Stratix 10 HyperFlex Architecture Overview

HyperFlex™ ARCHITECTURE

ALTERA®

Agenda

- Delivering the Unimaginable
- The HyperFlex Architecture Advantage
- HyperFlex Technical Overview
- Quartus II Design Flow
- Performance Benchmarks
- Early Access Software and Training

HyperFlex™ ARCHITECTURE

Stratix 10 FPGAs and SoCs

Delivering the Unimaginable

ALTERA

Stratix 10 FPGAs and SoCs

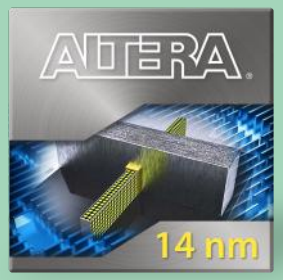
Delivering the Unimaginable

2X
Performance

Up to
70%
Lower power

4X+
Logic Density

Quad Core
64 -bit
ARM Cortex A53



14nm Intel Tri-Gate

+

HyperFlex™ ARCHITECTURE

New Architecture

HyperFlex™ Architecture Overview

Why Develop a New Architecture?

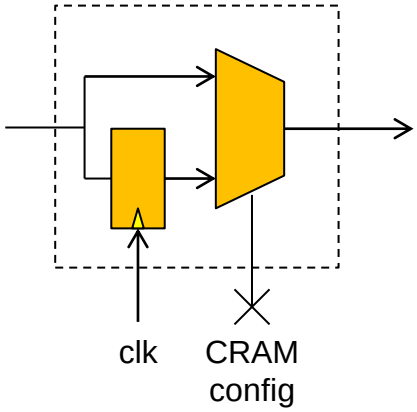
- Today's architectures will not hold up to tomorrow's performance demands
  - Making on-chip buses wider and wider is not sufficient, need to do more
- Need bigger step forward than we get with evolution
  - As geometries shrink, interconnect delays are dominating
- HyperFlex built on familiar concepts …
  - Retiming, Pipelining, Optimization
- With an innovative new approach
  - Allows unimaginable levels of performance
  - Not possible with conventional architecture

HyperFlex™ ARCHITECTURE

HyperFlex is New …
and It's a Big Improvement!

The HyperFlex Solution

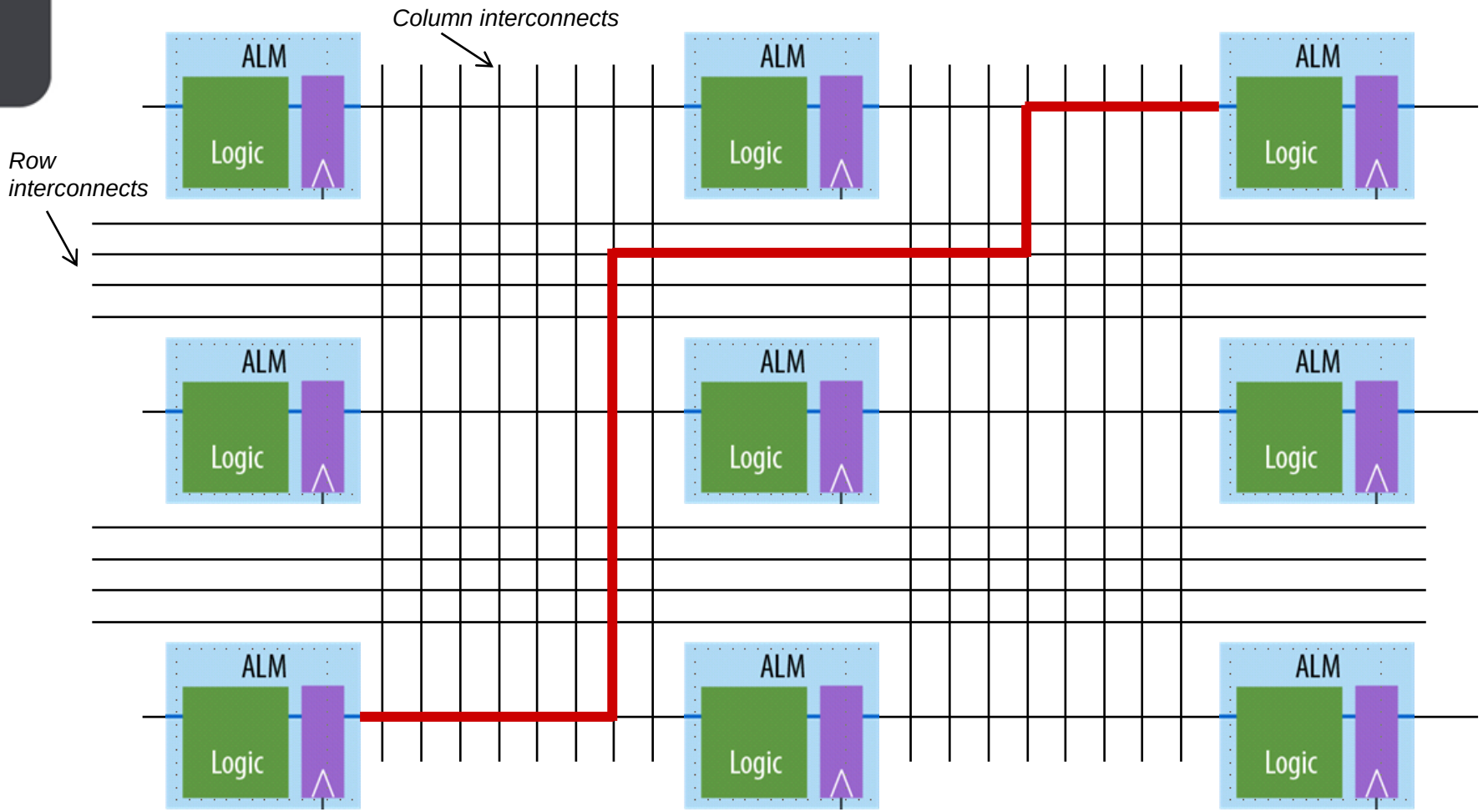
- HyperFlex has registers throughout the core fabric
- Bypassable Hyper-Registers in every routing segment
- Bypassable Hyper-Registers on all block inputs
  - ALMs, M20K blocks, DSP blocks, IO cells
- Register location is fine-grained
  - Throughout the interconnect
  - Available in optimal locations
- Allows new and better approach to
  - Retiming
  - Pipelining
  - Optimization



Bypassable Hyper-Register

Available "everywhere" throughout user logic and interconnect

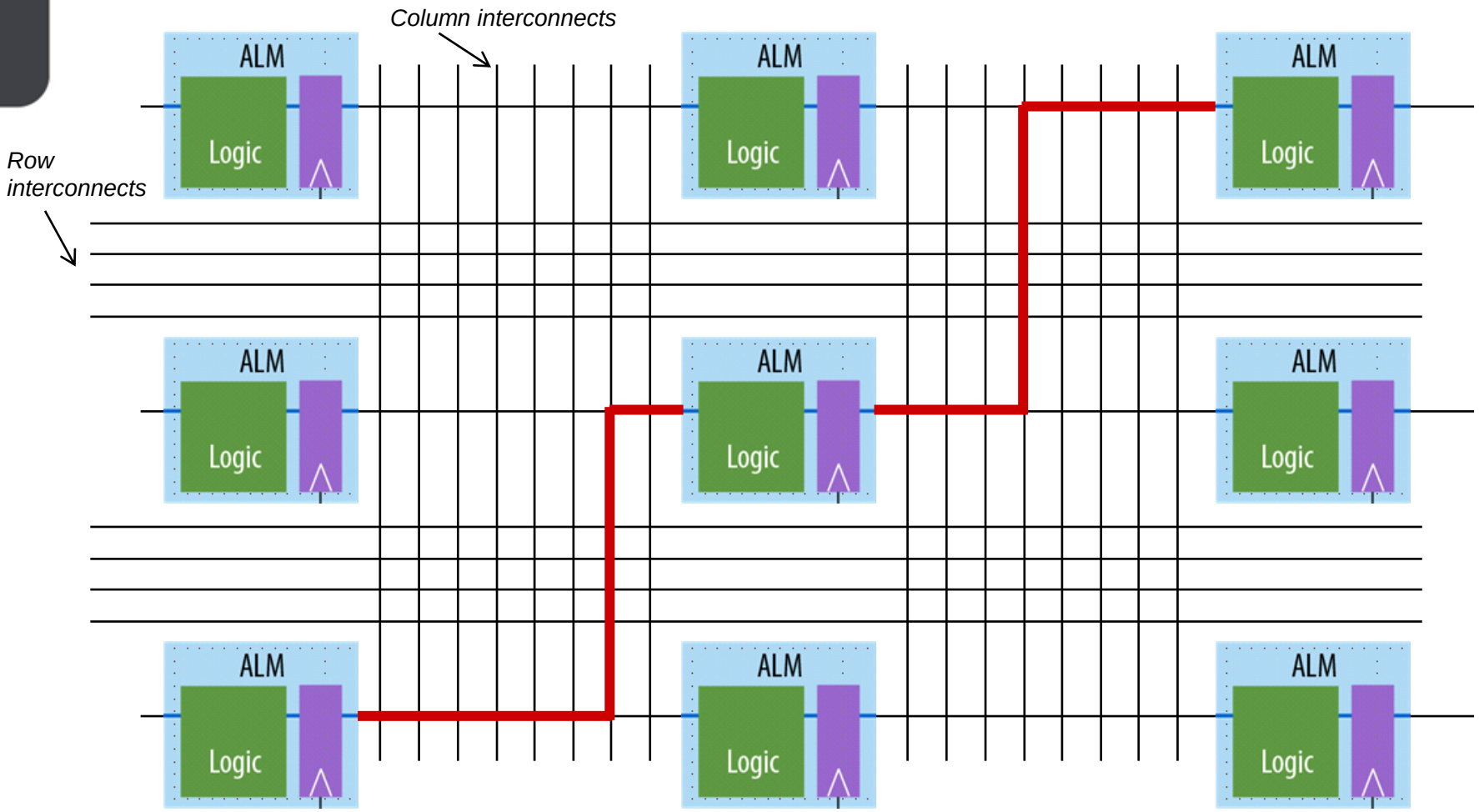
Conventional FPGA Architecture - Interconnect



Conventional logic cell to logic cell path

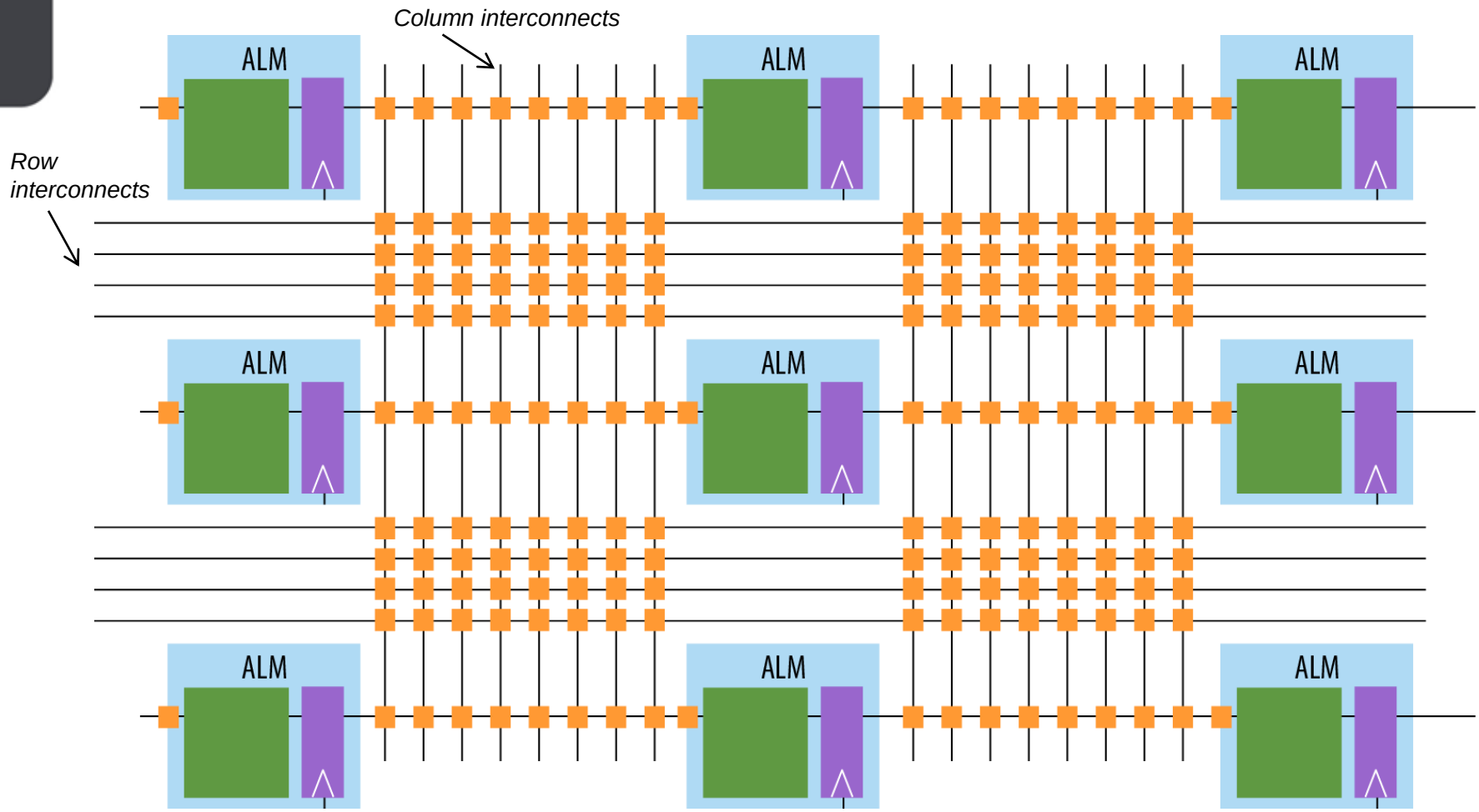
ALM = Adaptive Logic Module

Conventional FPGA Architecture - Interconnect



Conventional pipelined path between logic cells

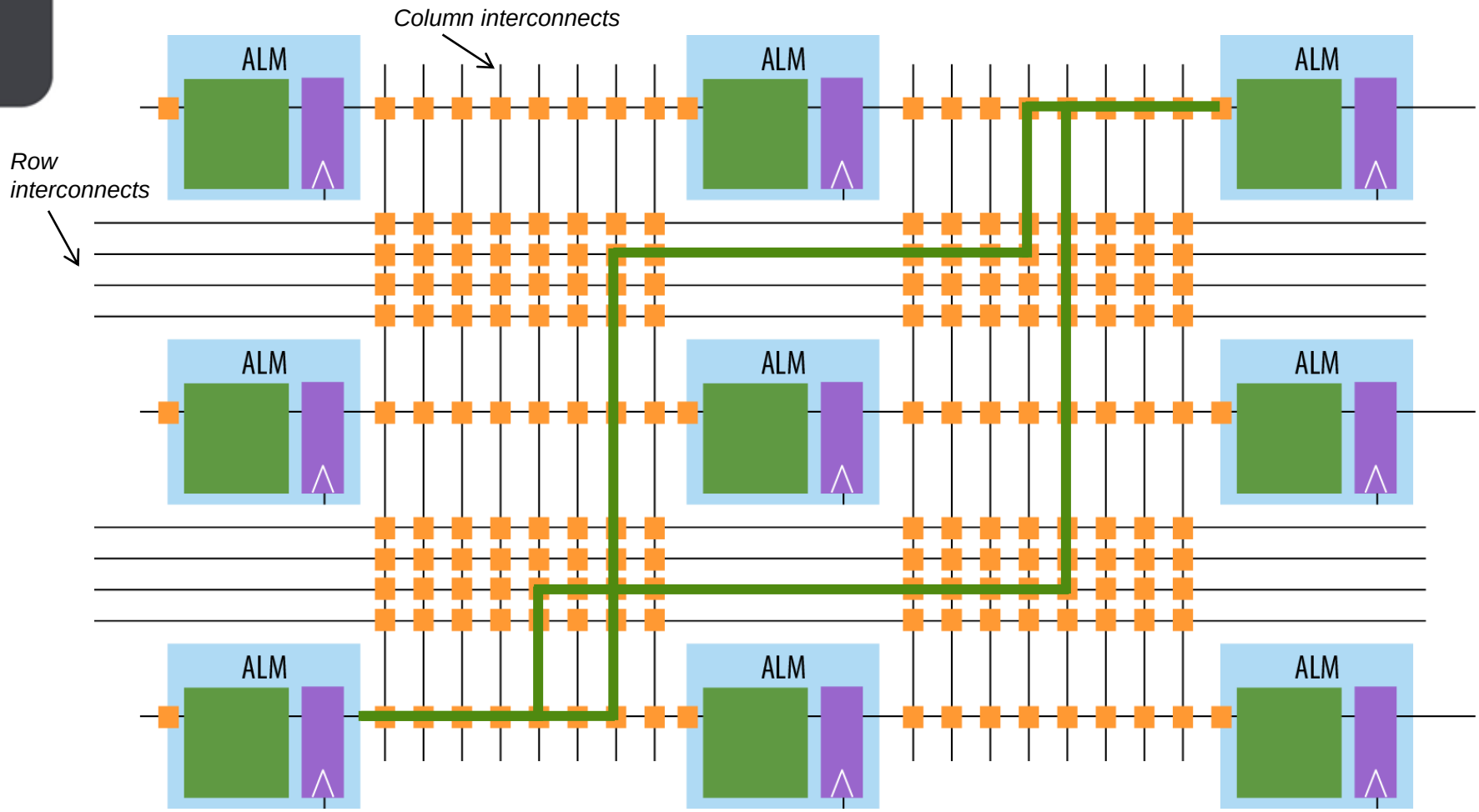
The HyperFlex Architecture - Interconnect



Number of Hyper-Registers >10X Number of ALM Registers!

= Hyper-Register

The HyperFlex Architecture - Interconnect



Any path between logic cells can be registered anywhere!

■ = ***Hyper-Register***

All New Stratix 10 HyperFlex Architecture

- Hyper-Registers throughout the FPGA fabric enable
  - Fine grain Hyper-Retiming to eliminate critical paths
  - Zero latency Hyper-Pipelining to eliminate routing delays
  - Flexible Hyper-Optimization for best-in-class performance
- Hyper-Aware design flow for accelerated timing closure with
  - Post place & route performance tuning
  - Hyper-register enabled synthesis and place & route for efficient pipelining
  - Fast Forward compilation enabling performance exploration
- Programmable clock tree synthesis offers
  - ASIC-like clocking to mitigate skew & uncertainty
  - Lowers power through intelligent clock enablement

How Do We Get to 2X Performance?

Step	Architecture Advantage	Customer Effort	Stratix 10 versus Stratix V (Average Gain)
1	Hyper-Retiming	No change, or minor RTL changes	1.4X
2	Hyper-Pipelining	Added Pipelining	1.6X
3	Hyper-Optimization	More Effort	2X or more

- Three-step process to achieve maximum performance
- Most of the gain comes from the first two steps
  - Uses well understood retiming and pipelining techniques
  - Large performance gains come from relatively small effort
- More effort required to implement the third step
  - May be required to achieve 2X or more performance gain

HyperFlex Optimization Strategies

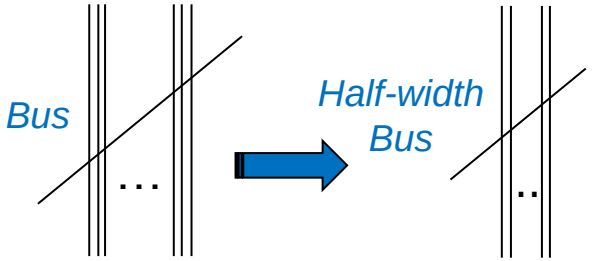
- Hyper-Retiming (effort level: low)
  - Moving **existing registers** in the design to optimize performance
  - Remove restrictions that prevent registers from retiming
  - Eliminates critical paths
- Hyper-Pipelining (effort level: medium)
  - Add **new registers to the design** to pipeline paths better
  - New registers need to be **inserted in the user RTL**
  - Eliminates routing delays
- Hyper-Optimization (effort level: high)
  - Optimizations to **improve the design further** beyond Hyper-Retiming and Hyper-Pipelining alone – **restructuring loops** in the design
  - Enables maximum performance

Core Performance is More Than Just Performance

- More Performance
  - Enabling higher performance applications
- Higher Productivity and Time to Market
  - Reduce engineering development time
  - Close timing faster

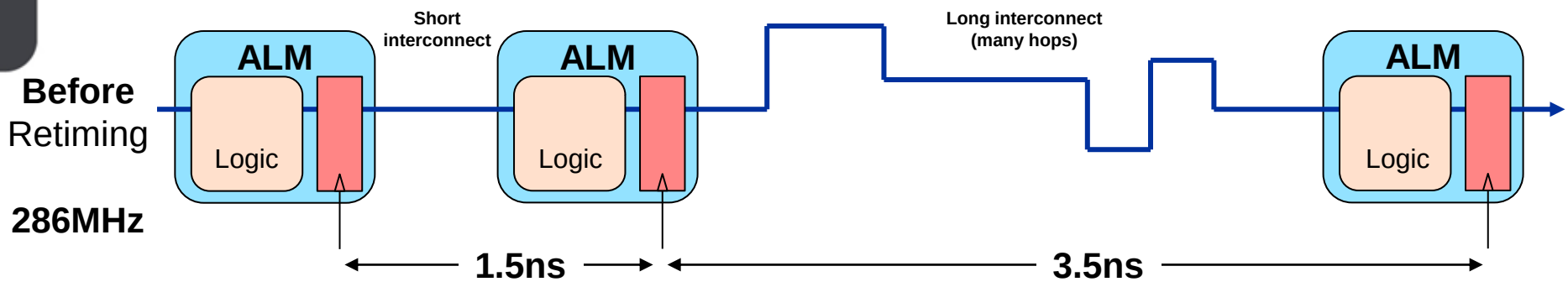
Quartus II
Design Software

- Reduce Device Cost
  - Choose a less-expensive slower device
    - With HyperFlex 2X performance, can you use a slower speed grade device?
  - Choose a less expensive smaller device
    - Can you use a smaller device now that you have Hyper-Registers throughout the fabric?
    - Could you run your bus at 1/2 the width and twice the frequency?

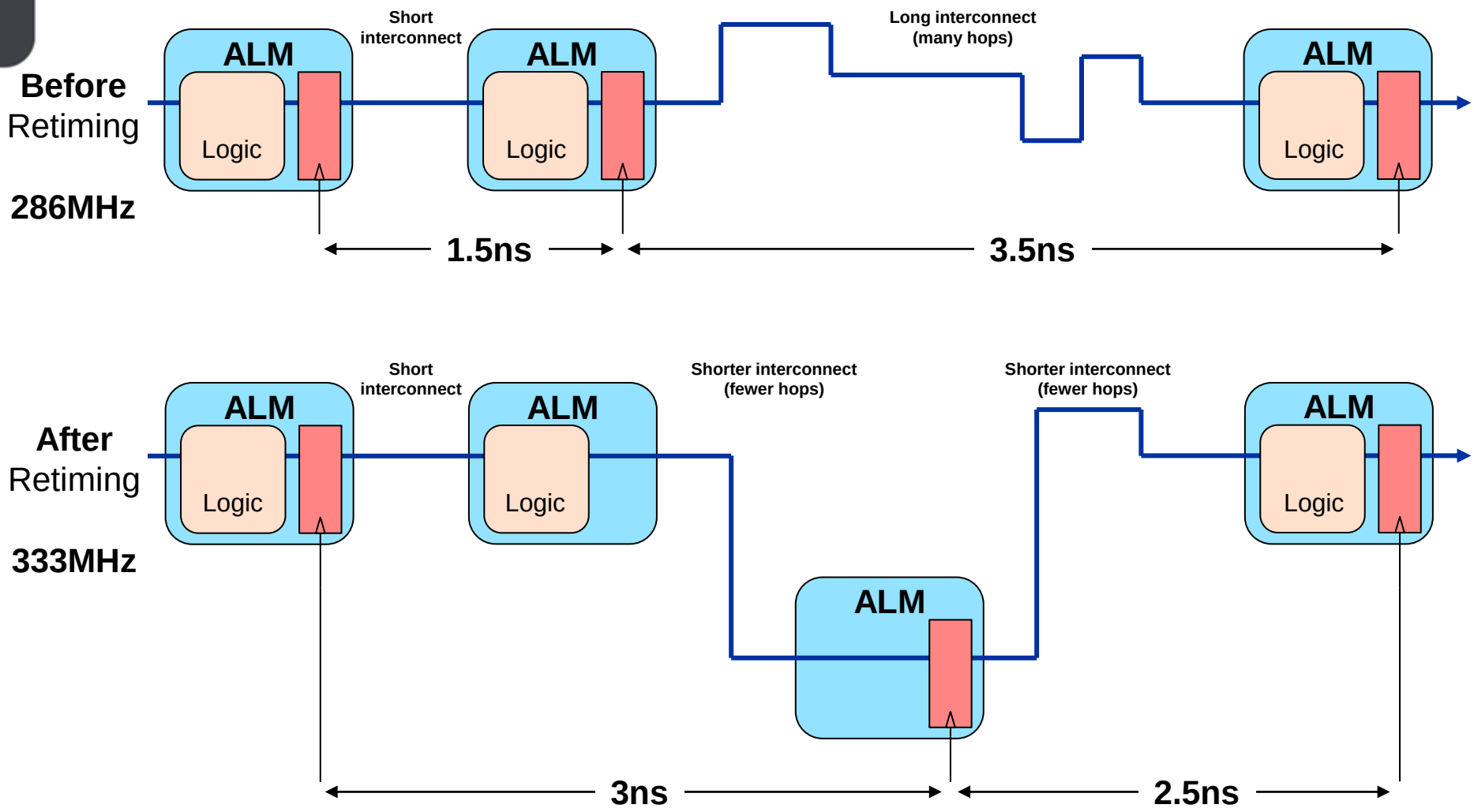


Hyper-Retiming

Conventional Register Retiming

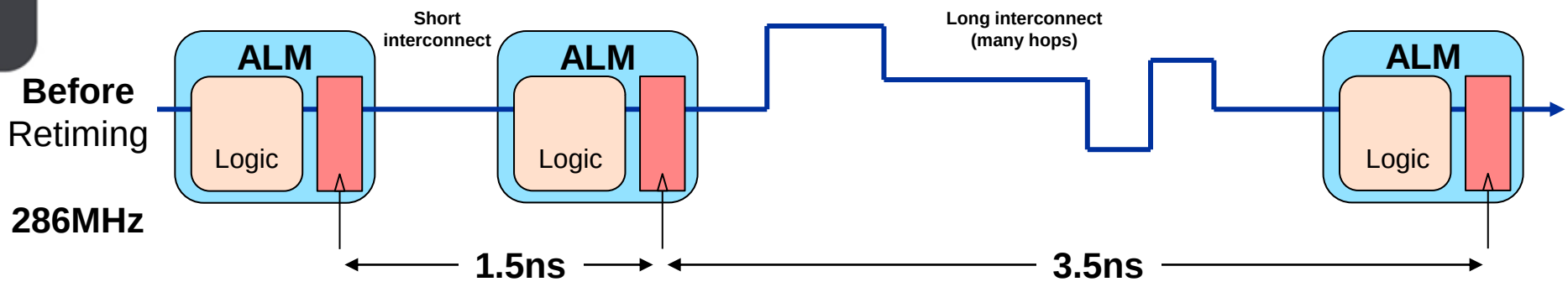


Conventional Register Retiming

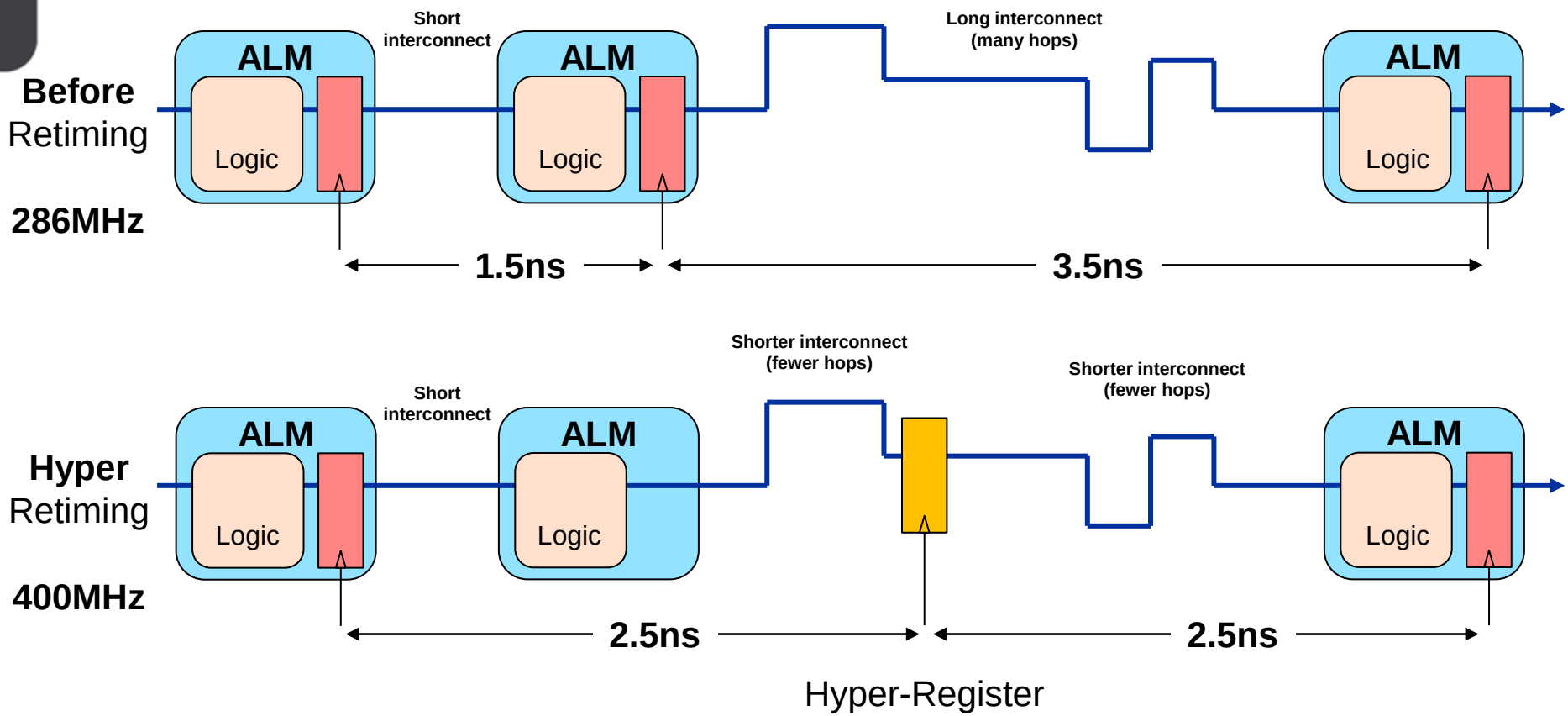


286MHz → 333MHz = 16% gain

Hyper-Retiming



Hyper-Retiming

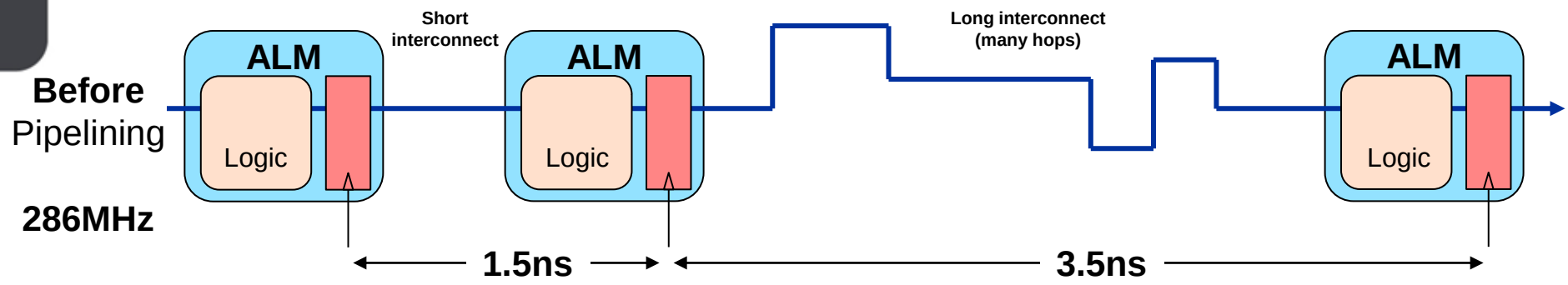


Hyper-Retiming step occurs AFTER place & route!

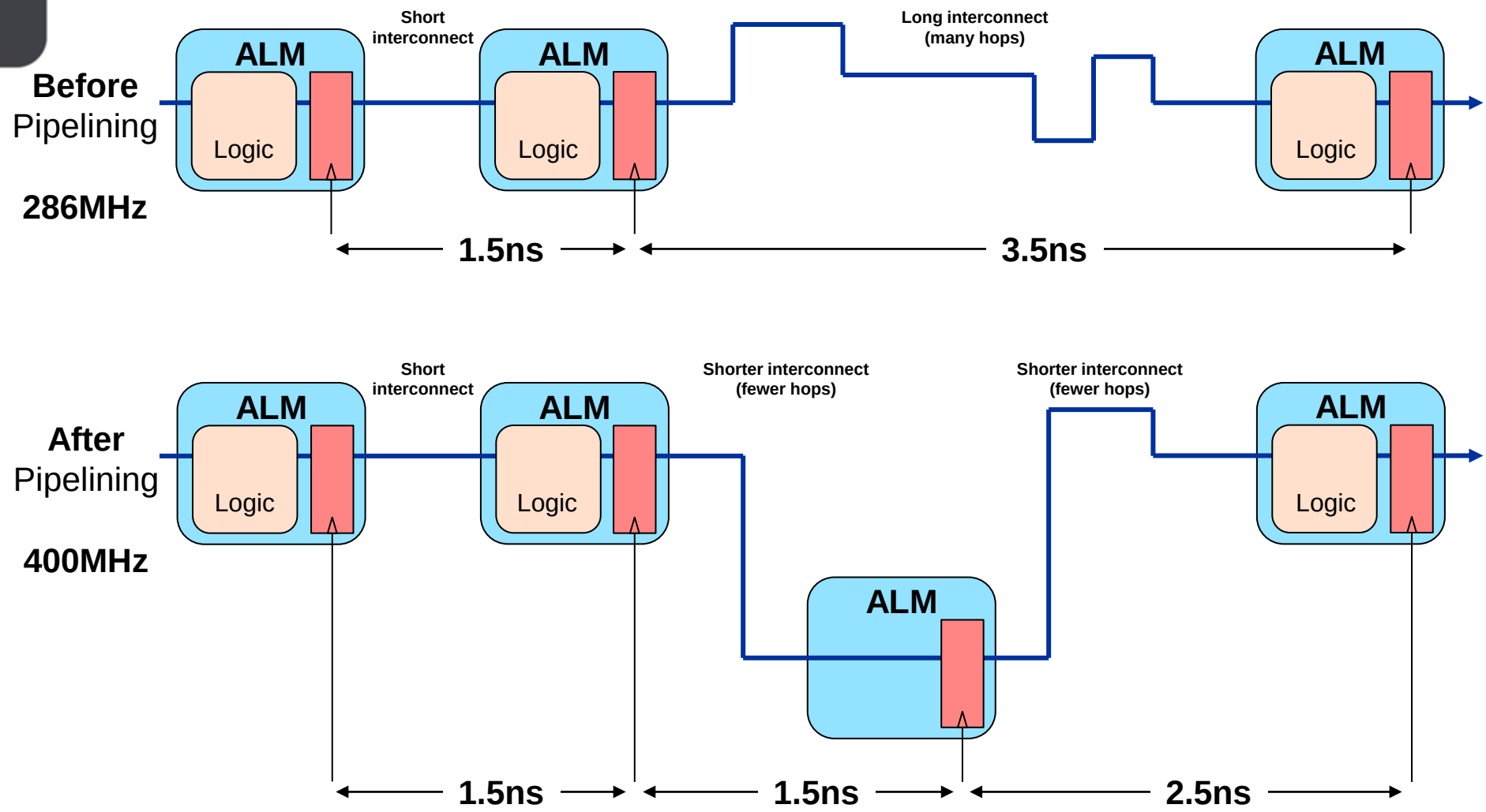
286MHz → 400MHz = 40% gain

Hyper-Pipelining

Conventional Pipelining



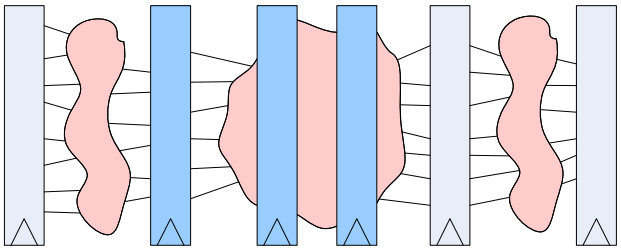
Conventional Pipelining



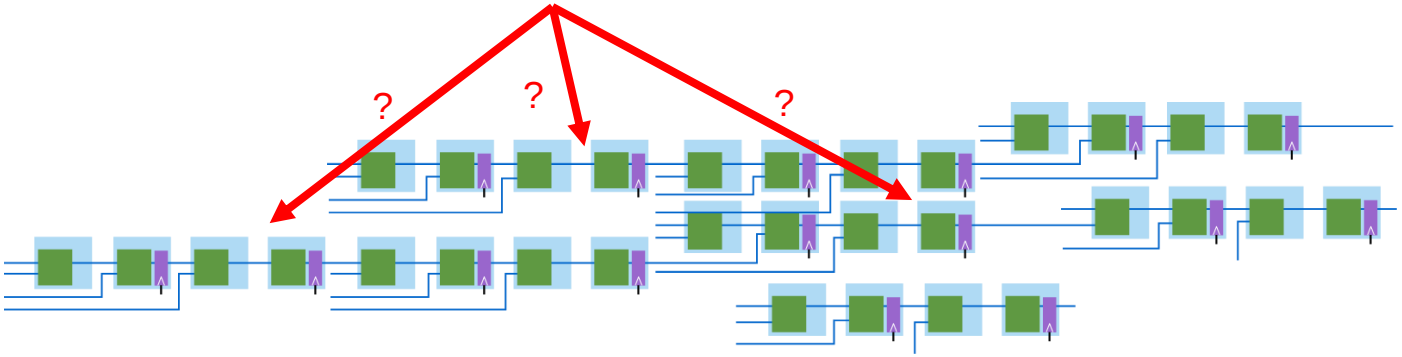
286MHz → 400MHz = 40% gain

Conventional Pipelining

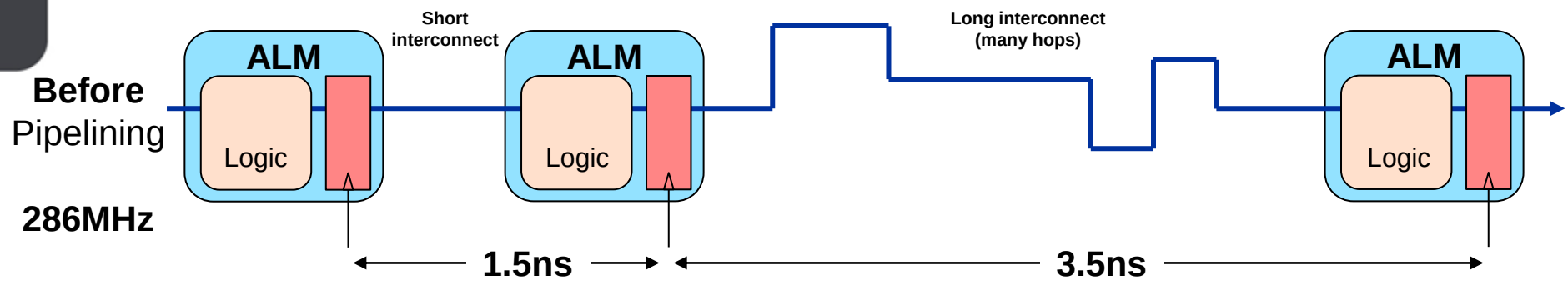
- Manually pipeline a block, by inserting registers at just the right place
- Can be **very difficult** to do for an **arbitrary combinational block**



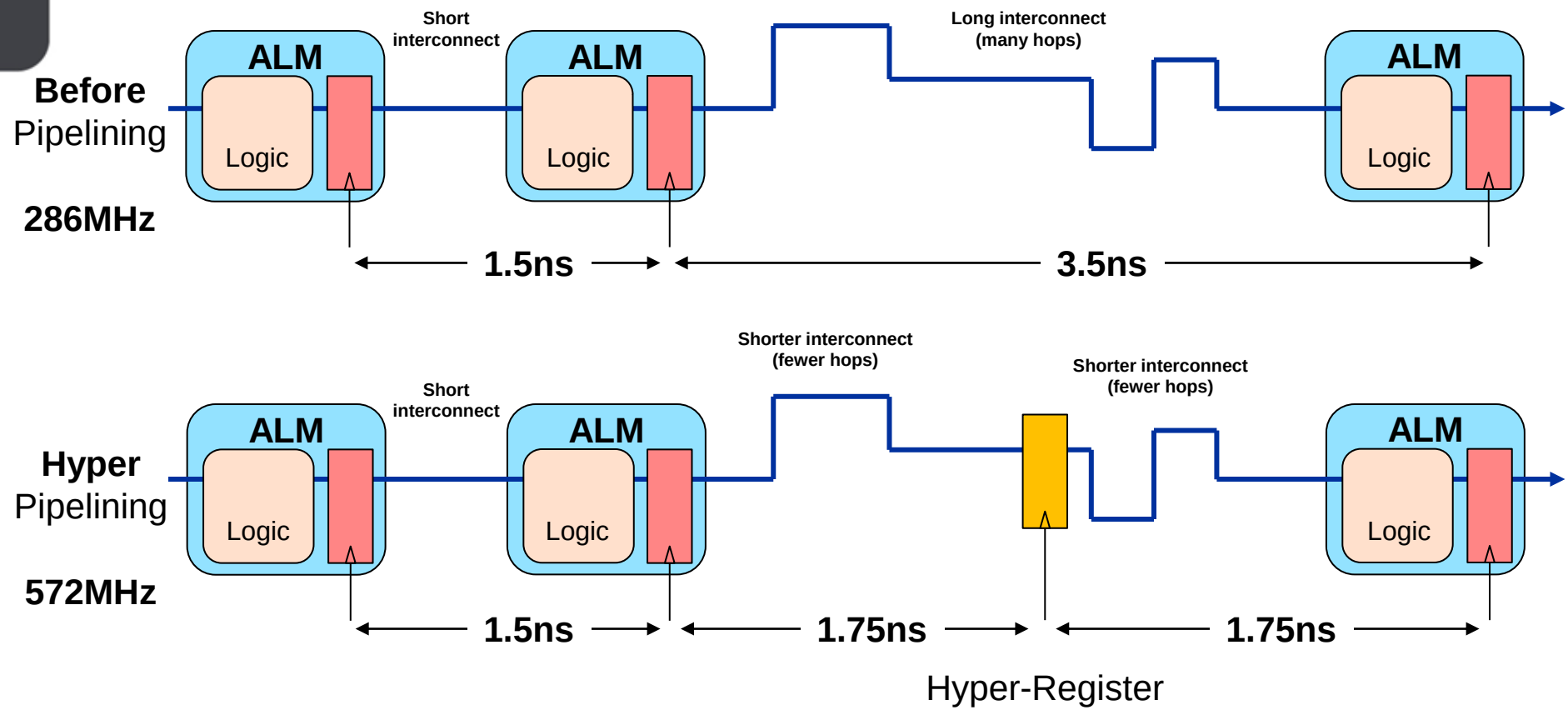
- This effort can be complex and time-consuming (iterative by nature)
- Requires designers to know **where** and **how many** registers to add



Hyper-Pipelining



Hyper-Pipelining

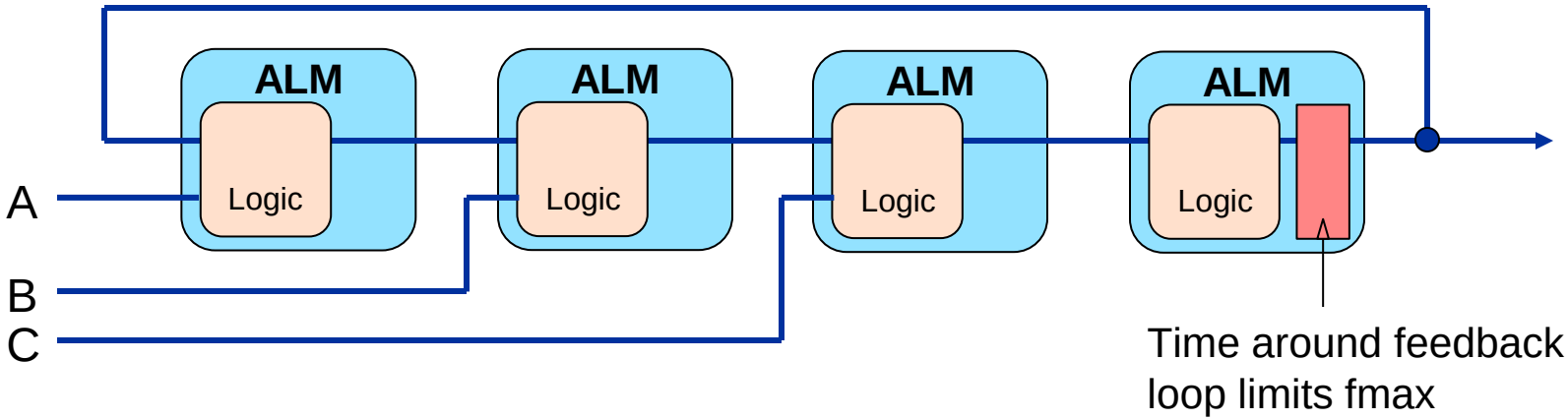


Final placement of Hyper-Pipeline register occurs AFTER place & route!

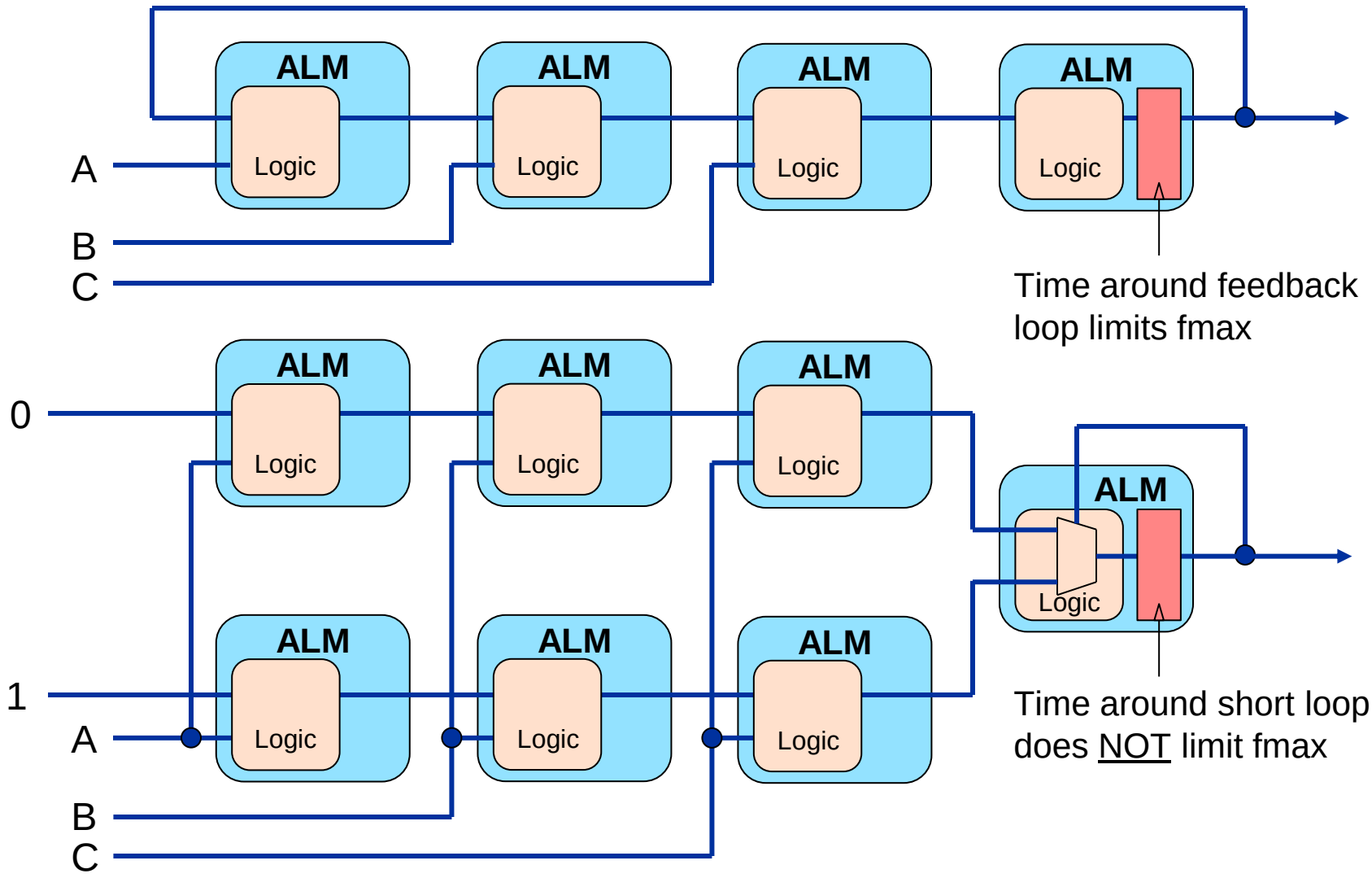
286MHz → 572MHz = 100% gain

Hyper-Optimization

Non-Optimized Feedback Loop

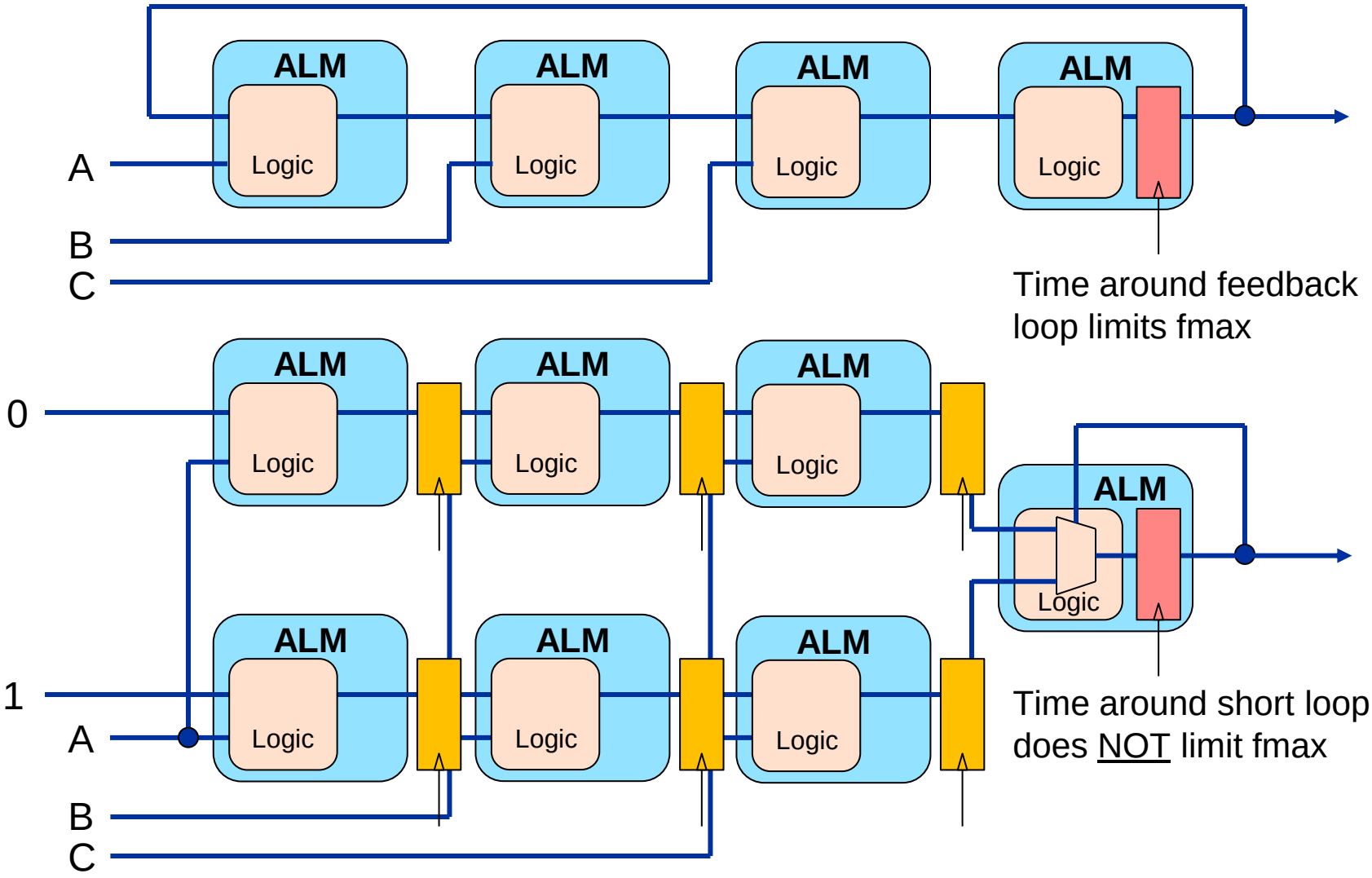


Optimization



Pre-compute to shorten feedback loops

Hyper-Optimization

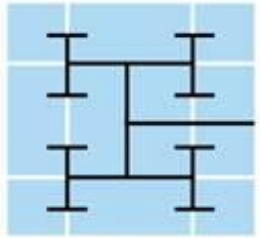


Pre-compute to shorten feedback loops

Programmable Clock Tree Synthesis

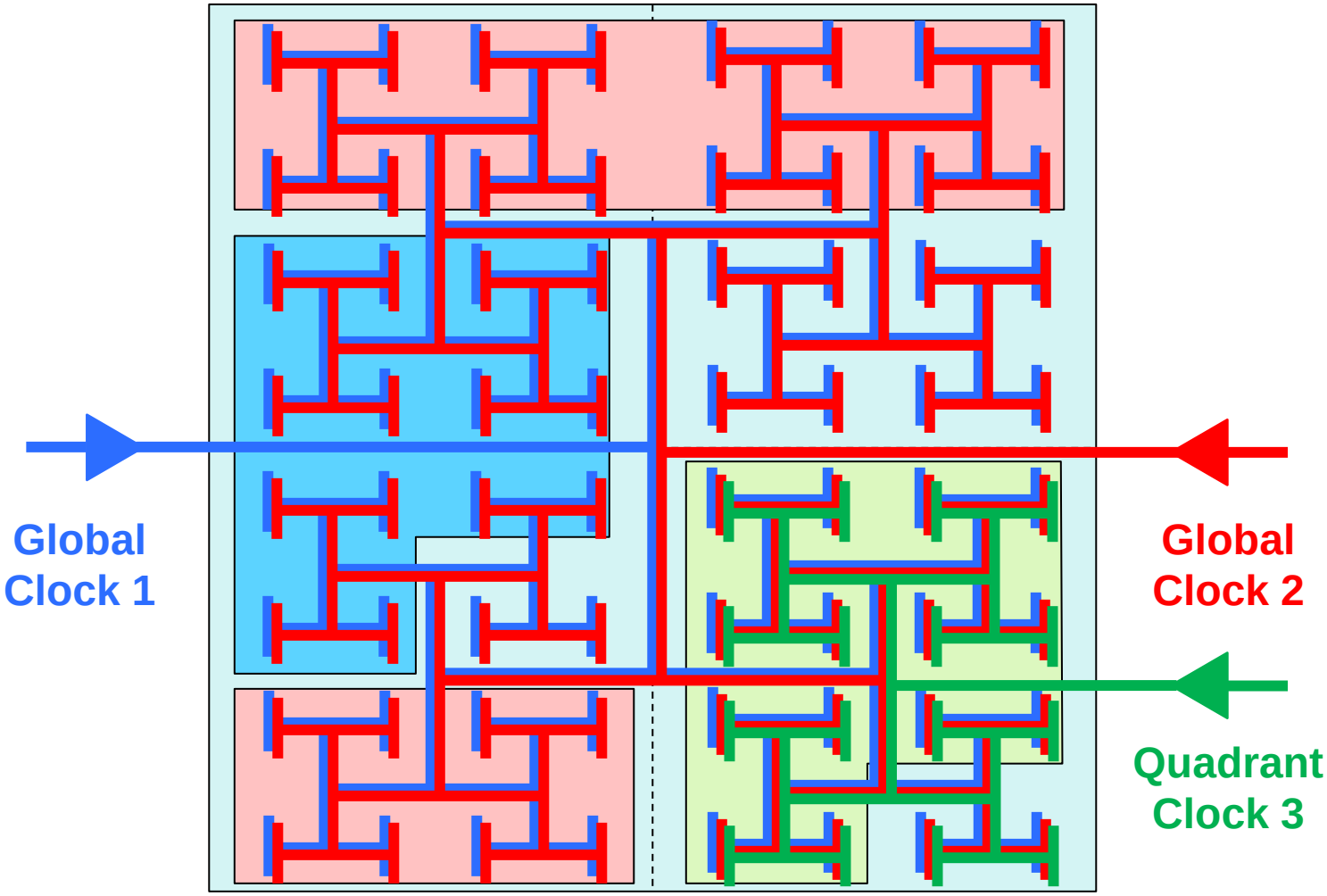
Programmable Clock Tree Synthesis

- Reduced clock tree insertion delay
  - Clock trees synthesized locally, precisely where needed
  - Less variability, tighter clock skew between end points
  - Allows higher fmax, up to 1GHz core clocking
- Greater clocking flexibility
  - Uses dedicated clock routing instead of predefined clock trees
  - Allows for greater number of clocks
  - Up to 32 independent clocks per sector (X rows by Y columns)
- Lower power dissipation
  - Smaller clock trees, fine-grained approach
  - Less switched capacitance
  - Smart branch disabling
- Backwards compatible
  - Easy to synthesize global, quadrant, regional clocks

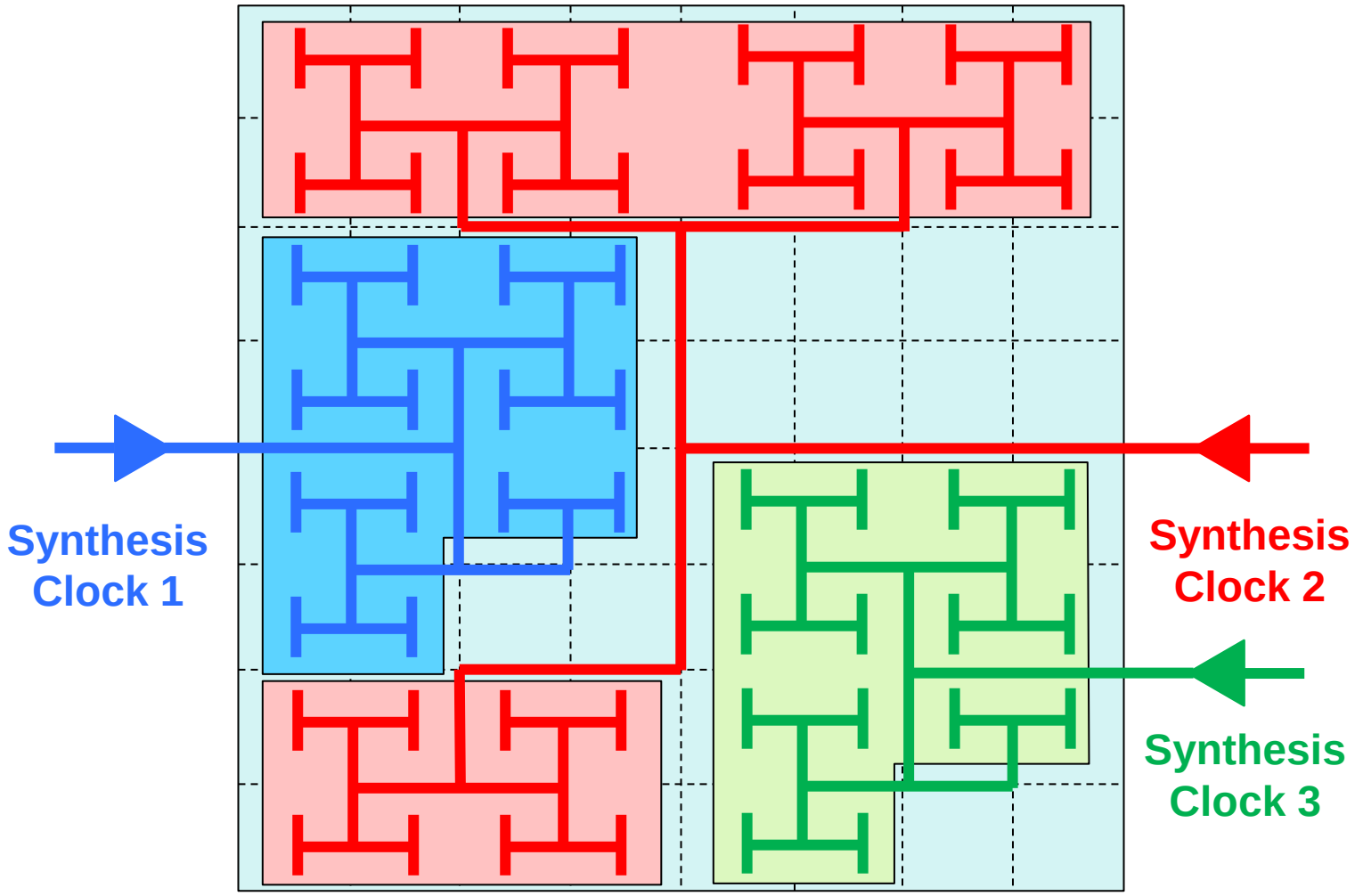


Programmable Clock Trees Support "Registers Everywhere"

Conventional Global/Regional Clock Architecture

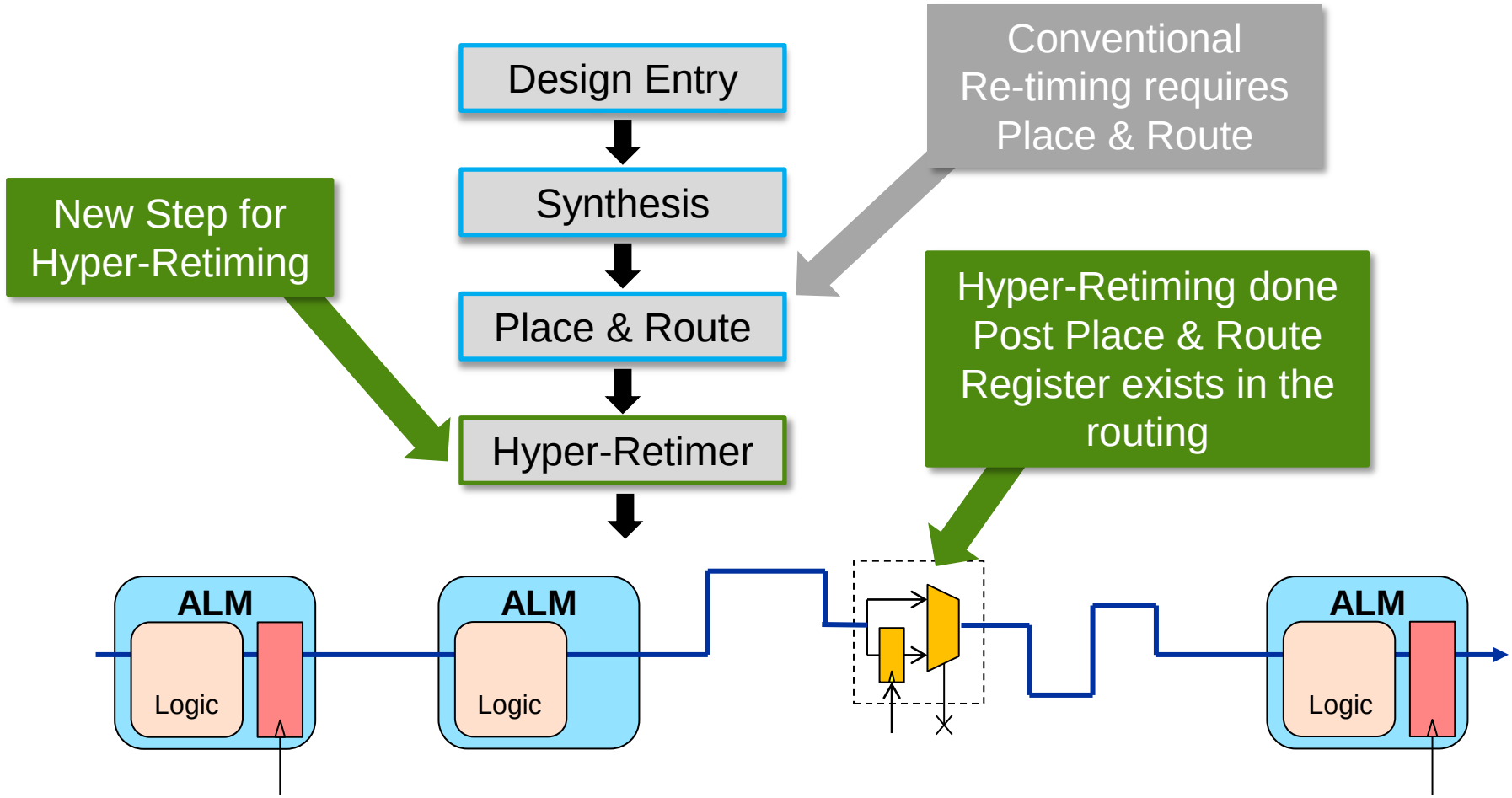


Stratix 10 Programmable Clock Tree Synthesis

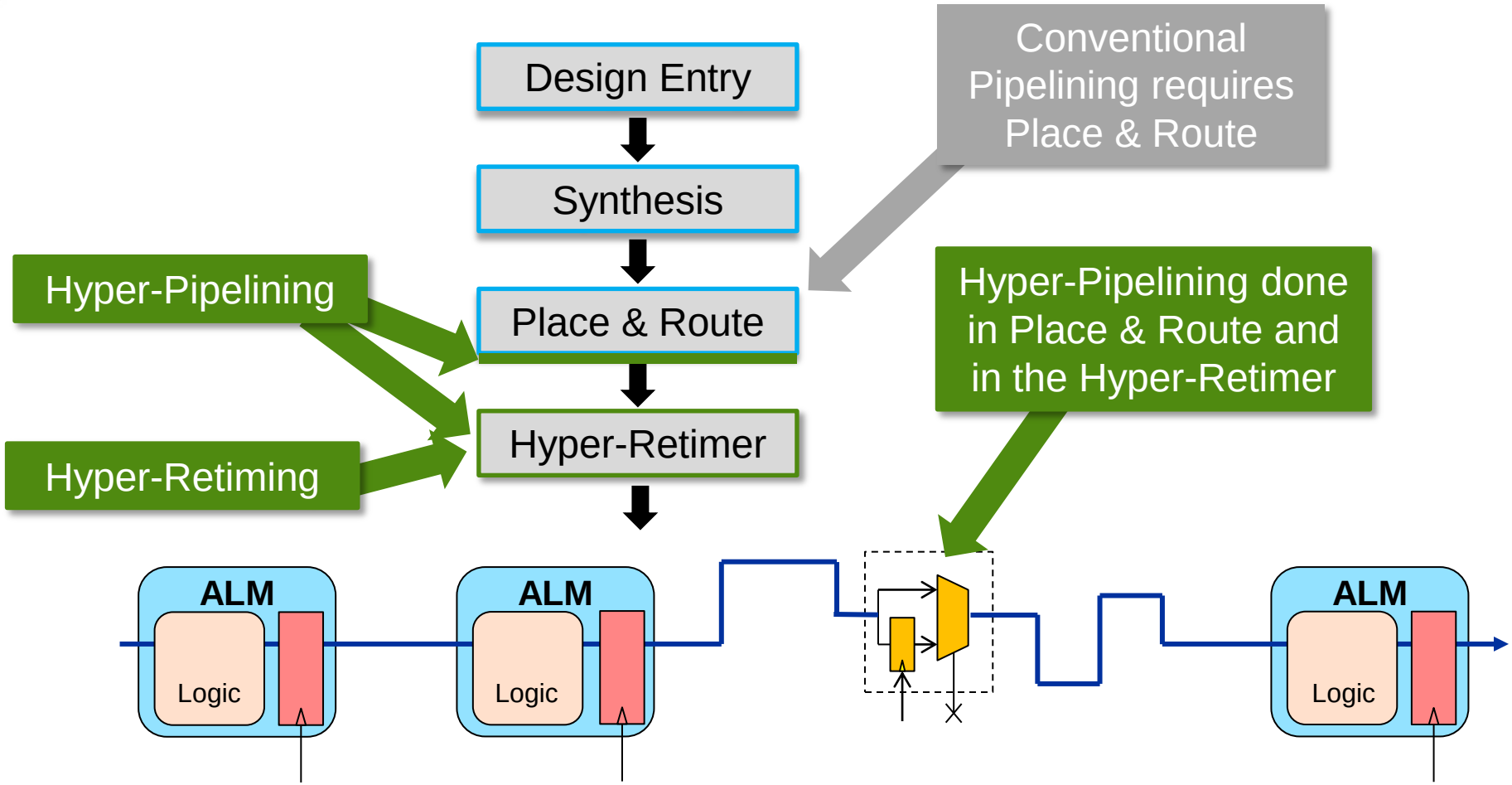


Hyper-Aware Design Flow

Hyper-Retiming Design Flow



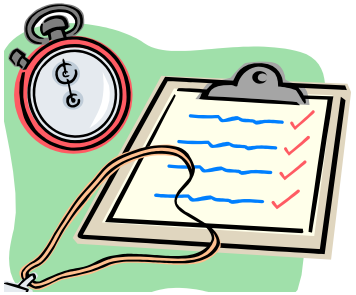
Hyper-Pipelining Design Flow



Stratix 10 Early Access Software

What's Available in Stratix 10 Early Access Software

- Stratix 10 Early Access SW Capabilities
  - ***Performance estimation* tool** for Stratix 10 core fmax
  - ***Design Performance exploration* tool** with HyperFlex
    - Hyper-Retiming
    - Hyper-Pipelining
    - Hyper-Optimization
  - Based on an Arria 10 baseline design & Fast Forward Compile
    - Requires porting the design to Arria 10



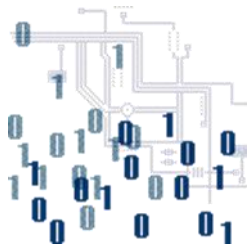
- Not a full Quartus Release for Stratix 10
  - No EMIF Support
  - No Transceiver Support
  - No Pin Assignments
  - No Device Selection
- Available Today through your FAE
  - Requires additional free license on top of valid Quartus II license

Stratix 10 Early Access Software Flow

```
module binary_adder_tree (a,
   parameter width = 16;
   input [width-1:0] a, b, c,
   input clk;
   output [width-1:0] out;

   wire [width-1:0] sum1, sum
   reg [width-1:0] sumreg1, s
   // Registers

   always @ (posedge CLK)
      begin
         sumreg1 <= sum1;
         sumreg2 <= sum2;
         sumreg3 <= sum3;
         sumreg4 <= sum4;
      end
```



Fast Forward

Compile

Stratix® 10

FPGA ▪ SoC

Design Entry (RTL)

Standard Arria 10 Compile

Fast Forward Compile

Report

Predicted Stratix 10 Fmax

+

Report how to increase performance further

HyperFlex™ ARCHITECTURE

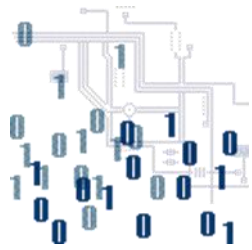
ALTERA.

Stratix 10 Early Access Software Flow

```
module binary_adder_tree (a,
   parameter width = 16;
   input [width-1:0] a, b, c,
   input clk;
   output [width-1:0] out;

   wire [width-1:0] sum1, sum
   reg [width-1:0] sumreg1, s
   // Registers

   always @ (posedge CLK)
      begin
         sumreg1 <= sum1;
         sumreg2 <= sum2;
         sumreg3 <= sum3;
         sumreg4 <= sum4;
      end
```



Stratix® 10
FPGA ▪ SoC

Design Entry (RTL)

Standard Arria 10 Compile

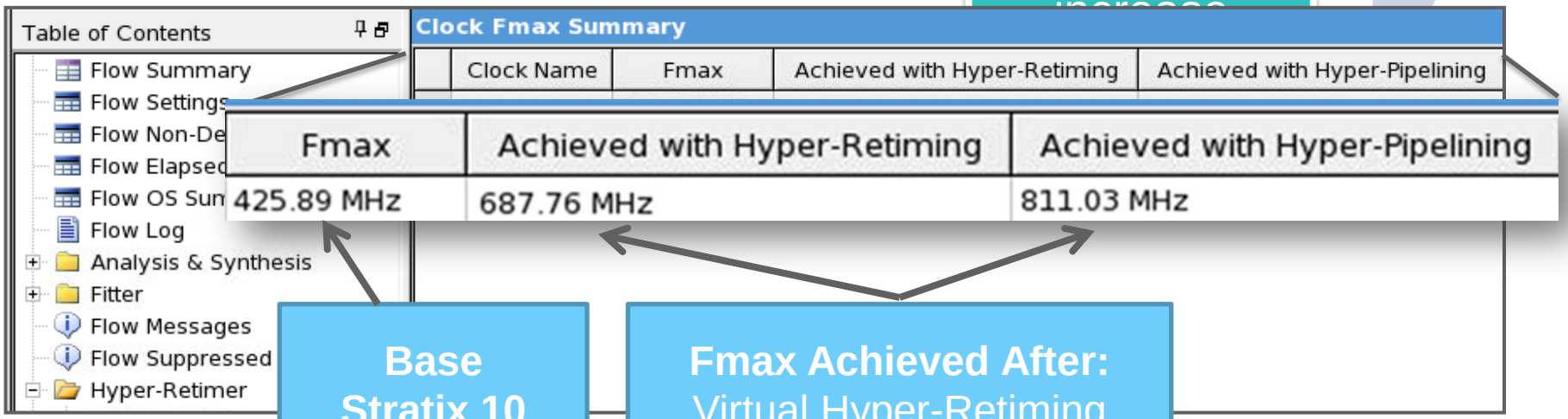
Fast Forward Compile

Report

Predicted Stratix 10 Fmax

+

Report how to

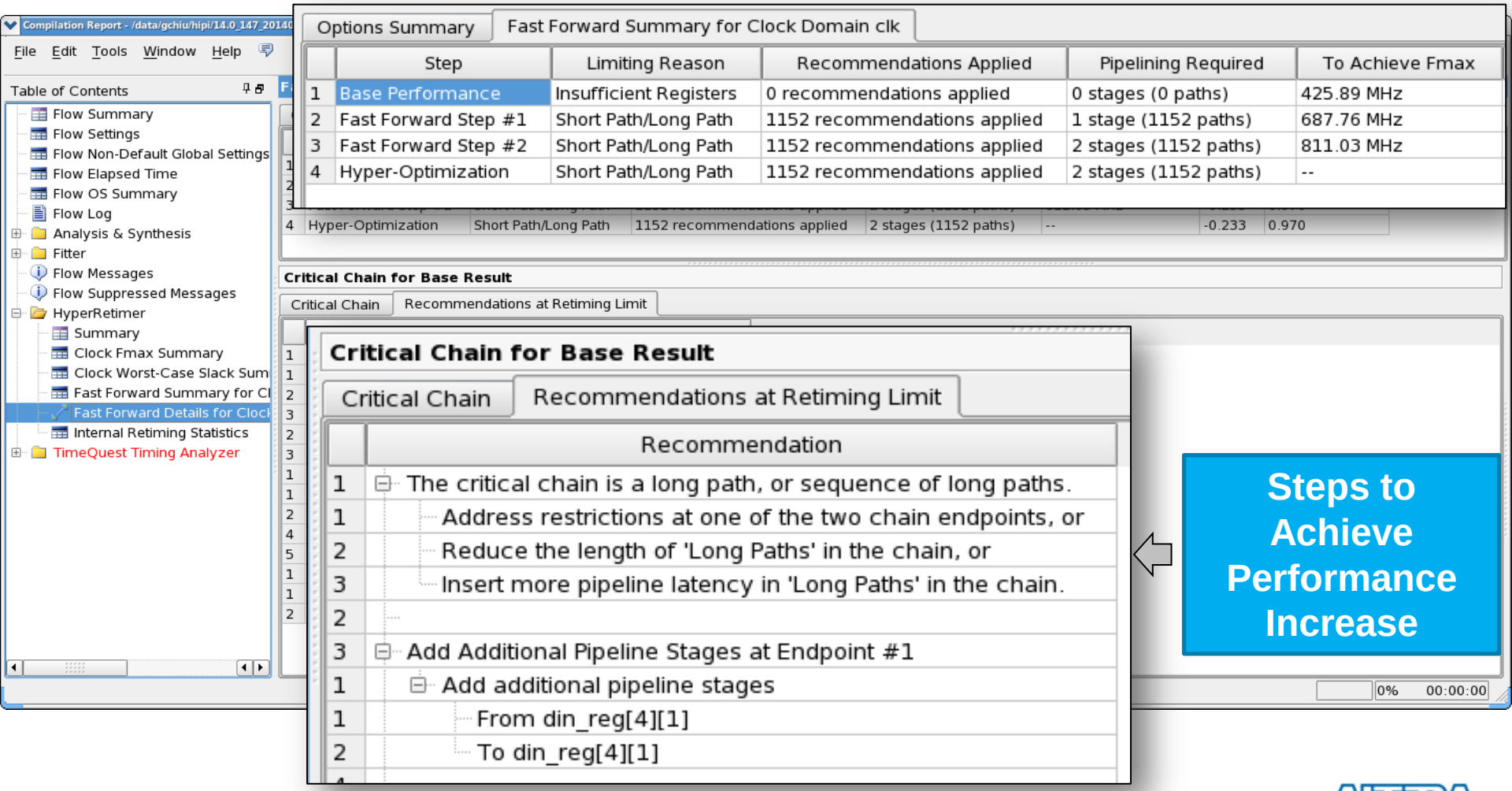


Base Stratix 10 Fmax

Fmax Achieved After:
Virtual Hyper-Retiming
Virtual Hyper-Pipelining

Step-by-Step Guidance to Increase Performance

Performance Improvement Recommendations



Steps to Achieve Performance Increase

Fast-Forward Compile – Performance Exploration

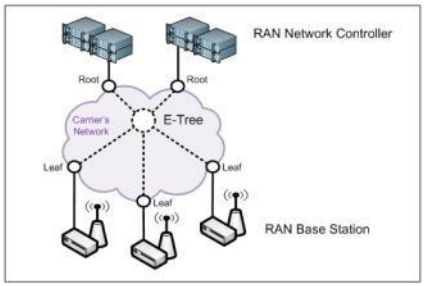
- Analyzes how fast a design can go with the HyperFlex Architecture
- How it works?
  - Virtually Implements Hyper-Retiming
  - Virtually Implements Hyper-Pipelining
  - Identifies bottlenecks for Hyper-Optimization
- Fast-Forward Compile generates an accurate “what-if” Fmax
  - Provides detailed guidance to maximize performance using Hyper-Retiming and Hyper-Pipelining
  - Identifies critical paths for further Hyper-Optimization

Available Now!
Stratix 10
Early Access
Software

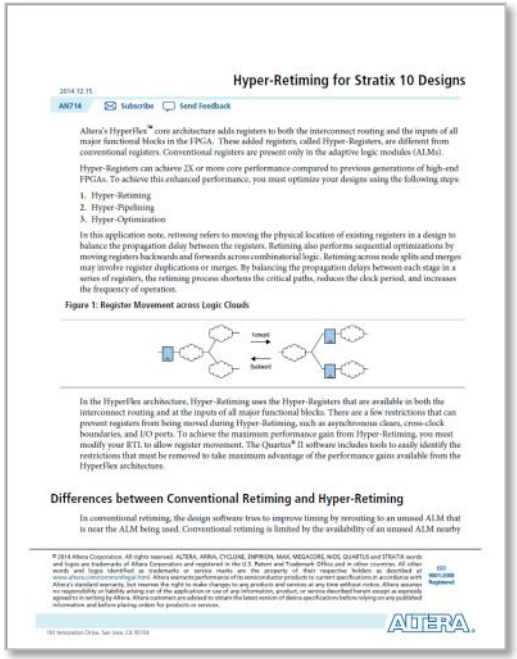
HyperFlex Performance Benchmarks

Benchmark Results From Real Designs

Benchmark	Data Path	Control Logic	Co-Processor
Design Target	> 700 MHz	> 550 MHz	300 MHz
Baseline	302 MHz (1X)	132 MHz (1X)	156 MHz (1X)
+ Hyper-Retiming	426 MHz (1.4X)	185 MHz (1.4X)	205 MHz (1.3X)
+ Hyper-Pipelining	518 MHz (1.7X)	276 MHz (2.1X)	305 MHz (1.96X)
+ Hyper-Optimization	745 MHz (2.4X)	623 MHz (4.7X)	Not required

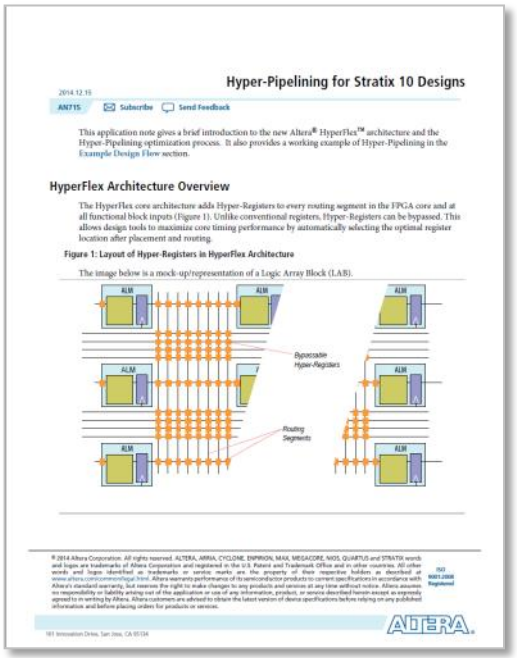


Stratix 10 HyperFlex Application Notes



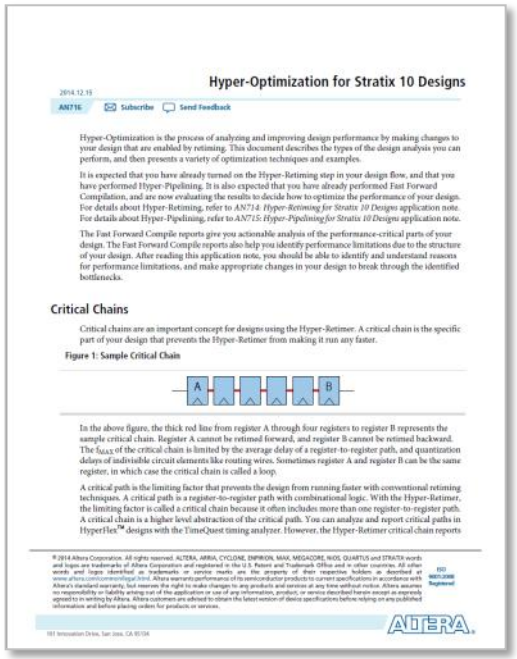
AN714

- **Hyper-Retiming** for Stratix 10 Designs
- How it works
- Understand what to do with register controls (async and sync clears, clock enables)



AN715

- **Hyper-Pipelining** for Stratix 10 Designs
- How it works
- Understand how to use pipelining to optimize speed of your design
- Example design flow



AN716

- **Hyper-Optimization** for Stratix 10 Designs
- Interpret fast forward report file
- Understand bottle-necks limiting further performance gains
- Know what to do about them

Thank You