk1[2].x_inst	xge_mac_6	0.57	3.94	5319	38(1.1%)		454(12.8%)	1661(47.0%)	120	0	2	9
rx_eq0	rx_enqueue_240	0.66	4.02	1776	0(0.0%)		132(10.2%)	735(56.8%)	0	0	0	0
tx_dq0	tx_dequeue_246	0.62	3.78	2305	32(1.9%)		276(16.3%)	825(48.7%)	0	0	0	0
genblk1[2].xbar_inst	xbar_top_7	0.72	4.41	35625	2(0.0%)		55(0.3%)	19666(98.8%)	0	0	0	13056
decode_inst	xbar_dec_235	0.74	4.67	33940	1(0.0%)		28(0.1%)	19338(99.1%)	0	0	0	12864
xbar_inst	xbar_core_236	0.34	1.59	1685	1(0.3%)		27(6.8%)	328(82.4%)	0	0	0	192

Congestion Report Example

report_design_analysis -congestion

- Placer congestion section

Window defined in CLB tiles

Top contributors to the region

1. Placed Maximum Level Congestion Reporting

Direction	Window Size	Congestion in Window (%)	Congestion Window	Modules Names	LUT usage %
North	16x16	103	(CLE_M_X10Y134,CLEL_R_X17Y149)	xbar_inst(50%),decode_inst(50%),	80
			(CLEL_R_X17Y72,CLE_M_X21Y79)	xbar_inst(50%),decode_inst(50%),	73
South	16x16	95	(CLE_M_X10Y60,CLEL_R_X17Y75)	xbar_inst(50%),decode_inst(50%),	79
West	8x8	97	(CLEL_R_X8Y70,CLE_M_X12Y77)	xbar_inst(50%),decode_inst(50%),	86

Largest congested region

find cells using:

get_cells -hier <Name>

- Note: In 2015.3 -congestion must be run in same session as place_design and route_design

Placer Congestion Report Example

- Placed tile-based section (smear metrics tables)

3. Placed Tile Based Congestion Metric (Vertical)

Tile Name	RPMX	RPMY	Congestion in Window (%)	Modules Names
CLBLM_L_X34Y110	88	93	75	usbEngine1(50%),fftEngine(50%),
CLBLM_L_X34Y85	88	119	72	usbEngine0(100%),
CLBLM_L_X34Y86	88	118	72	wbArbEngine(25%),usbEngine0(25%),fftEngine(25%),
CLBLM_L_X34Y160	88	41	72	mgtEngine(50%),fftEngine(50%),
CLBLM_L_X34Y139	88	63	71	usbEngine1(100%),
CLBLM_L_X34Y128	88	74	70	usbEngine1(100%),
CLBLM_L_X34Y87	88	117	70	usbEngine0(50%),fftEngine(50%),
CLBLM_L_X34Y90	88	114	70	usbEngine0(100%),
CLBLM_L_X34Y150	88	51	69	mgtEngine(50%),fftEngine(50%),
CLBLM_L_X34Y89	88	115	69	usbEngine0(50%),fftEngine(50%),

Top contributors to the region

4. Placed Tile Based Congestion Metric (Horizontal)

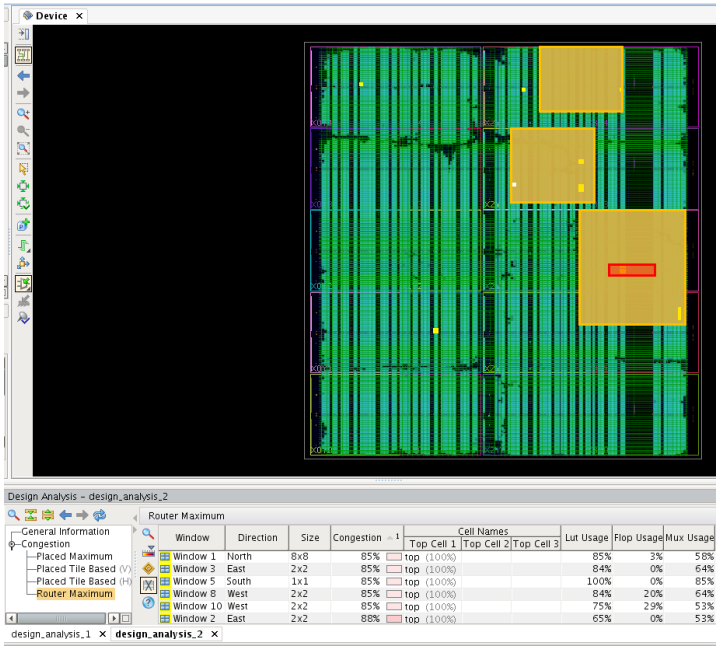
Tile Name	RPMX	RPMY	Congestion in Window (%)	Modules Names
CLBLM_R_X15Y155	43	46	73	usbEngine1(50%),fftEngine(50%),
CLBLL_L_X18Y35	50	171	73	cpuEngine(100%),
CLBLM_L_X10Y135	30	67	72	usbEngine1(50%),fftEngine(50%),
CLBLL_R_X21Y35	57	171	70	cpuEngine(100%),
CLBLL_R_X19Y35	53	171	70	cpuEngine(100%),
CLBLL_R_X21Y36	57	170	69	
CLBLL_L_X26Y36	69	170	69	
CLBLL_L_X22Y35	58	171	68	
CLBLL_L_X20Y35	54	171	67	cpuEngine(100%),
CLBLL_L_X20Y36	54	170	67	cpuEngine(100%),

find using: get_cells -hier <Name>

Routing Congestion

report_design_analysis -congestion

- Graphical View
- Text Report



Actual routing resource utilization

Size of region

Window dimensions

Direction	Size	Congestion	Congestion Window	Cell Names	Lut Usage
North	8x8	85%	(CLEL_L_X64Y100,CLEL_L_X64Y107)	top(100%)	85%
East	2x2	88%	(CLEL_R_X48Y214,CLE_M_X49Y215)	top(100%)	65%
East	2x2	85%	(CLEL_R_X48Y196,CLE_M_X49Y197)	top(100%)	84%
East	2x2	89%	(CLEL_R_X48Y194,CLE_M_X49Y195)	top(100%)	90%
South	1x1	85%	(CLE_M_X8Y271,CLE_M_X8Y271)	top(100%)	100%
South	1x1	93%	(CLE_M_X38Y268,CLE_M_X38Y268)	top(100%)	75%
South	1x1	92%	(CLEL_R_X56Y268,CLEL_R_X56Y268)	top(100%)	50%
West	2x2	85%	(CLEL_R_X56Y138,CLEL_L_X57Y139)	top(100%)	84%
West	2x2	91%	(CLEL_R_X56Y134,CLEL_L_X57Y135)	top(100%)	78%
West	2x2	85%	(CLEL_R_X22Y90,CLE_M_X23Y91)	top(100%)	75%

* Congested regions with less than 85% congestion are not reported.

XILINX ALL PROGRAMMABLE

CLUB VIVADO users group

Potential Solutions for Congestion

- Reduce Logic or Pick a Bigger Device
  - Look for wide bus and mux structures
- Optimize modules in congested regions
  - Disable LUT combining design-wide or in congested instances
    - Globally with *synth_design -no_lc*
    - set_property SOFT_HLUTNM “” [get_cells -hier -filter {name =~ instance/*}]
  - Consider OOC synthesis with different options, strategies
  - Turn off cross-boundary optimizations in synthesis
    - Globally with *synth_design -flatten_hierarchy none*
    - On specific modules with KEEP_HIERARCHY in RTL
- Try several implementation strategies or placer directives
  - Try congestion-oriented placer strategies and directives first
  - Try other strategies and placer directives

  => Re-use some or all RAMB and DSP placement from good runs
- Try floorplanning the congested logic
  - Prevent complex modules from overlapping
  - Consider dataflow through device

- Correct Timing Constraints
- Analyze Before Doing
- Implementation Strategies and Directives
- Congestion and Complexity
- Advanced Physical Optimization

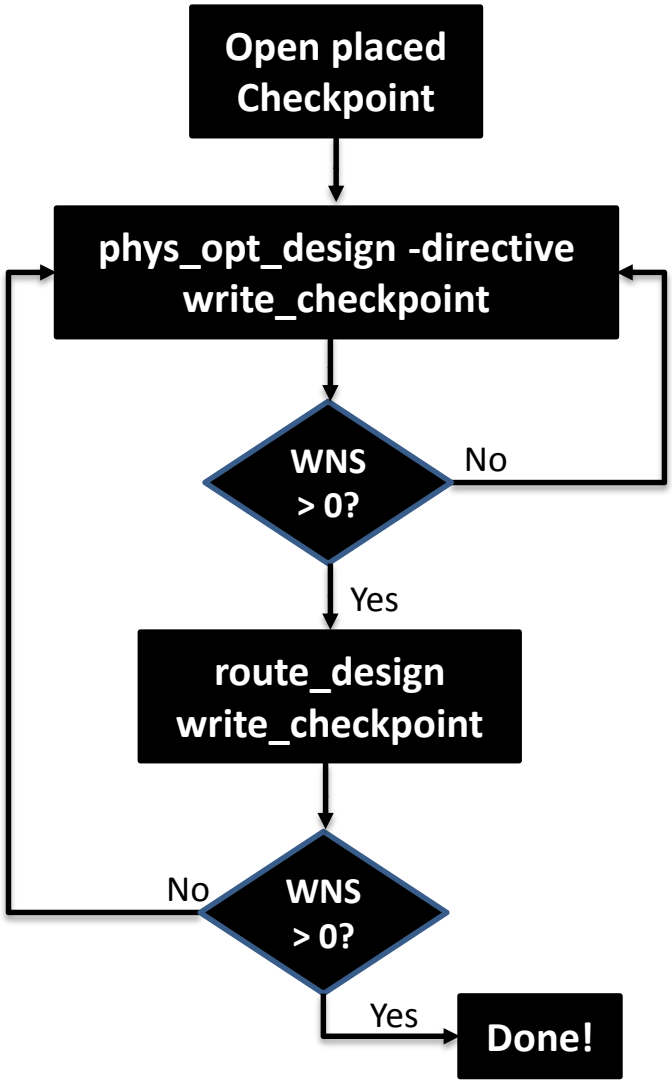
Post-Place Physical Optimization

Can Make a Big Difference

- Many useful Tricks are implemented
  - Replication (based on fanout, timing or specified nets)
  - BRAM/DSP/SRL register optimization
  - Retiming
  - Moving cells to better location after each optimization
- Not part of the default strategies
  - You need to choose the tradeoff in extra runtime
- Designed to be “Re-entrant”
  - This means you can run it multiple times in a script

Post-Place Physical Optimization Looping

- Primary goal: improve WNS as much as possible
  - WNS limits max frequency
- Secondary goal: improve TNS as much as possible
  - TNS increases stress on router algorithms, which can impact WNS & WHS
- Run phys_opt_design until timing is met (or close), or until WNS and TNS do not improve
- Insert into run flow as a hook script



Using Post-Place Physical Optimization

- DO NOT RUN post-place physical optimization if
  - Worst paths can only be fixed by changing the RTL
  - Haven't tried several placer directives first
  - The design has not been properly baselined first
  - There are CRITICAL WARNINGs that have not been dealt with

- RUN post-place physical optimization if
  - Timing constraints are known to be good
  - Worst timing violations are related to
    - High fanout nets
    - Nets with loads placed far apart
    - High RAMB/DSP/SRL delay impact
  - WNS and TNS are "reasonable" (WNS > -1ns, TNS > -10,000ns)
    - **Try several placer directives to identify the best placement startpoint**

Over-Constraining with Clock Uncertainty

- Recommended technique to over-constrain a design
  - XDC command: **`set_clock_uncertainty`**
  - Fine granularity: clock pair
  - Setup and Hold separately constrained
  - Easy to reset: **`set_clock_uncertainty 0 <clockOptions>`**
  - Does not affect clock relationships
    - Modified clock periods can make CDC paths overly tight or asynchronous
- Where and when to add/remove user clock uncertainty
  - Add before place_design or phys_opt_design (Hook Script)
    - Increases optimization range to provide better timing budget for router
    - Reduces impact of delay estimates variation or congestion
  - Remove before route_design in most cases
    - Over fixing hold is bad

Review Physical Optimization Timing QoR

- WNS and/or TNS improve after each phys_opt_design
- Example (below) with partial over-constraining

Directive	WNS	TNS	Failing Endpoints
Best Placement Result	-0.247	-289.95	3498
Add 200ps user clock uncertainty			
Popt1 (AggressiveExplore)	-0.329	-866	7829
Remove 200ps user clock uncertainty			
Popt2 (AggressiveExplore)	-0.060	-1.971	182
Popt3 (AggressiveFanoutOpt)	-0.029	-0.243	31
Routed	0.003	0.000	0

Analyze the Physical Optimizations Log

- Reviewing detailed information
  - Type of optimization, object name
  - Intermediate timing numbers
  - Optimizations prevented by DONT_TOUCH
- Applying some of the changes to RTL
  - RAMB/DSP register optimization
  - Some register replication on RAMB/DSP or IO paths
- Using scripting to identify the optimizations with more impact
  - Example: **`grep -P '(Optimized|Estimated)' vivado.log`**

```
vivado.log:INFO: [Physopt 32-619] Estimated Timing Summary | WNS=-0.367 | TNS=-1139.370 |
vivado.log-INFO: [Physopt 32-29] End Pass 1. Optimized 33 nets. Created 119 new instances.
vivado.log:INFO: [Physopt 32-619] Estimated Timing Summary | WNS=-0.367 | TNS=-1071.577 |
vivado.log-INFO: [Physopt 32-661] Optimized 98 nets.  Re-placed 98 instances.
vivado.log:INFO: [Physopt 32-619] Estimated Timing Summary | WNS=-0.343 | TNS=-1055.180 |
vivado.log-INFO: [Physopt 32-608] Optimized 33 nets.  Swapped 36 pins.
vivado.log:INFO: [Physopt 32-619] Estimated Timing Summary | WNS=-0.329 | TNS=-865.770 |
```

Post-Route Physical Optimization Expectations

- When should I run post-route phys_opt_design?

  => For fixing small violations only

  - WNS > -0.2ns
  - TNS > -10ns

- How many times should I run post-route phys_opt_design?

  => ONLY ONE TIME!!

  - Very high runtime

Router and Timing Closure

- Cost Function
  - Timing, Congestion and Architecture device model rules
    - Timing first but congestion impacts timing
    - Architecture rules also impact timing
- Targets critical paths first
  - Number of Logic levels impacts router algorithms
  - Lower level logic paths may fail timing after route_design
- Addresses TNS and WNS
  - WNS first priority, TNS second

Summary

- Timing closure – A difficult problem
  - Start with good constraints
  - Analyze and Understand issues
  - Investigate RTL changes to improve timing first
- Vivado has powerful analysis utilities:
  - Basic: report_timing, check_timing, report_exceptions, report_clock_utilization …
  - Advanced: report_design_analysis, report_cdc, Baselining,
  - Methodology: UltraFast Design Methodology …
- Powerful optimization techniques
  - Phys opt looping, post-route phys opt, over constraining, floor-planning etc.