Timing Closure Tips and Tricks

Presented By

Ron Plyler
Product Marketing, Vivado Implementation Tools
October 2, 2018
Topics for Today

- Implementation Flow: Timing Closure Enhancements
- SSI Design: Tips for Maximizing Performance
- Timing Closure: Automating Solutions
Implementation Flow

Timing Closure

Enhancements
## Implementation: Evolving for Timing Closure

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

**Increasing Effectiveness**
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 (planned 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
Tips on Timing Closure and Power Optimization

> Optimize for power up front, don’t wait until after timing closure
  > Include power_opt_design in the flow
  > Enable BRAM power opt in opt_design (disabled for Explore)
  > Maximize usage and depths of cascaded BRAMs and URAMs
  > Use set_power_opt to enable and disable power_opt at the cell level

> Adding PowerOpt too late changes the netlist which may affect timing

> Set a power budget for reporting
  set_operating_conditions -design_power_budget 45

> Consistently monitor total power for large swings
  > Alerts you to design changes, tool options, and strategies with negative power impact
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
Incremental Implementation Enhancements

> New `read_checkpoint -incremental` options to reuse portions of a design
  >> Streamlined, consistent, and scalable
  >> Only two options: `-reuse_objects <args>` and `-fix_objects <args>`
  >> Arguments are objects: `<cells>` `<clock regions>` `<SLRs>` `[current_design]`

> Automatic Incremental Implementation for Projects (EA in 2018.3)
  >> Pushbutton mode where Vivado manages Incremental DCP for each run

- Auto mode can coexist with manual mode
- Multiple runs supported
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
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 pipeline for traversing SLRs

Laguna Column

Reduced vertical congestion for wide bus crossings

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 BEL and LOC to constrain and lock SLR crossing interfaces

- Use USER_SLL_REG register property (EA until 2018.3)
  - Easier method to move register from fabric to nearby Laguna register
  - Similar behavior as IOB property
SLR Crossing Optimization Throughout Implementation

> **Placer**
  >> Improved automatic placement and spreading, new property-driven mapping
  >> TX_REG to RX_REG direct connection (UltraScale+ only)

> **Post-Place PhysOpt**
  >> -slr_crossing_opt now supported, considers small positive slack paths too
  >> Add -tns_cleanup option to focus on SLR crossings more aggressively

> **Router**
  >> Adjusts skew using programmable clock leafs to fix TX_REG -> RX_REG hold

> **Post-Route PhysOpt**
  >> slr_crossing_opt + -tns_cleanup to focus on SLR crossings more aggressively

Note: phys_opt_design -slr_crossing_opt and -tns_cleanup are optional, not default
## Feature Summary and Availability

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual Laguna TX-&gt; RX</td>
<td>No</td>
<td>Yes</td>
<td><img src="image" alt="Not yet supported" /></td>
<td><img src="image" alt="Production" /></td>
<td><img src="image" alt="Production" /></td>
</tr>
<tr>
<td>Auto Laguna TX-&gt; RX</td>
<td>No</td>
<td>Yes</td>
<td><img src="image" alt="Early Access" /></td>
<td><img src="image" alt="Production" /></td>
<td><img src="image" alt="Production" /></td>
</tr>
<tr>
<td>USER_SLR_ASSIGNMENT</td>
<td>Yes</td>
<td>Yes</td>
<td><img src="image" alt="Production" /></td>
<td><img src="image" alt="Production" /></td>
<td><img src="image" alt="Production" /></td>
</tr>
<tr>
<td>USER_SLL_REG</td>
<td>Yes (no TX-&gt;RX)</td>
<td>Yes</td>
<td><img src="image" alt="Early Access" /></td>
<td><img src="image" alt="Production" /></td>
<td><img src="image" alt="Production" /></td>
</tr>
<tr>
<td>USER_CROSSING_SLR</td>
<td>Yes</td>
<td>Yes</td>
<td><img src="image" alt="Early Access" /></td>
<td><img src="image" alt="Production" /></td>
<td><img src="image" alt="Production" /></td>
</tr>
<tr>
<td>PhysOpt Laguna TX-&gt; RX</td>
<td>No</td>
<td>Yes</td>
<td><img src="image" alt="Not yet supported" /></td>
<td><img src="image" alt="Production" /></td>
<td><img src="image" alt="Production" /></td>
</tr>
</tbody>
</table>

**Notes**

- USER_CROSSING_SLR only applicable to pins in 2018.1, nets planned for 2018.2
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

<table>
<thead>
<tr>
<th>Category</th>
<th>Suggestion Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Congestion</td>
<td>Congestion due to macro primitives/high fanout nets</td>
</tr>
<tr>
<td>Timing</td>
<td>Critical Control Signals Remap</td>
</tr>
</tbody>
</table>

RQSPreSynth.xdc output file

```bash
### Following section is for unrolling SRLs into flip-flops from the congested regions
### Following SRLs are unrolled into flip-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 flip-flops for congested regions
### Following section is for Critical Paths Ending at Control Pins Suggestion
### For following flip-flops control pin logic is moved to data path.
set_property EXTRACT_RESET № [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
  Note: XDC being phased out for Tcl

> 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)

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

> Planned release: 2018.3

<table>
<thead>
<tr>
<th>Report QoR Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
</tr>
<tr>
<td>1. QoR Assessment summary</td>
</tr>
<tr>
<td>2. UltraFast Design Methodology checks summary</td>
</tr>
<tr>
<td>1. QoR Assessment summary</td>
</tr>
<tr>
<td>QoR Assessment Score</td>
</tr>
<tr>
<td>Recommendation</td>
</tr>
<tr>
<td>2. UltraFast Design Methodology checks summary</td>
</tr>
<tr>
<td>No report_methodology results found!</td>
</tr>
<tr>
<td>Run report_methodology and review design and constraint checks to ensure properly functioning hardware.</td>
</tr>
<tr>
<td>Assessment Scores</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
Assessment and RQS Suggestion Integration

Assessment is used to generate RQS suggestions

Overall design assessment: 1-5
Summary

- Use 2018.2 for the best UltraScale+ QoR

- 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