



Timing Closure Tips and Tricks

Presented By

Ron Plyler

Product Marketing, Vivado Implementation Tools

December 10, 2018



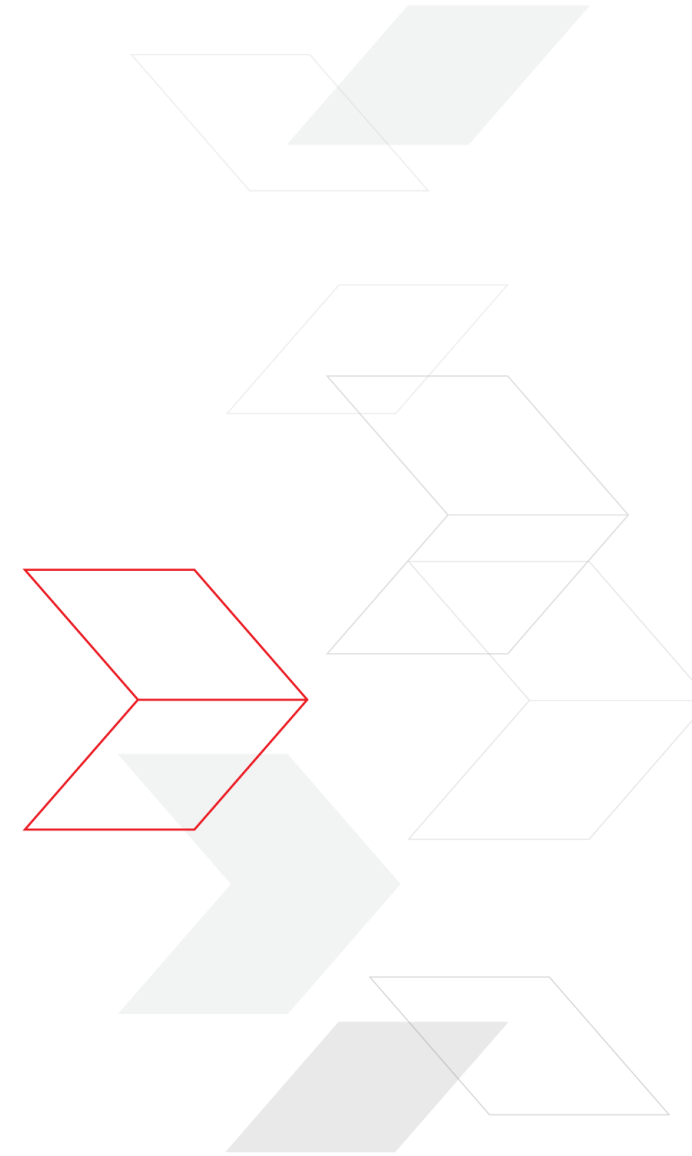
Topics for Today

- > **Implementation Flow: Built for Timing Closure**
- > **SSI Design: Tips for Maximizing Performance**
- > **Timing Closure: Automating Solutions**




Implementation Flow

Built for Timing Closure



Start New Projects with Latest Vivado Versions

2012-2013	2013-2014	2015-2016	2017-2018+
opt_design	opt_design +HFN global buffering	opt_design +Hierarchy-based replication driver merging	opt_design
power_opt_design	power_opt_design	power_opt_design	power_opt_design
place_design	place_design	place_design	place_design +PSIP (phys_opt_design replication, MAX_FANOUT support) +HFN global buffering
	+phys_opt_design +replication, retiming, and re-placement	phys_opt_design	phys_opt_design +SLR crossing optimization
route_design	route_design	route_design	route_design +PSIR (phys_opt_design replication and re-routing)
Bold indicates <i>required</i> flow step		+phys_opt_design (post-route) +critical path replication and re-routing	phys_opt_design +SLR crossing optimization

More and more Timing Closure features built into Implementation 

Core Placement and Routing Improvements

> Replication provides better default QoR

- >> PSIP (Physical Synthesis In Placer) enabled by default starting 2018.2
- >> New MAX_FANOUT recommendations
 - Synthesis: use MAX_FANOUT only on local, low-fanout replication, not design-wide signals
 - PSIP: use MAX_FANOUT to suggest replication candidate nets during placement phase
- >> PSIR (Physical Synthesis In Router) introduced 2018.1 for UltraScale+

> Router directives for higher design performance

- >> New directives built on top of **Explore**
- >> More runtime tradeoffs

-directive AggressiveExplore (2018.3)

-directive NoTimingRelaxation

-directive Explore

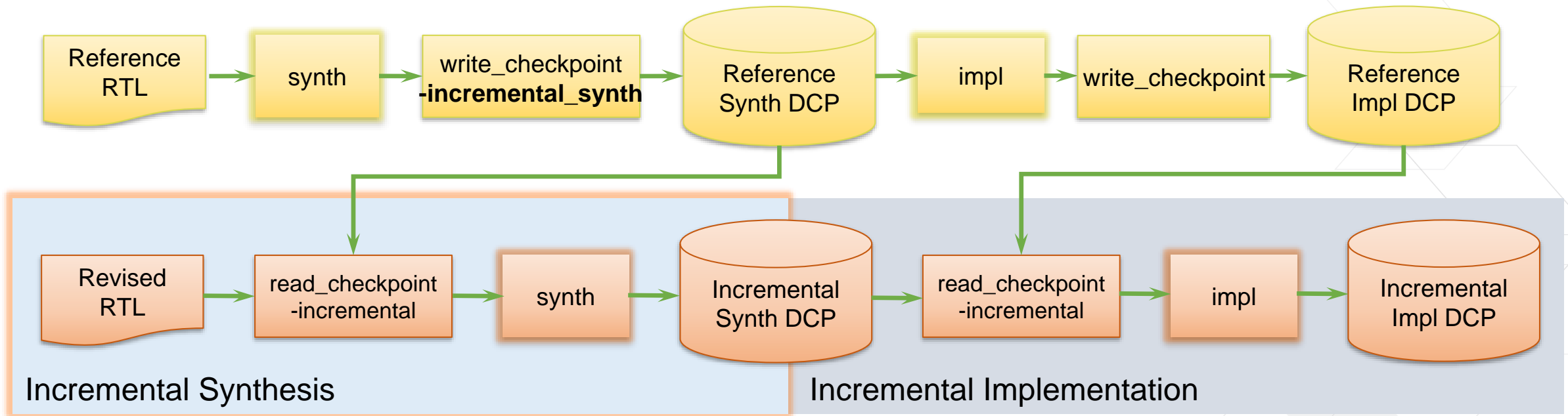
Post-route critical path optimization

US+ clock skew optimization

+Maintain original timing targets (don't relax)

+More exhaustive thresholds and effort levels

The New Incremental Compile Flow



> **Incremental Synthesis and Implementation bolt together to reduce compile time and preserve timing-closed results**

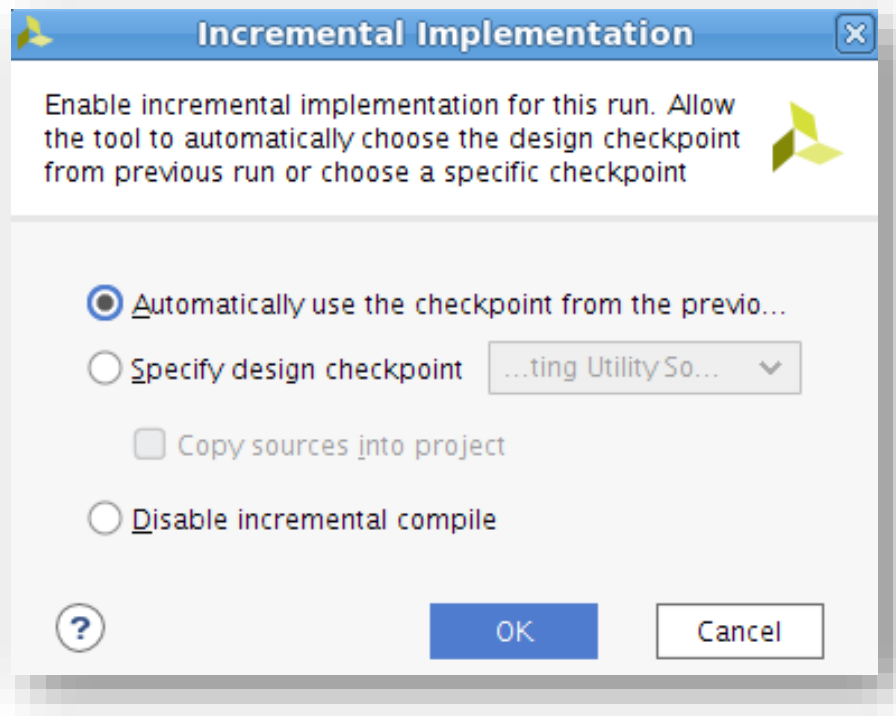
> **Incremental Synthesis minimizes netlist changes**

>> Requires write_checkpoint *-incremental_synth* option to save incr data

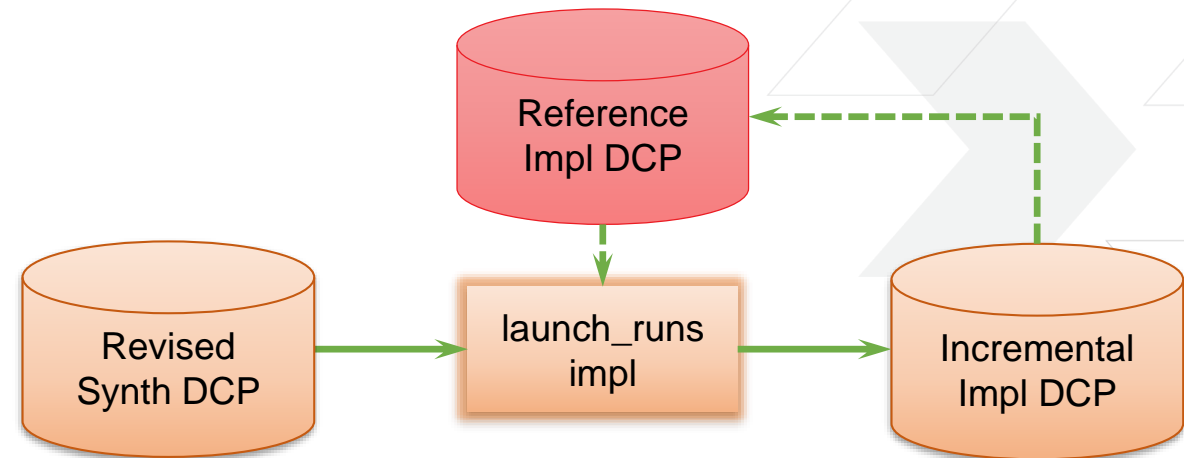
Automatic Incremental Implementation for Projects

> Pushbutton mode where Vivado manages Incremental DCP for each run

New in 2018.3

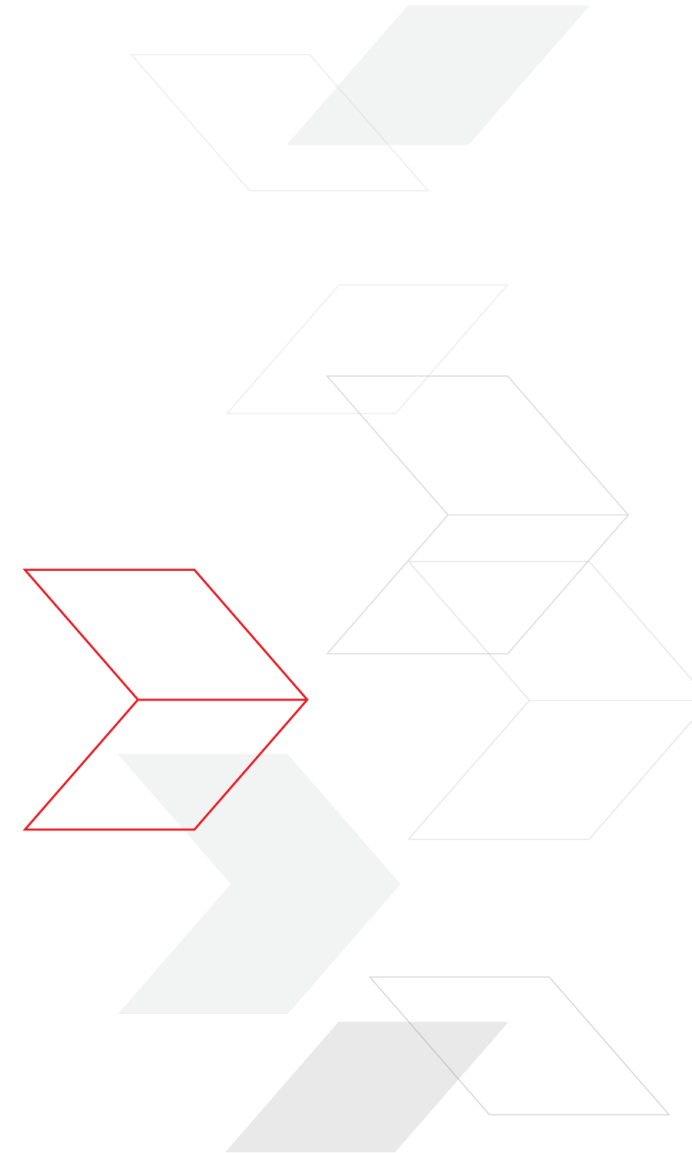


- >> Recirculates latest routed DCP as the reference DCP
 - if it is a good fit
- >> Reference DCP added to sources in `utility_1` fileset
 - Captured in project archives



SSI Design

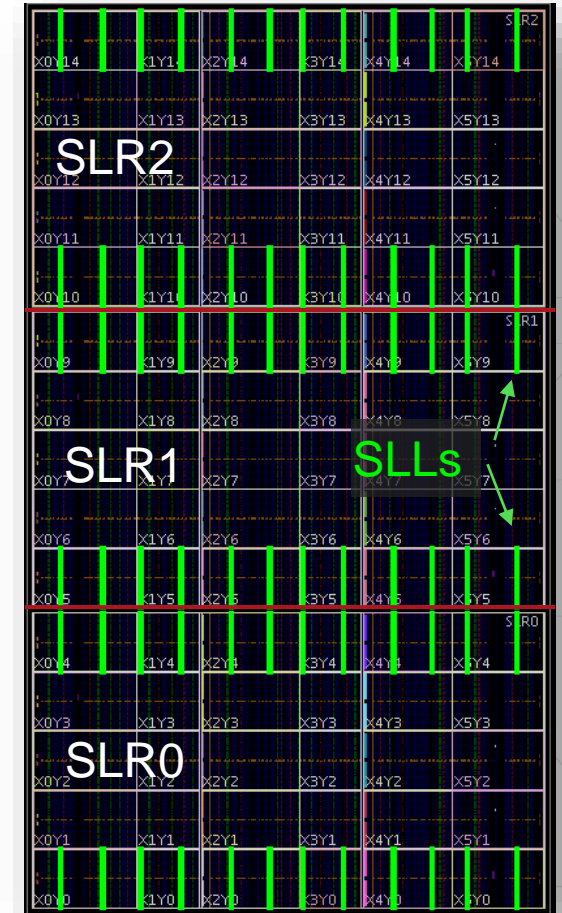
Tips for Maximizing Performance



Proliferating SSI as the Platform of Choice

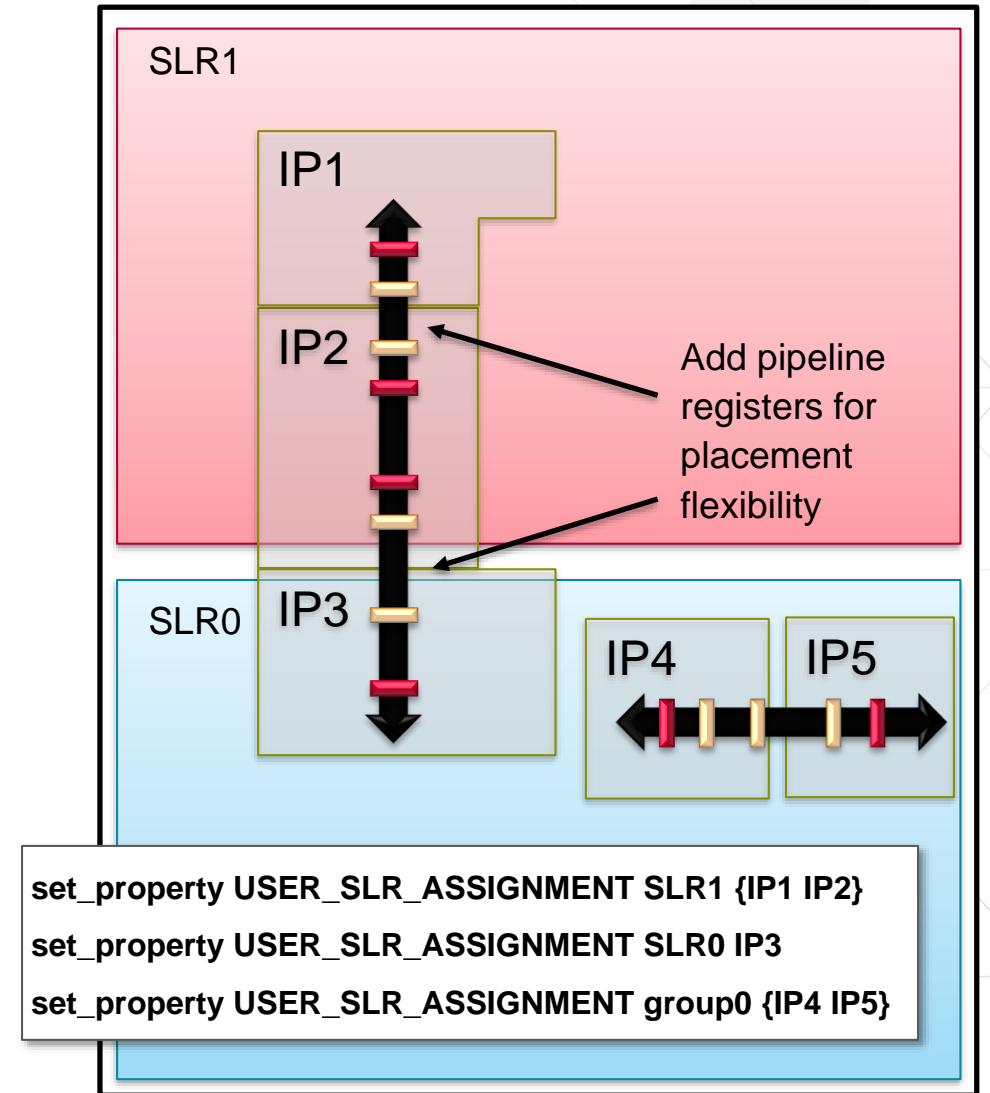
- > **Vivado placement and routing are continuously improving in basic key areas**
 - >> Delay Estimation - more accurate pre-route modeling of SLR crossings
 - >> Congestion - better spreading near SLR crossings
 - >> SLR Crossing Speed - more opportunistic use of SLL registers
- > **New features improve quality of Partitioning and Placement**
 - >> USER_SLR_ASSIGNMENT: Control partitioning of cells
 - >> USER_CROSSING_SLR (EA): Control partitioning based on nets/pins
 - >> Laguna TX_REG -> RX_REG direct connection (UltraScale+ only)
 - >> USER_SLL_REG (EA): SLL (Laguna) register preference to improve speed, predictability

VU9P: 3 SLRs



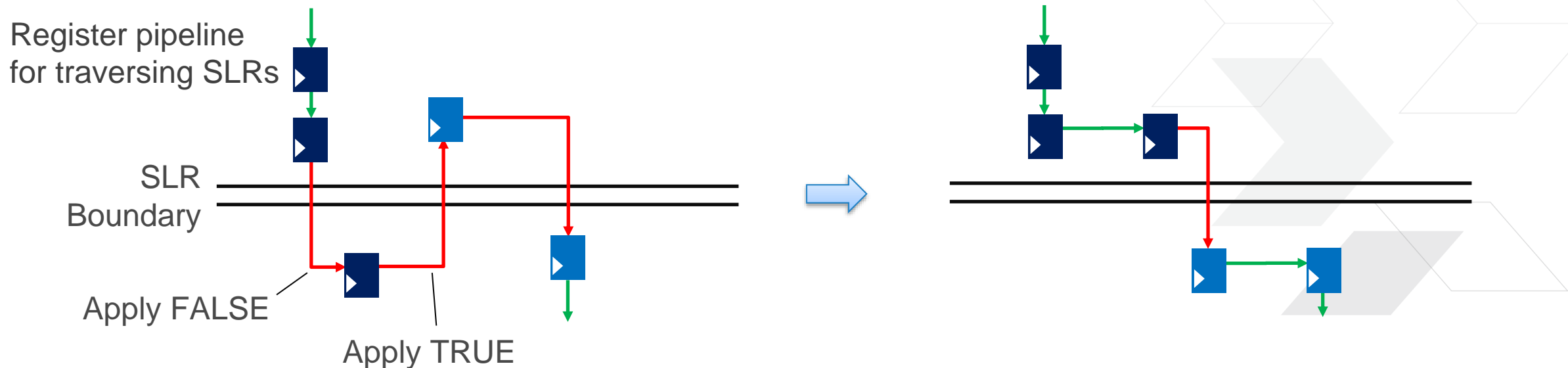
Partitioning with USER_SLR_ASSIGNMENT

- > **Hierarchical cell property (*not for leaf cells*)**
 - >> Assigns cells to SLR when SLR name is used: SLR0, SLR1, SLR2, ...
 - >> Keeps cells in same SLR when value is a string
 - >> Tries to prevent cells from crossing SLR boundaries
- > **More flexible than Pblocks**
 - >> Soft constraint, ignored if prevents successful partition
 - >> Placer, PhysOpt not restricted by Pblock bounds
- > **Add pipeline registers on cell boundaries**
 - >> Helps maintain clock speed
 - >> Allows even greater placement flexibility



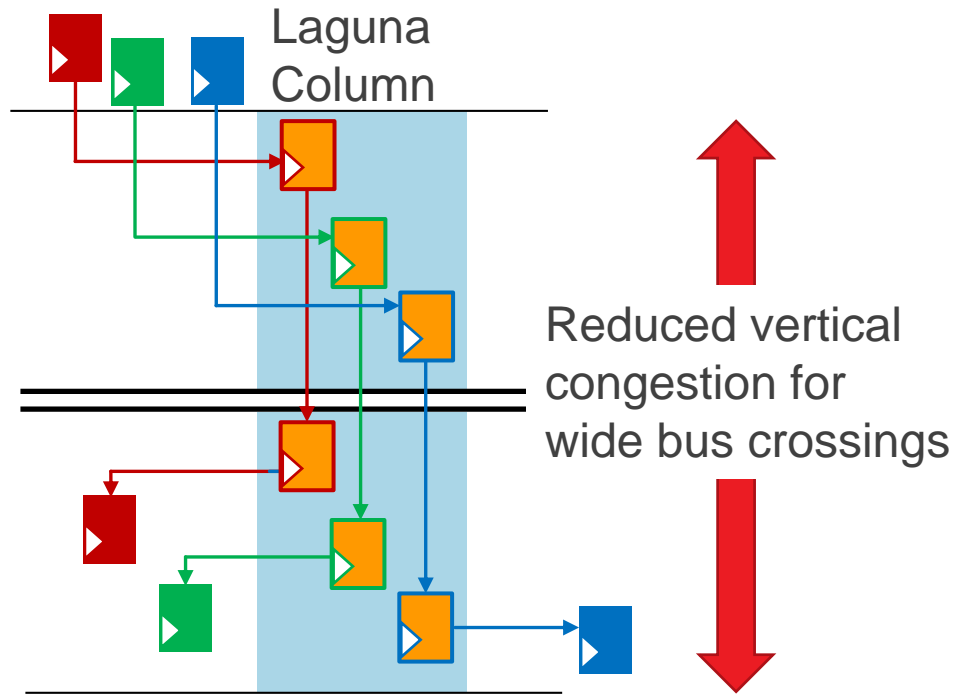
USER_CROSSING_SLR (Early Access Until 2018.3)

- > **Soft constraint: pin/net property for fine-tuning SLR partitioning**
- > **Specifies a preference that connections should cross an SLR boundary**
 - >> True applies only to single-fanout pipeline register connections
 - >> False applies to any net or input pin except internal library macros: `PRIMITIVE_LEVEL == INTERNAL` (restriction removed in 2018.3)



Using UltraScale+ SLL Registers

Register pipelines
for traversing SLRs



Consistent crossing performance

- > **TX_REG can drive RX_REG directly (2018.1)**
 - >> Router adjusts leaf clock skews to fix hold
 - >> Fits most intra-clock topologies
 - >> **Not for use with Clock Domain Crossings**
- > **Use USER_SLL_REG register property (2018.3)**
 - >> Easier method to move register from fabric to nearby Laguna register
 - >> Similar behavior as IOB property
- > **Use BEL and LOC to constrain and lock SLR crossing interfaces**

Timing Closure

Automating Solutions



report_qor_suggestions (RQS)

- > Reduce timing closure time and effort (Introduced in 2017.1)
- > Run report and follow suggestions
- > Example: RQS analysis generated suggestions to:
 - >> Improve congestion
 - >> Improve critical paths ending at control pins



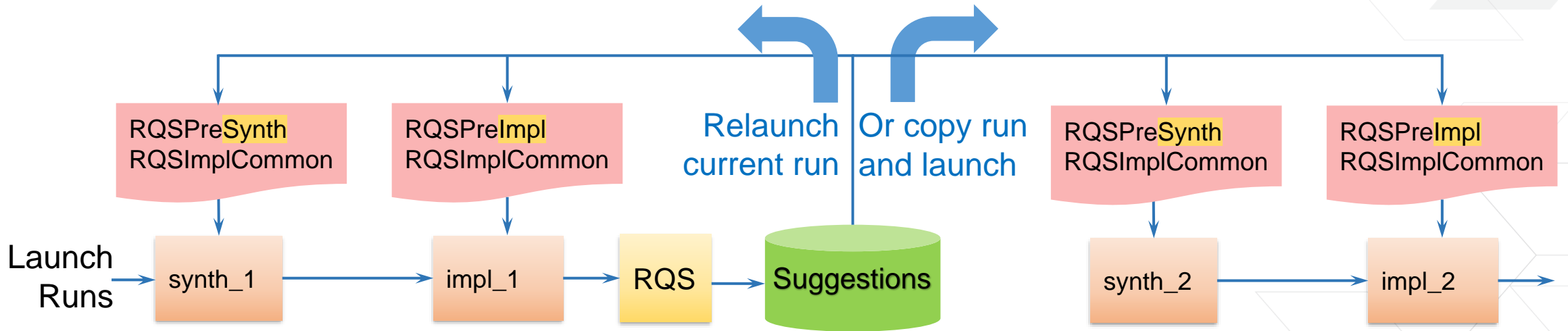
1. QoR Suggestions Report Summary

Category	Suggestion Type
Congestion	Congestion due to macro primitives/high fanout nets
Timing	Critical Control Signals Remap

RQSPreSynth.xdc output file

```
### Following section is for unrolling SRLs into flops from the congested regions
###
### Following SRLs are unrolled into flops to remove congestion.
###
set_property BLOCK_SYNTH.SHREG_MIN_SIZE 17 [get_cells { inst_0}]
set_property BLOCK_SYNTH.SHREG_MIN_SIZE 3 [get_cells { inst_1 inst_2}]
set_property BLOCK_SYNTH.SHREG_MIN_SIZE 33 [get_cells { inst_3 inst_4}]
###
### End of section for unrolling SRLs into flops for congested regions
### Following section is for Critical Paths Ending at Control Pins Suggestion
###
### For following flops control pin logic is moved to data path.
###
set_property EXTRACT_RESET NO [get_cells { inst_6 inst_7 inst_8 inst_9/inst_reg[*]}]
###
### End of section for Critical Paths Ending at Control Pins Suggestion
```

Applying Suggestions



- > Design sources frozen?
 - Start with Implementation
- > Design sources in development?
 - Start with Synthesis
- > Add RQS XDC and Tcl.pre files

> RQS Automation Roadmap

- >> 2018.3: Interactively create & launch runs
- >> 2019.X: Integrate Incremental Compile
- >> 2019.X: Dynamically update suggestions throughout the flow

Introducing report_qor_assessment (RQA) in 2018.3

- > How likely will design goals be met? Evaluates an entire design and generates a simple score

```
Report QoR Assessment

Table of Contents
-----
1. QoR Assessment summary
2. UltraFast Design Methodology checks summary

1. QoR Assessment summary
-----

+-----+-----+
| QoR Assessment Score |           2 - Unlikely to meet timing constraints |
+-----+-----+
| Recommendation       | Run report_qor_suggestions and review Next Steps |
+-----+-----+

2. UltraFast Design Methodology checks summary
-----
No report_methodology results found!

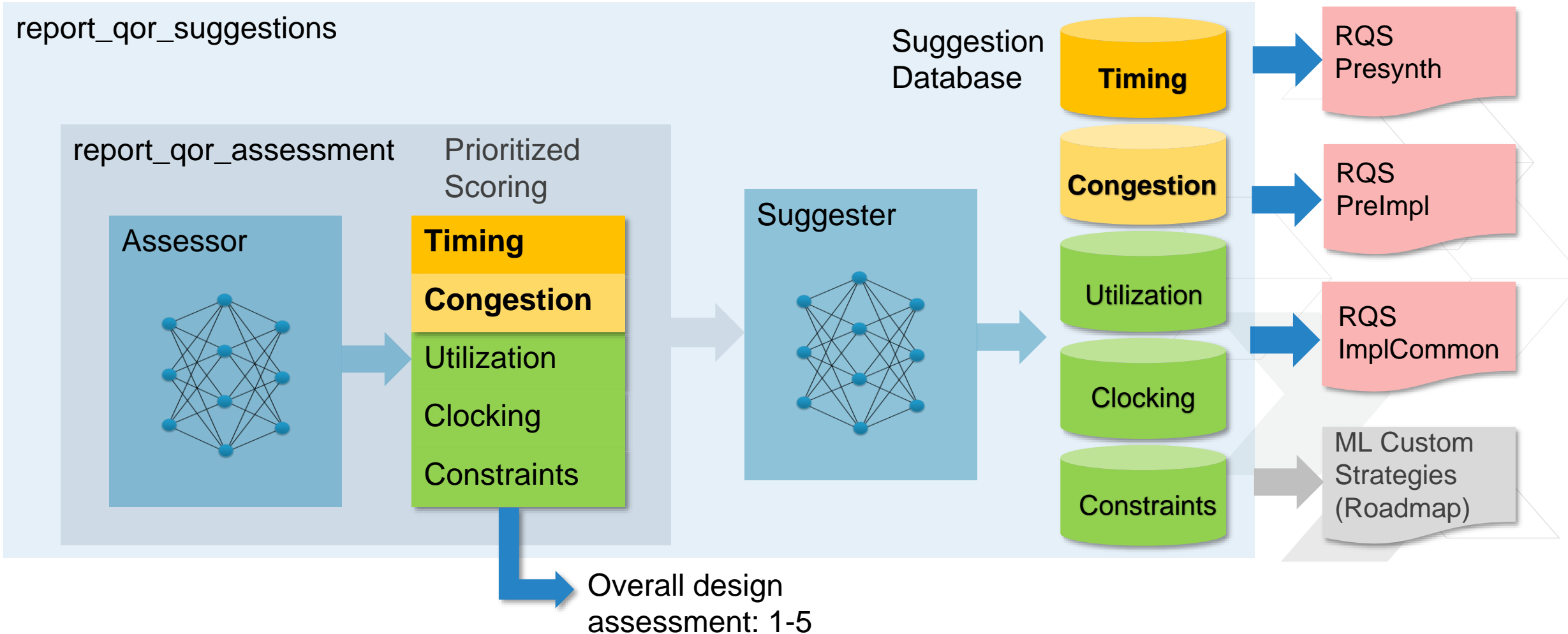
Run report_methodology and review design and constraint checks to ensure
properly functioning hardware.
```

Assessment Scores

- 1 Implementation will fail, stop flow
- 2 Timing will fail, review RQS
- 3 Timing difficult, add many strategies
- 4 Timing fair, add a few strategies
- 5 Timing easy to meet

Assessment and RQS Suggestion Integration

Assessment is used to generate RQS suggestions



Summary

- > **Begin new projects with the latest Vivado version**
 - >> 2018.3 planned for mid-December
- > **Use Incremental Compile to reduce compile times and preserve timing closure**
- > **Apply new SSI constraints to improve UltraScale and UltraScale+ performance**
- > **Benefit from automated analysis and solutions: report_qor_assessment (2018.3) and report_qor_suggestions (Now)**
- > **Please share feedback on problems and improvements**



XILINX
DEVELOPER
FORUM

