WHITE PAPER

FPGA

Using Intel® Quartus® Prime Software to Maximize Performance in the Intel HyperFlex™ FPGA Architecture

The Intel Quartus Prime software includes algorithms that take advantage of Intel Stratix® 10 HyperFlex registers, resulting in extremely fast clock speeds.

Author

Gordon Chiu
Director, Software Engineering
Intel Programmable Solutions Group

Table of Contents

Introduction

The innovative Intel® HyperFlex™ FPGA Architecture features a "registers everywhere" design that includes ubiquitous retiming and pipelining registers, called Hyper-Registers. These Hyper-Registers are available in every routing wire on the device. Combined with the new Intel Quartus® Prime Hyper-Aware design flow, these Hyper-Registers allow designers to break the performance barrier, achieving 2X the core performance in Intel Stratix 10 FPGAs and SoCs compared to previous-generation high-performance FPGAs, with clock speeds of up to 1 GHz. To achieve this high performance, designers use three design optimization strategies:

- **Hyper-Retiming** leverages the fine-grained nature of the Hyper-Registers. The Intel Quartus Prime software can perform meticulous retiming operations to balance the slack between two paths. It shifts the traditional adaptive logic module (ALM) register to a Hyper-Register that is in a better location for timing.
- **Hyper-Pipelining** capitalizes on the fact that the delay to enter a Hyper-Register is significantly lower than the delay to enter a conventional register. Adding additional registers to a long routing path does not incur the significant delay cost of routing into a logic array block (LAB) to utilize the register. Therefore, paths can be more deeply pipelined. The Intel Quartus Prime software can move existing pipeline registers to optimal locations, alleviating the need to break up existing design blocks to add intermediate pipeline stages.
- **Hyper-Optimization** unlocks the high-performance potential of a design. After the designer applies the Hyper-Retiming and Hyper-Pipelining techniques, the design's true critical path (for example, a computational loop) is exposed. Retiming or pipelining techniques alone cannot improve this true critical path. To improve performance, designers exploit parallelism in the design by applying decomposition techniques such as Shannon's decomposition [(1)] or time domain multiplexing (TDM).

Challenges with conventional register retiming

Conventional register retiming [(2)] is a sequential optimization technique that designers use to improve design performance. In this transformation, a register can be retimed across combinational logic. The circuit's internal sequential state may be different, but the circuit output is functionally and provably equivalent to the original design specification. Figure 1 shows backward and forward retiming across combinational logic implemented in look-up-tables (LUT).

In a forward-push operation, a register is moved forward across a combinational node. The delay that was on the register output moves to the register input. Designers can improve the critical path at the register output at the expense of a path at the register input. This balancing operation can significantly improve design performance.

Conventional FPGA design flows leverage conventional sequential register retiming in different ways, with only moderate success. During logic synthesis, registers can be retimed to minimize the depth of combinational logic paths in the circuit. Because this operation is done early in the FPGA design flow, where there is poor prediction of the final place-and-route, the gains from this optimization are limited. Thus, it is difficult to predict and focus on the design's most critical paths.

Physical synthesis, introduced in the Quartus II software in 2004, performs register retiming after placement and before routing. Although it addresses the delay prediction problem, leading to more targeted optimizations and improvement, the scope of retiming operations is limited. The software still needs to legalize the retimed register or combinational logic placement.

Hyper-Aware design flow

The Intel HyperFlex FPGA Architecture adds a Hyper-Register to each interconnect routing segment, significantly increasing the number of registers available for retiming and pipelining. To take advantage of these Hyper-Registers, Intel's design flow for the Intel HyperFlex FPGA Architecture leverages automated retiming optimization techniques to improve circuit performance and suggest further enhancement. This flow minimizes the effort traditionally spent in design modification and iteration. The Intel Quartus Prime software includes the following enhancements to enable the new design flow for the Intel HyperFlex FPGA Architecture:

The Intel Quartus Prime software includes the advanced Hyper-Retimer optimization algorithms that perform automatic register retiming. These new algorithms achieve high performance without the typical effort associated with manually retiming a design for a conventional FPGA architecture.

To improve designer productivity and accelerate the path to high performance, the Intel Quartus Prime software introduces the Fast Forward Compile tool, which can analyze a design to provide step-by-step recommendations for RTL changes to improve performance. The tool provides the estimated performance improvement for these changes. With these changes, such as adding pipeling stages and removing retiming restrictions, designers can explore performance limits in the Intel HyperFlex FPGA Architecture.

Intel has redesigned the Intel Quartus Prime synthesis and place-and-route algorithms (making them Hyper-Aware) so that it is easy to take advantage of the Hyper-Retimer and Fast Forward Compile tools.

Hyper-Retiming

The Hyper-Retimer relocates existing registers to achieve higher overall performance. Leveraging the novel Intel HyperFlex FPGA Architecture with its ubiquitous retiming and pipelining registers, the Intel Quartus Prime software can make very late changes to the design's registers: after placement and routing. The optimizations are based on

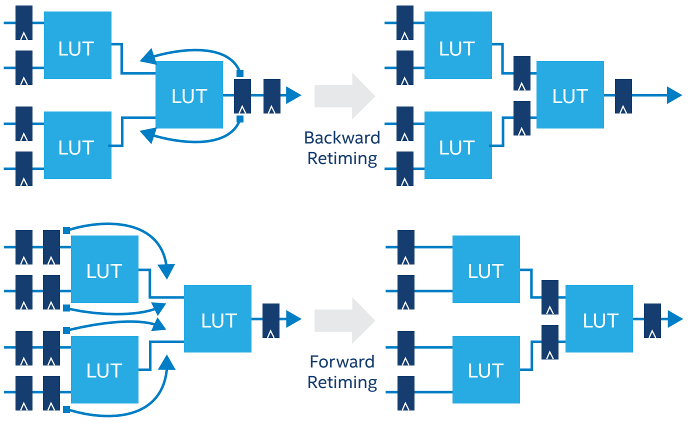


Figure 1. Backward and Forward Retiming

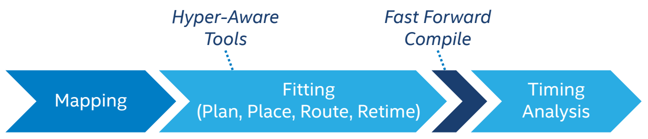


Figure 2. Intel HyperFlex FPGA Architecture Design Flow

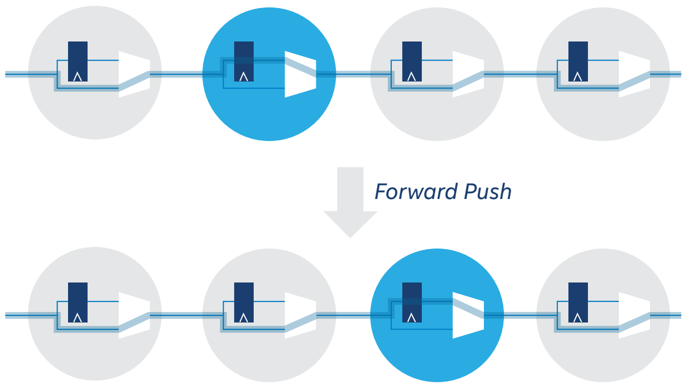


Figure 3. Hyper-Retiming Forward Push

extremely accurate delay predictions, resulting in maximum performance.

The design flow includes Hyper-Retiming optimizations in the Fitter, which performs register retiming at a global level, simultaneously repositioning all registers within the design to improve performance. Figure 2 shows the design flow for the Intel HyperFlex FPGA Architecture.

Register repositioning is easy in the Intel HyperFlex FPGA Architecture. Every Hyper-Register in the interconnect routing is optionally bypassable; it can be configured as a register or a simple wire. As a result, performing a forward-push move (see Figure 3) is like flipping a switch–turn some registers off and turn others on.

Hyper-Pipelining

In conventional pipelining, designers add register stages to break up long routing paths. Designs that were originally targeted for conventional, register-starved architectures obtain performance improvements by inserting additional pipeline stages into the circuit.

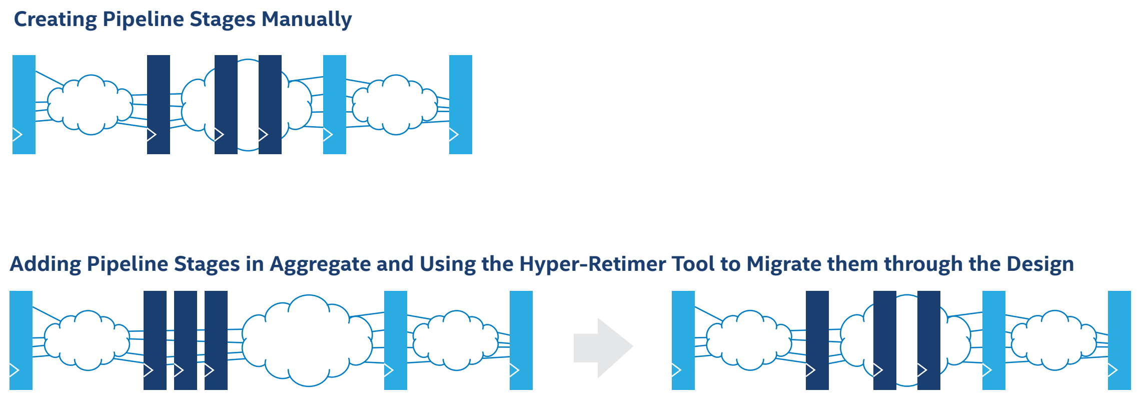

Creating Pipeline Stages Manually
Adding Pipeline Stages in Aggregate and Using the Hyper-Retimer Tool to Migrate them through the Design

Figure 4. Hyper-Pipelining vs. Conventional Pipelining

In a conventional design, additional pipeline stages must be distributed throughout the design to provide a performance benefit. This process can be time consuming to implement and difficult to verify. Additionally, it often requires breaking up hardware description language (HDL) logic and inserting register stages at unnatural places in the design. It may also require modifications to legacy intellectual property (IP) blocks or modules, forcing design units to be re-qualified and re-verified.

Hyper-Pipelining simplifies this process. Designers add pipeline stages in aggregate at one place in the design and the Intel Quartus Prime Fitter can retime the logic throughout the circuit (see Figure 4). Hyper-Pipelining requires much less effort than conventional, manual pipelining optimizations.

Hyper-Optimization

After removing retiming restrictions and adding pipelining to the design, designers may need to perform further optimization to reach high performance goals. In these cases, designers need to find the design's critical paths and optimize those trouble spots. The Intel Quartus Prime synthesis and place-and-route algorithms have been redesigned to be Hyper-Aware: the software algorithms can take advantage of the Intel HyperFlex FPGA Architecture. The synthesis and place-and-route algorithms use retiming aware models and delays to predict the true critical paths and focus optimization efforts.

In addition to providing large performance gains, the Hyper-Aware tools can reduce the number of conventional registers a design uses. Moving simple registers from ALMs to the Hyper-Registers in the interconnect routing fabric frees valuable resources and reduces logic utilization (particularly for highly register-dominated designs).

Streamline exploration with the Fast Forward Compile tool

Conventional FPGA software flows stop optimizing when they hit a performance limit, for example, a path that cannot be improved by retiming or insufficient pipeline stages between two fixed end points. The designer then needs to interpret the tool's output, decide on a course of action, implement a design change or workaround, and re-run the tool to continue making progress towards performance goals. See Figure 6.

To improve designers' efficiency, the Hyper-Aware design flow features the Fast Forward Compile tool. When the Intel Quartus Prime software finds a performance restriction during exploration, the software analyzes a design to provide step-by-step recommendations for RTL changes to improve performance. The Fast Forward tool can analyze the effect of removing retiming restrictions and adding pipelining and provide an estimate of the design f_{MAX} performance improvement. For example, upon discovering a register that cannot be retimed due to a restriction (such as an asynchronous clear or a user preserve statement), the Fast Forward Compile tool analyzes what happens if the designer removes the restriction and provides a performance improvement estimate for the RTL change. See Figure 7.

Similarly, upon discovering a path that can be helped by additional pipelining, the Fast Forward Compile tool can can analyze the performance improvement if the designer inserts additional pipeline register stages. This register insertion is only performed at asynchronous transfers. These asynchronous transfers are the ingress or egress to clock domains; they cut clock transfers from unrelated clocks or transfers to or from I/O pins. These locations are natural places to insert additional pipeline registers. The additional registers can then be retimed and pipelined through the design to improve performance.

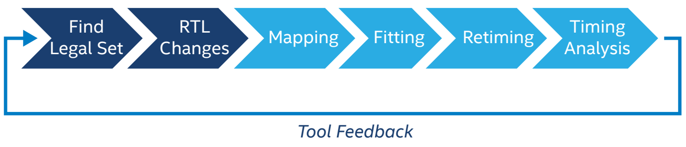
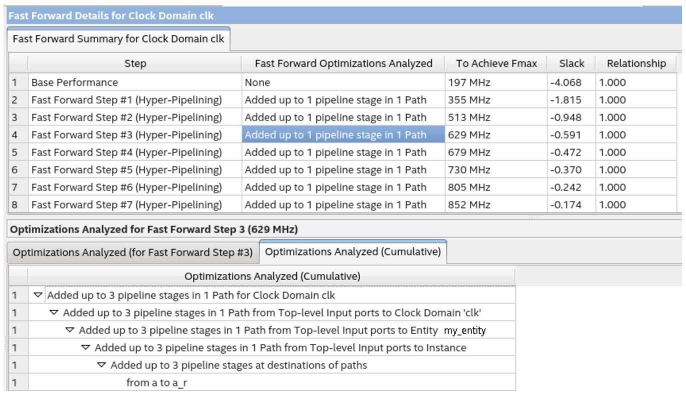


Figure 6. Conventional Design Flow


Fast Forward Details for Clock Domain clk

Fast Forward Summary for Clock Domain clk

	Step	Fast Forward Optimizations Analyzed	To Achieve Fmax	Slack	Relationship
1	Base Performance	None	197 MHz	-4.068	1.000
2	Fast Forward Step #1 (Hyper-Pipelining)	Added up to 1 pipeline stage in 1 Path	355 MHz	-1.815	1.000
3	Fast Forward Step #2 (Hyper-Pipelining)	Added up to 1 pipeline stage in 1 Path	513 MHz	-0.948	1.000
4	Fast Forward Step #3 (Hyper-Pipelining)	Added up to 1 pipeline stage in 1 Path	629 MHz	-0.591	1.000
5	Fast Forward Step #4 (Hyper-Pipelining)	Added up to 1 pipeline stage in 1 Path	679 MHz	-0.472	1.000
6	Fast Forward Step #5 (Hyper-Pipelining)	Added up to 1 pipeline stage in 1 Path	730 MHz	-0.370	1.000
7	Fast Forward Step #6 (Hyper-Pipelining)	Added up to 1 pipeline stage in 1 Path	805 MHz	-0.242	1.000
8	Fast Forward Step #7 (Hyper-Pipelining)	Added up to 1 pipeline stage in 1 Path	852 MHz	-0.174	1.000

Optimizations Analyzed for Fast Forward Step 3 (629 MHz)

Optimizations Analyzed (for Fast Forward Step #3) | Optimizations Analyzed (Cumulative)

	Optimizations Analyzed (Cumulative)
1	▽ Added up to 3 pipeline stages in 1 Path for Clock Domain clk
1	▽ Added up to 3 pipeline stages in 1 Path from Top-level Input ports to Clock Domain 'clk'
1	▽ Added up to 3 pipeline stages in 1 Path from Top-level Input ports to Entity my_entity
1	▽ Added up to 3 pipeline stages in 1 Path from Top-level Input ports to Instance
1	▽ Added up to 3 pipeline stages at destinations of paths
1	from a to a_r

Figure 7. Fast Forward Compile Recommendations

As it successively finds limits, the Fast Forward Compile tool works around the limits with proposed changes, giving an accurate estimate of the performance based on “what if” that design change was in place. These proposed changes are combined into a detailed recommendation list for achieving the new higher performance.

Simplify design modifications during realization

For the Intel Quartus Prime software to achieve higher levels of performance, some design changes must be realized.

- To achieve Hyper-Retiming performance, designers need to make changes to their reset strategy because Hyper-Registers do not have dedicated asynchronous clear circuitry. Asynchronous clears must be converted to synchronous or removed.
- To achieve Hyper-Pipelining performance, designers need to add additional pipeline stages to a design.
- To achieve Hyper-Optimization performance, designers need to restructure large computational loops.

The Intel Quartus Prime software cannot perform reset strategy changes or insert pipeline stages automatically. Automatically adding pipeline stages means the design would no longer match the input description (Verilog HDL or VHDL), which can cause problems during functional verification.

Identifying and evaluating design changes (such as inserting pipeline stages) is iterative and time-consuming. Intel recommends that designers use the Fast Forward Compile tool to analyze their design and to provide them with step-by-step recommendations to help achieve their design performance target.

Conclusion

To support the new Intel HyperFlex FPGA Architecture, Intel introduces the novel high-performance Hyper-Aware design flow in the Intel Quartus Prime software. Central to this flow are:

- Retiming algorithms that take advantage of the Hyper-Registers in the routing fabric to improve design performance after place-and-route.
- The Fast Forward Compile tool that reduces designer effort and iterations by recommending and evaluating changes to achieve high performance.

Combining these Hyper-Aware tools with the Intel HyperFlex FPGA Architecture results in 2X the core performance compared to previous-generation high-performance FPGAs, with ultra-fast clock speeds of up to 1 GHz.

References

[1] Sovani, Cristian, Tardieu, Oliver, Edwards, Stephen A, “Optimizing Sequential Cycles Through Shannon Decomposition and Retiming,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol 26, 2006.
[2] Leiserson, Charles E., and Saxe, James B, Retiming Synchronous Circuitry, August 1986.

Where to Get More Information

For more information about Intel and Intel Stratix 10 FPGAs, visit **https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html**

[1] http://www.altera.com/en_US/pdfs/literature/wp/wp-01220-hyperflex-architecture-fpga-socs.pdf
[2] http://www.altera.com/en_US/pdfs/literature/wp/wp-01231-understanding-how-hyperflex-architecture-enables-high-performance-systems.pdf