Elevating Productivity with Intelligent Design Runs

WP535 (v1.0) July 28, 2021

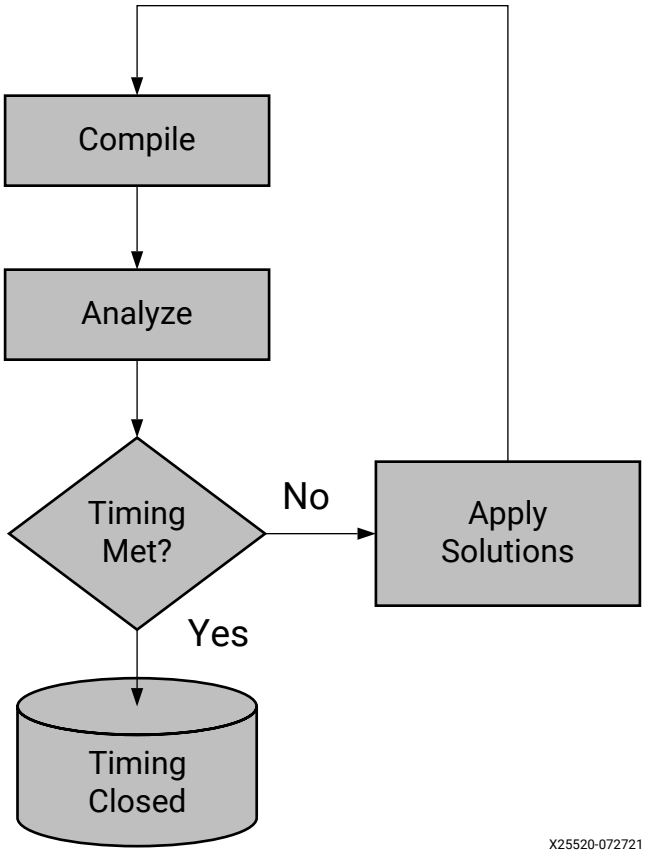
Abstract

Timing closure of high-speed designs has traditionally been one of the more challenging phases of hardware design. The shortest path to a timing-closed design is a laborious journey through complex analysis of timing failures and iteration over countless solutions. Since the launch of the Vivado® Design Suite in 2012, Xilinx has focused on improving the productivity of our customers by building intelligence into timing closure tasks to minimize intervention, freeing up more time to work on product differentiation. Intelligent design runs (IDR) is the latest advancement in productivity that encapsulates all the timing closure building blocks into a powerful high-effort compilation, with push-button access for ease of use.

Introduction

Timing closure is a laborious process of reviewing design tool reports, finding timing failures, evaluating possible root causes of each violation, then formulating solutions to potentially fix the timing failures. The design implementation is rerun with these solutions to resolve the timing issues. This process is shown in the following figure.

Figure 1: **Simplified Diagram of the Timing Closure Process**



Typical solutions cover a wide range of actions:

- HDL source code changes to reduce logic depth and logic utilization
- Insertion of pipeline stages to increase clock speed
- Exploring tool options and compilation strategies using different synthesis, placement, and routing algorithms
- Physically constraining and floorplanning timing-critical logic to help guide design tools

Resolving all timing violations in a design usually involves multiple clock domains and clock domain crossings (CDCs). Therefore, it is important to focus first on the largest perceived bottlenecks and choose among many possible solutions expected to have the greatest positive impact. Typically, multiple solutions are run in parallel as there is not always a clear favorite, and a single iteration beginning with synthesis or implementation can take hours to compile. Running multiple solutions in parallel also reduces risk of losing time from pursuing ineffective solutions,

however, that places a strain on computing resources and requires more effort to manage parallel design runs. A single iteration often fails to close timing so the best solution or solutions of the parallel runs seeds the next round of analysis, solutions, and parallel runs, repeating until no timing failures remain. Thus, timing closure can be a complex, iterative process that can span days, weeks, and possibly months depending on the design size, complexity, and clock frequency requirements.

Taking Productivity to New Levels

To determine how best to improve productivity of timing closure, expert techniques are studied in detail as more successful experience accumulates over years of support escalations. Patterns emerge, revealing that certain solutions could be matched to problem types. For example:

- **Problem:** A path fails timing with too many logic levels for its required delay.
- **Solution:** An XDC constraint to re-time the specific path is added, and the design is recompiled with the new constraint to improve the delay and meet its timing requirement.

While the solution seems obvious and simple to automate, the decision-making is the more difficult part:

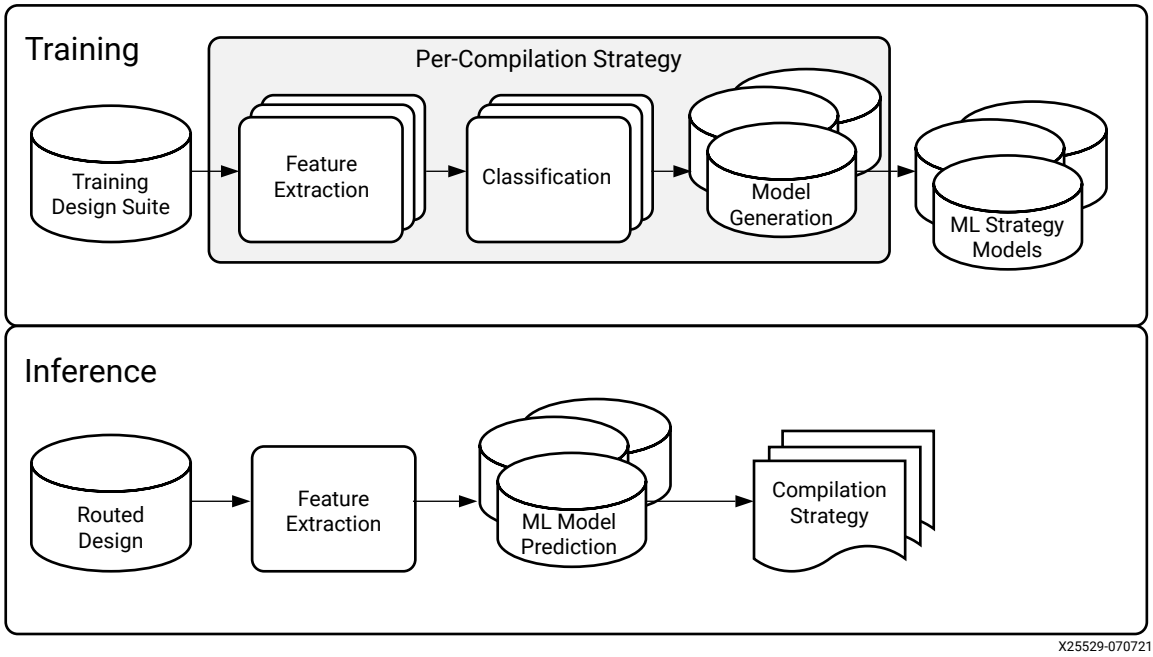
- How many logic levels need to be reduced to meet timing?
- Is re-timing the best solution to apply for this failure or is there a better solution?
- What is the likelihood of success?
- Are there other higher-priority design problems to tackle first?

Timing closure experts grapple with these questions for each issue to be resolved and tend to follow certain rules when making decisions. Many rules can apply for seemingly straightforward problems such as failing critical paths with many logic levels. Considerations include the target architecture, speed grade, utilization, fanout along the critical path, and the path resources and placement just to mention a few. This gives rise to a rule-based system, now known as report QoR suggestions (RQS), providing the intelligence required for making the best decisions. The rules are developed and tuned based on test results covering thousands of unique designs with unique timing failures across all device architectures.

Rule-based systems are sufficient for solving problems when working with concrete quantities such as path delays and slack. Other problems become too complicated for rule-based systems, for example, determining which placement algorithms will work the best for a particular design. It is not feasible to use rules to connect design data to the compatibility of certain algorithms.

Enter machine learning (ML). ML use has rocketed in recent years to solve many complex real-world problems and is recently making its way into EDA tools. ML models solve the problem of predicting which algorithms or more generally which compilation strategies work best for a given design. The process consists of two phases, training and inference, as shown in the following figure.

Figure 2: **Using ML to Predict Top Compilation Strategies**

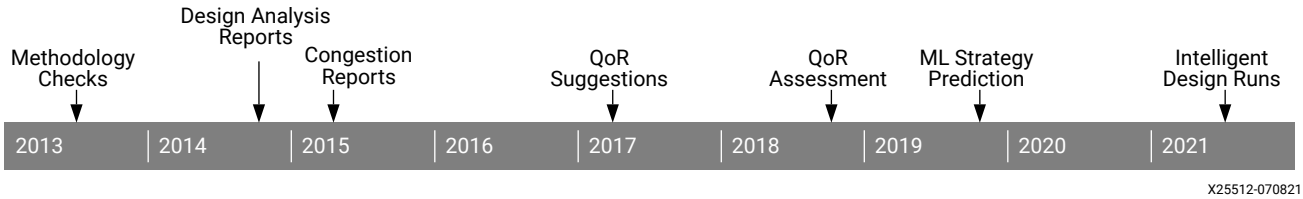


Training involves building the ML models using an extensive design suite covering many device architectures and design styles. Training is a lengthy process with each design targeting all possible compilation strategies, in total approaching a million Vivado Design Suite runs. Design metrics or features are extracted from each run and used to build ML models for each compilation strategy. As more designs are run and features extracted, the models are reinforced with additional data and learn to recognize a favorable design for that strategy based on its features.

Inference is the process of running the prediction on a design. Design features are extracted and used by each strategy's ML model to determine the probability that the strategy will meet the design's timing requirements. Then the top few strategies are used to compile the design in Vivado tools to get better performance.

Many features for timing closure productivity built upon rule-based engines and ML models have been introduced into Vivado tools. A timeline is shown in the following figure.

Figure 3: **Timing Closure Features Introduced in Vivado Tools**



Since its launch in 2012, Vivado tools have progressively injected more intelligence into the timing closure process, with each new feature combining existing building blocks into more powerful features. Following is a timeline review of the release of major features.

- **2013:** Methodology checks are the first to be automated in Vivado tools. Common best practices from the UltraFast Design Methodology User Guide are converted into rules that fit the Vivado tools design rule check (DRC) infrastructure. Design methodology compliance becomes built-in to the tools, pointing out methodology violations that are often responsible for timing violations.
- **2014:** Design analysis and congestion reports provide important insight into timing failure causes, insight based on analysis techniques from timing closure experts. These reports go beyond basic timing reports to include the impact of logical structures, physical implementation, and design complexity on critical paths. This is data that experts would normally extract from designs using schematics, placed and routed design views, and cross-probable timing reports, to decide how to fix timing failures. Metrics like the Rent Exponent are introduced to identify logical instances most likely to generate routing congestion. The UltraFast Methodology documentation begins to document the rules and procedures for resolving timing failures using these reports.
- **2017:** As timing closure techniques begin to mature, a complex rule-based engine is created to provide turnkey solutions, automated as tool suggestions. A suggestion is a new object representing a design constraint or Tcl command to address specific issues uncovered by analysis. RQS becomes a major advancement in rule-based timing closure.
- **2018:** Using design metrics collected throughout synthesis and implementation, a rule-based engine called report QoR assessment (RQA) is developed to assess and predict the chances of successful timing closure, issuing a simple numeric score on a scale from 1 to 5 in increasing likelihood. The report includes details of the problems driving the score and RQS becomes a complementary tool for improving assessments.
- **2019:** Machine Learning models in Vivado tools make the leap from research to practice, predicting which compilation strategies out of dozens of possibilities are most likely to find the best design performance. This reduces the amount of parallel processing needed for timing closure iterations by limiting the number of compilation strategies to try. The predictions are packaged as an RQS option.

The most recent offering (2021) is IDR, which combines both rule-based and ML-based features into a high-effort timing closure flow. It has built-in intelligence to make complex decisions that navigate the timing closure process making the same decisions as a hands-on expert. The internal complexity of the flow is hidden under a simple right-click menu item.

The Beginnings of Intelligent Design Runs

With RQA, RQS, and ML strategy prediction built into the Vivado Design Suite, timing closure experts now have a rich set of features in the timing closure toolbox. By observing how timing closure experts combine and apply the different features to solve problems, a vision arises of a *super-flow* that mimics their techniques. The flow is designed to use the timing closure building blocks in the most effective way.

The first aspect involves the flow engines.

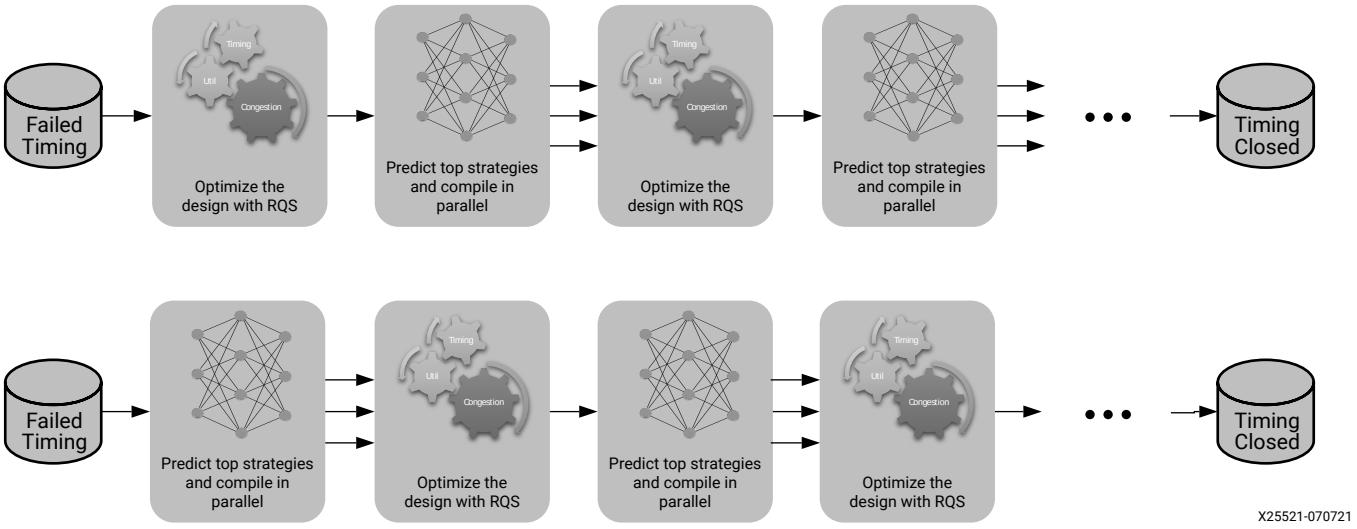
- RQS is used to generate suggestions to solve timing failures due to specific design and constraint problems.

- ML strategies drive parallel exploration of different tool options and compilation strategies to potentially find a better solution than when using default tool settings.

How should these two components interact? A basic approach is to simply serialize the two and iterate until timing is closed, choosing one of the following two approaches:

- Optimize the design first with RQS, run the top ML strategies in parallel, take the best result and repeat until timing is closed, as shown in the upper-half of the following figure.
- Run the top ML strategies in parallel first, then take the best result and optimize it with RQS, and repeat until timing is closed, as shown in the lower-half of following figure.

Figure 4: **Two Approaches to RQS and ML Strategy Combination**



Generally, the first approach is better i.e., begin with RQS design optimization then explore multiple compilation strategies. There are two main reasons for this:

- RQS can improve almost every design that fails timing. Stabilizing QoR yields a better starting point for compilation strategy exploration.
- ML-based strategy prediction can become more accurate when minimizing the tool settings during training runs, and to simplify further, training is limited to the top two commonly used compilation strategies, default and the popular high-effort compilation strategy, performance_explore. The compilation preceding the prediction will be one of these two strategies.

The second aspect of the flow is how the flow is executed, how experts would drive the steps based on timing closure metrics gathered at various stages, and how design data is efficiently managed throughout. To support this, the Vivado tools runs infrastructure that is upgraded with rule-based intelligence to execute RQS and the ML strategies, with decisions based on design metrics like RQA score and WNS, and new file organization to manage all design data and reports.

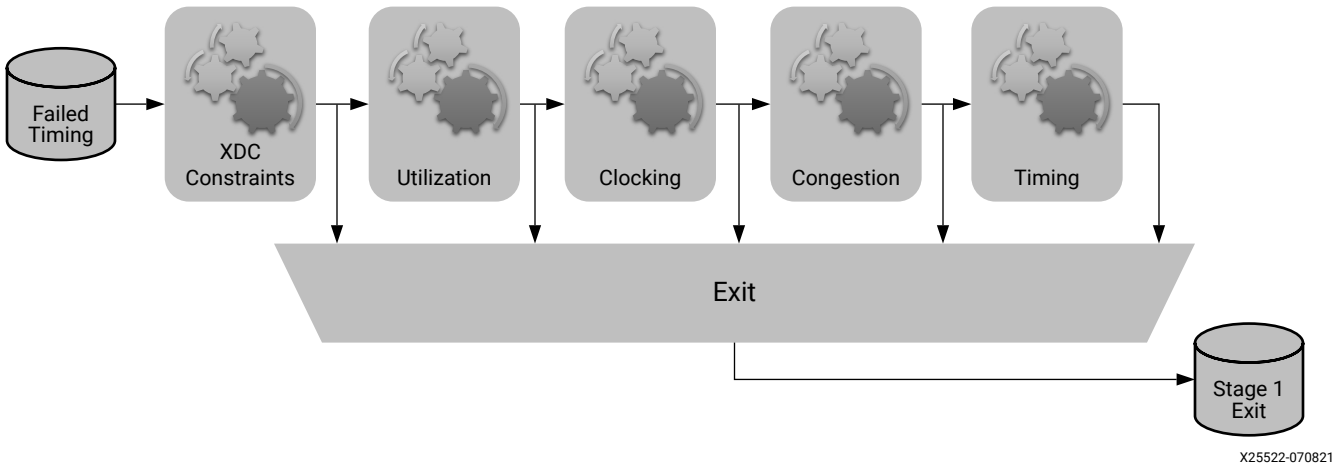
To maximize the reach of the new flow, all of the complexity is hidden under a simple push-button style interface so that most Vivado Design Suite customers can benefit. A simple dashboard displays progress, and expert users can access detailed reports for deep-dive analysis if needed.

For fine-tuning the RQS engine, the key guiding principle is that certain categories of failures should be prioritized over others because they have much higher impact and affect a larger number of timing endpoints. For example, high global clock skew due to an unbalanced clock network affects all timing paths related to those clocks and can cause artificial timing failures that hide the real design challenges. After the high clock skew is addressed, the more meaningful design problems will surface. Fortunately, RQS was designed with this principle in mind with the ability to filter and apply any subset of suggestions such as a set of clocking improvements. Lower priority suggestions with limited reach, such as improving individual failing paths are deferred to later iterations.

To iterate most efficiently, the new flow uses RQS to generate suggestions as early as practical. High clock skew can be detected after placement without needing to run through a full routing phase. If needed, the flow can be stopped, reset, and re-run with the generated suggestions.

The result of extensive research and experimentation is the recipe that becomes the first stage of IDR as shown in the following figure.

Figure 5: **First Stage of Intelligent Design Runs**



Stage 1 of IDR begins with implementation runs of the design with failed timing. Each implementation run launches RQS with focus on one problem category at a time. The categories are tackled from most to least impact. Unlike typical Vivado tool implementation runs, IDR monitors the suggestion types and timing status at each stage and chooses to exit or continue to the next category, hence the name intelligent design runs. Following is a brief description of each step of Stage 1.

- **XDC Constraints:** Check for impossible-to-meet constraints, often the result of timing constraints on asynchronous clock domain crossings (CDCs). IDR will exit early as there is no automatic resolution of incorrectly specified constraints.
- **Utilization:** Consider optimizations that reduce utilization but do not degrade timing on critical paths. Suggestions can be generated early because the analysis is mostly netlist based. Reduced utilization often improves compile time and sometimes improves design performance.

- **Clocking:** Check for issues such as sub-optimal clocking topology and unbalanced global clock trees. If suggestions exist after placement, rerun with the suggestions all the way through routing.
- **Congestion:** Analyze and apply suggestions to reduce congestion.
- **Timing:** Apply suggestions to fix individual failing paths.

The problem categories of Stage 1 are the same as the ones used for QoR assessments. RQA is run automatically during Stage 1, with the simple score at each step increasing if timing closure is improving.

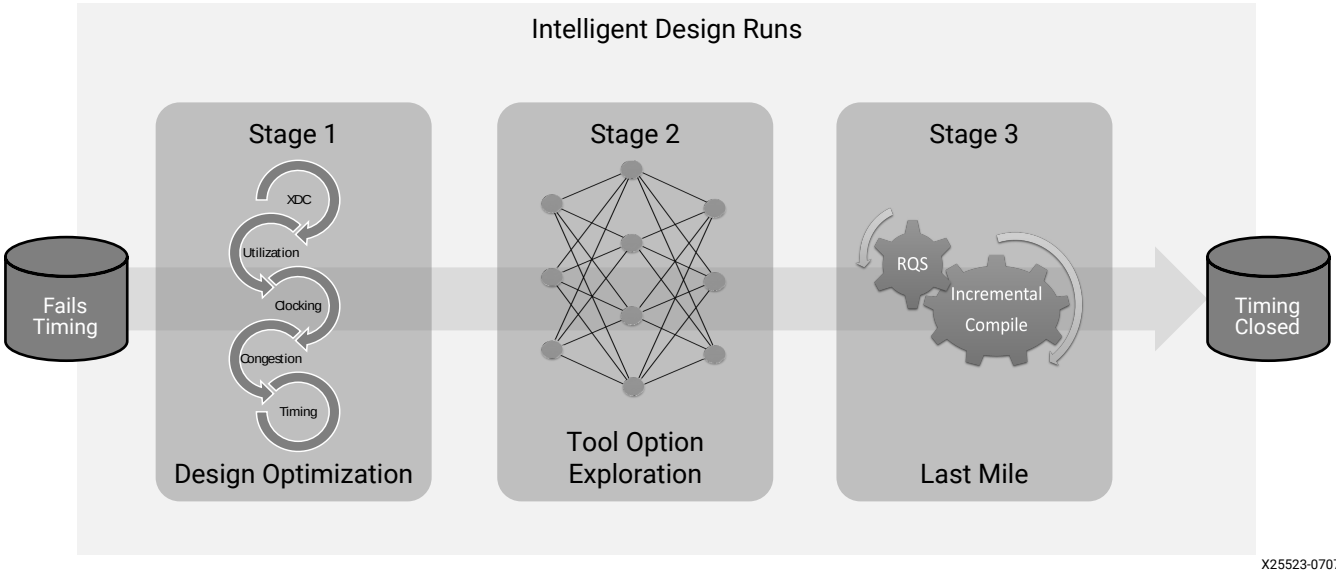
Intelligent Design Runs: Going the Last Mile

With Stage 1 in place, making the best use of RQS to produce an optimized result, Stage 2 becomes a tool option exploration stage where ML models predict the top compilation strategies and tool options. Occasionally, Stage 1 execution closes timing, at which point IDR can exit. If timing failures are present after Stage 1, the result is run through the top predicted compilation strategies. At the end of Stage 2, IDR exits if timing is met. If timing failures remain, IDR evaluates the effort required to close timing and decides if it is worthwhile to continue to Stage 3 or just quit as further optimization is unlikely to help.

The third and final stage of IDR operates with the assumption that wide-scale solutions have been exhausted and concentrates on improving timing on the top critical paths. Stage 1 progressively attacks timing problems from high to low impact, and Stage 2 explores the place and route algorithm solution space. The only practical actions remaining are *surgical strikes* on the top remaining critical paths. Stage 3 optimizations try to disturb the placement and routing as little as possible while working on critical paths and use the Incremental Compile feature to help achieve this. Stage 3 is known as the last mile stage, working diligently to extract the few remaining bits of timing slack.

The three stages of IDR are shown in the following figure.

Figure 6: **Conceptual Diagram of Intelligent Design Runs**

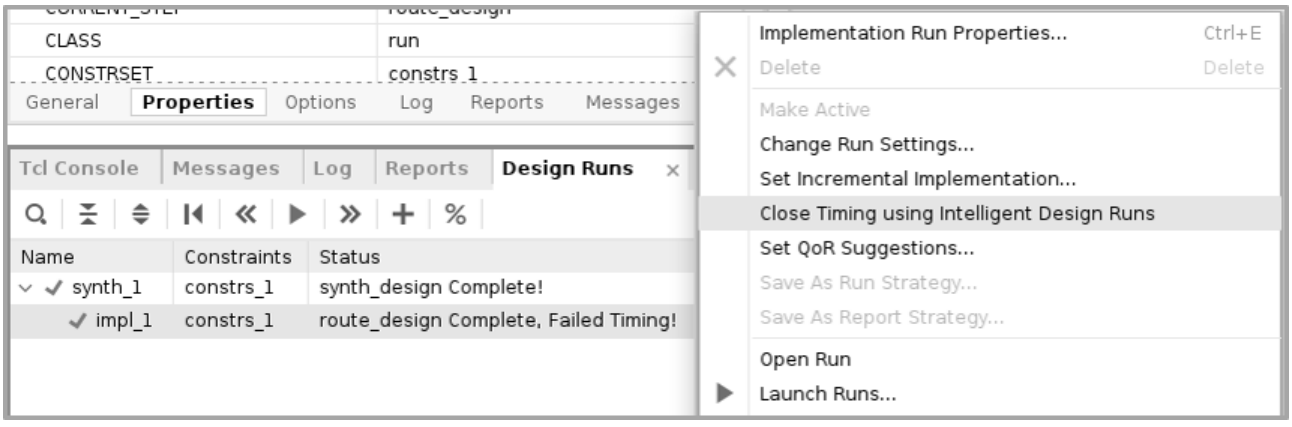


Intelligent Design Runs in Action

Beginning with Vivado Design Suite 2021.1, IDR can be launched with simple steps as shown in the following figure.

1. Select a Design Run with Timing Failures.
2. Right-click and choose **Close Timing using Intelligent Design Runs**.

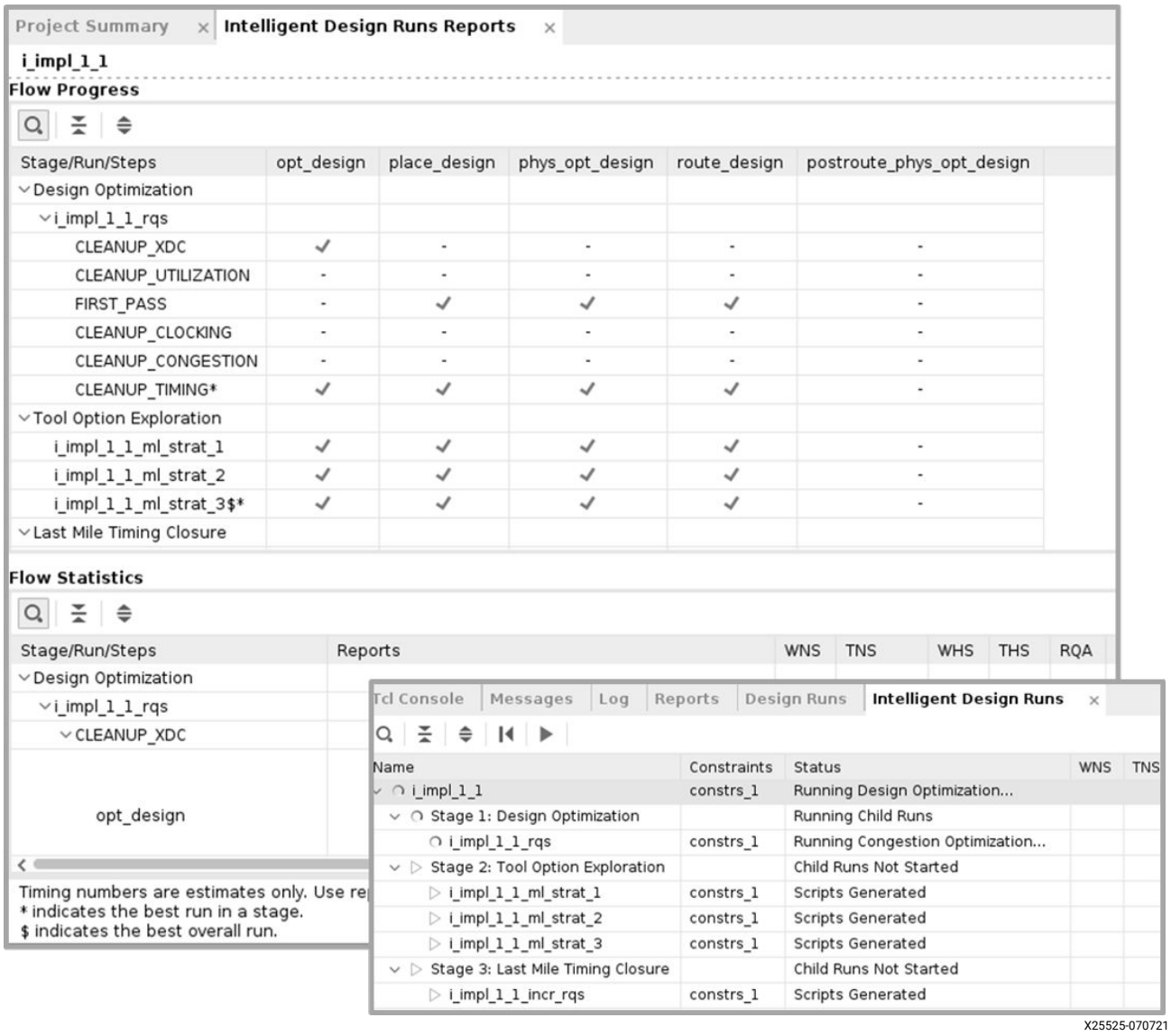
Figure 7: **Launch IDR with Simple Right-Click Menu Selection**



IDR appears in a new tab next to Design Runs. The IDR tab organizes runs hierarchically by stage, Design Optimization, Tool Option Exploration, and Last Mile (following figure, inset). The supporting implementation runs are contained underneath the stages. Unlike the default design runs, IDR can make decisions to stop, skip, or continue after evaluating changes in timing closure design metrics such as improvement or worsening of negative slack.

IDR progress can also be monitored from a dashboard called Intelligent Design Runs Reports next to the Project Summary, which displays the status of each step of each stage as shown in the following figure.

Figure 8: **Intelligent Design Runs and Dashboard**



X25525-070721

Finally, after a timing-closed result is achieved, a single pass design run can be extracted from IDR as a standard implementation run with the RQS suggestions, tool options, and incremental reference checkpoints if needed. This single pass run contains everything needed to recreate the timing-closed result in a single implementation run.

The IDR report summarizes the progress throughout IDR as it passes through each stage. An excerpt from a successful IDR run is shown in the following figure.

Figure 9: **The Intelligent Design Runs Report**

1. IDR Flow Summary

IDR Flow Stage	IDR Flow Step	PostRoute-WNS	PostRoute-RQA	Route-Status	Suggestions & Strategy files	Status
RQS Design Improvements	First Pass	-	-	-	-	-
RQS Design Improvements	Utilization	-1.968	2	Routed	stage1_util_suggestions.rqs	-
RQS Design Improvements	Clocking	-	-	-	-	Skipped - No issue found
RQS Design Improvements	Congestion	-0.510	3	Routed	stage1_congestion_suggestions.rqs	-
RQS Design Improvements	Timing	-0.162	4	Routed	stage1_timing_suggestions.rqs	Best result from stage 1
Expand Placer Solution	ML Strategy-1	-0.301	4	Routed	checkpoint_postlogicoptSuggestionFile1.rqs	-
Expand Placer Solution	ML Strategy-2	-0.171	4	Routed	checkpoint_postlogicoptSuggestionFile2.rqs	-
Expand Placer Solution	ML Strategy-3	-0.019	4	Routed	checkpoint_postlogicoptSuggestionFile3.rqs	Best result from stage 2
Last Mile	RQS-Incremental	0.005	5	Routed	stage3_suggestions.rqs	-

X25526-070721

Next to the descriptions of each IDR flow stage and IDR flow step are the key indicators of timing closure progress.

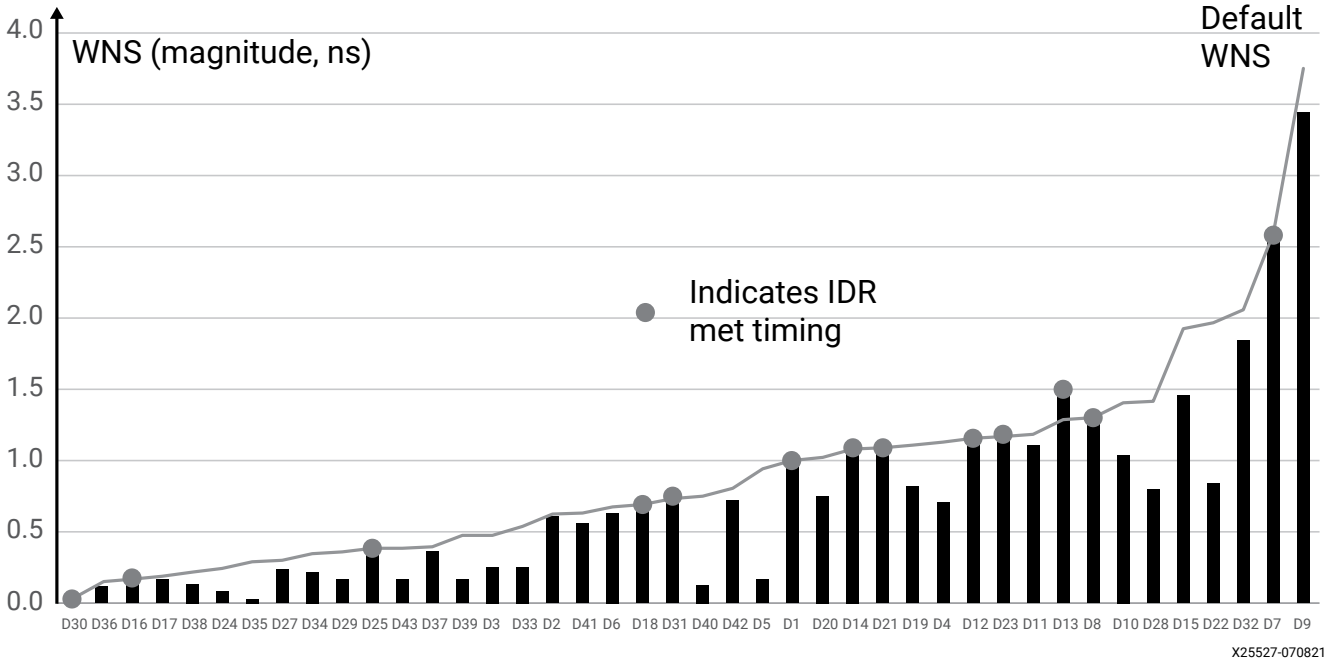
- **Worst Negative Slack (WNS):** The amount by which the worst timing path exceeds its requirement. The WNS value should steadily decrease if IDR improves timing.
- **Report QoR Assessment (RQA):** The prediction of the likelihood to meet timing on a scale from 1 (not likely) to 5 (very likely). The assessment should steadily increase if IDR improves timing.

The report for this particular design reveals the following:

- Early on, timing was in very poor condition with WNS of –1.968 ns and the assessment score was 2, which indicates little likelihood to meet timing.
- In Stage 1, design optimization made significant improvements, in particular in congestion and timing. WNS shrank to –0.162 and the assessment score jumped to 4 (likely to meet timing).
- In Stage 2, the third compilation strategy almost met timing, missing by only 19 ps.
- The result of Stage 2 showed sufficient progress to proceed to Stage 3, which is the last mile stage. Here the combination of incremental compile and path-specific suggestions closed timing.

In the previous example, IDR was very successful at steadily improving timing until achieving timing closure but results can vary depending on the design. Prior to the Vivado Design Suite 2021.1 release, IDR was tested on a collection of the most challenging designs to date, to measure out-of-box improvements against the Vivado tools defaults. The baseline for comparison is the out-of-box Vivado tools defaults, where none of the 42 designs met timing, with WNS ranging from –0.032 ns all the way to –3.744 ns. Applying IDR to each design consistently showed significant improvement. This is illustrated in the graph in following figure.

Figure 10: **IDR Results on a Suite of Challenging Designs**

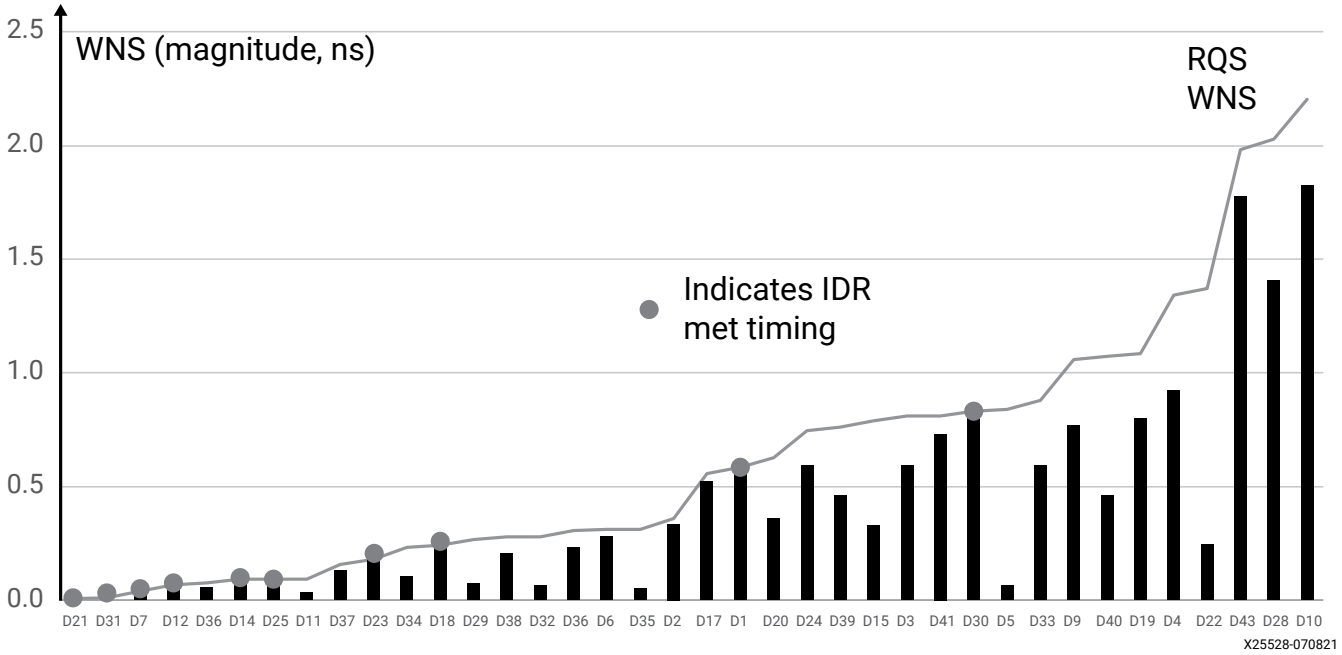


In this graph:

- The line indicates the magnitude of the original, out-of-box WNS using the Vivado tools defaults. The design results are sorted in increasing magnitude of WNS or in order from best to worst timing.
- The bars for each design indicate the amount of improvement from IDR or the WNS reduction. Note that for almost every design, the bar meets or nearly meets the line, showing that IDR makes substantial progress toward the original timing requirement.
- In 13 designs, around 30%, IDR was able to meet or exceed timing requirements where the defaults failed. Nearly half of the runs were within 50 ps of meeting timing.

A more compelling comparison uses RQS as the baseline. RQS is the incumbent leading automatic timing closure feature in Vivado tools and is able to close timing on three of the designs, leaving 39 with timing violations. With a less favorable baseline, IDR still shows impressive gains as seen in the graph of the following figure.

Figure 11: **IDR Results on a Suite of Challenging Designs Compared to RQS**



When compared to RQS, the original WNS is much lower in magnitude but is still able to improve WNS to meet or almost meet timing on nearly every design. These results show that even on the most challenging designs, IDR provides substantial QoR gains with virtually no additional effort.

Conclusion

IDR is a breakthrough in productivity, providing push-button access to an advanced, powerful timing closure feature giving an average 10% improvement in maximum clock frequency. IDR builds intelligence into the design process with both rule-based and ML-driven features and can navigate the largest and most complex designs. IDR is just scratching the surface of design closure. As IDR becomes established, future possibilities include:

- More *on-the-fly* prediction to reduce the need for iteration.
- Additional intelligence built into the last mile stage to explore more algorithms and reduce compile time.
- Finer grained decision-making while expanding the coverage of design metrics, enabling more efficient execution, and discovery of shortcuts.

IDR will continue to fulfill its promise of faster timing closure, freeing up more time to focus on product innovation and differentiation.

References

These documents provide supplemental material useful with this guide:

1. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
07/28/2021 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices