Synthesis Options

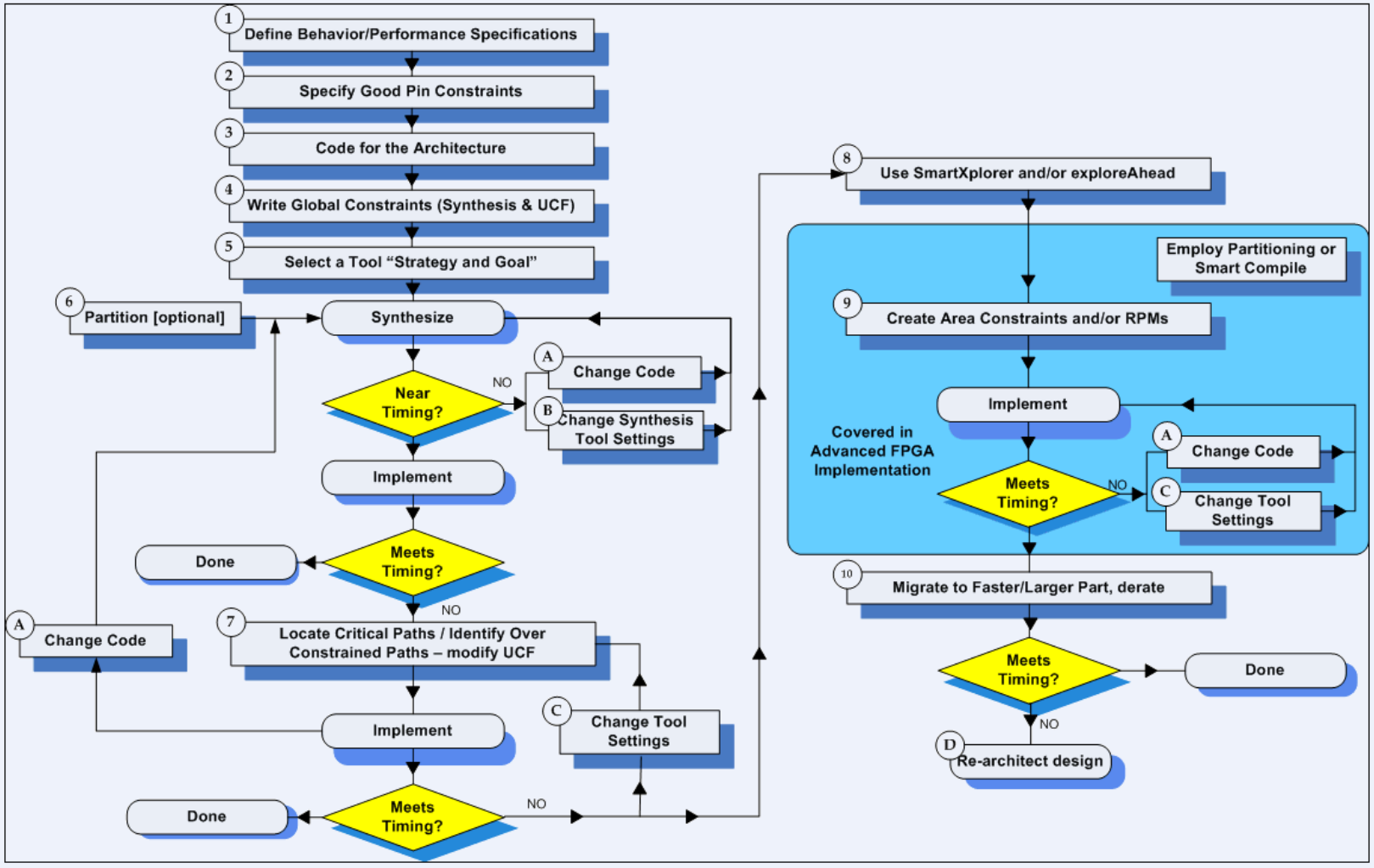
Welcome

- If you are new to FPGA design, this module will help you synthesize your design properly
- These synthesis techniques promote fast and efficient FPGA design development
- Tips for all major FPGA synthesis tools are included in this module

After completing this module, you will able to:

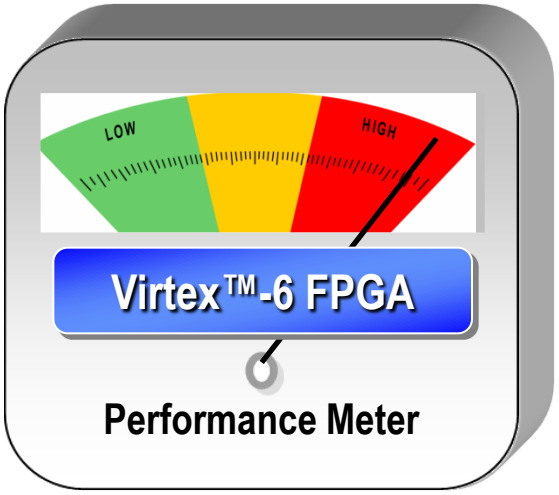
- Identify synthesis tool options that can be used to increase performance and/or reduce your design size
- Describe an approach to using your synthesis tool to obtain higher performance

Timing Closure



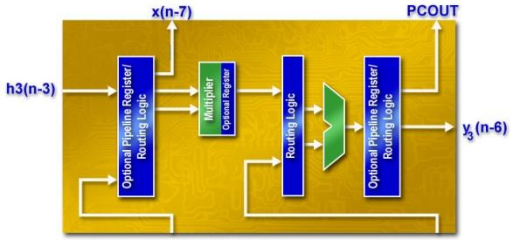
Breakthrough Performance

- Three steps to achieve breakthrough performance
  - 1. Utilize embedded (dedicated) resources
    - Performance by construction
    - DSP slice, block RAM, ISERDES, OSERDES, EMAC, and MGT
  - 2. Write code for performance
    - Use synchronous design methodology
    - Ensure the code is written optimally for critical paths
    - Pipeline
  - 3. Drive your synthesis and Place & Route tools
    - Try different synthesis optimization techniques
    - Add critical timing constraints in synthesis
    - Preserve hierarchy
    - Apply full and correct constraints
    - Use High effort

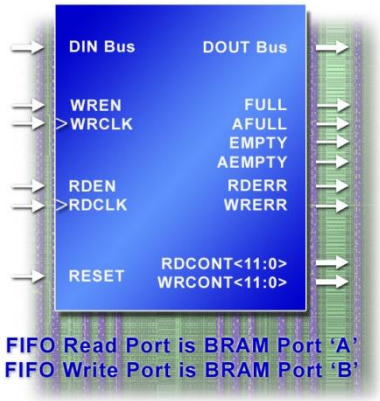


Use Dedicated Hardware

- Dedicated hardware block timing is *correct by construction*
  - Not dependent on programmable routing
- Offers as much as 3x the performance of soft implementations
- Examples
  - FIFO at 600 MHz
  - DSP slices at 600 MHz
  - Block RAM at 600 MHz



XtremeDSP™ Solution Slice



Block RAM/FIFO

Simple Coding Techniques

- Use pipeline stages—more bandwidth
- Use synchronous reset—better system control
- Use Finite State Machine (FSM) optimizations
- Use inferable resources
  - Multiplexer
  - Shift Register LUT (SRL)
  - Block RAM, LUT RAM
  - Cascade DSP
- Think about the levels of logic required for the logic you are building
  - Be aware of the circuit structures being inferred
  - Pay attention to the expected combinatorial complexity

See the *Synthesis and Simulation Design Guide*:
Help → Software Manuals → Synthesis and Simulation Design Guide

Synthesis Options

- There are many synthesis options that can help you obtain your performance and area objectives
  - Timing-driven synthesis
  - FSM extraction
  - Retiming
  - Register duplication
  - Hierarchy management
  - Resource sharing
  - Physical optimization
- Note that these options are included with Synplify, Precision, and XST synthesis tools
  - The notes include instructions for each tool

Synthesis Guidelines

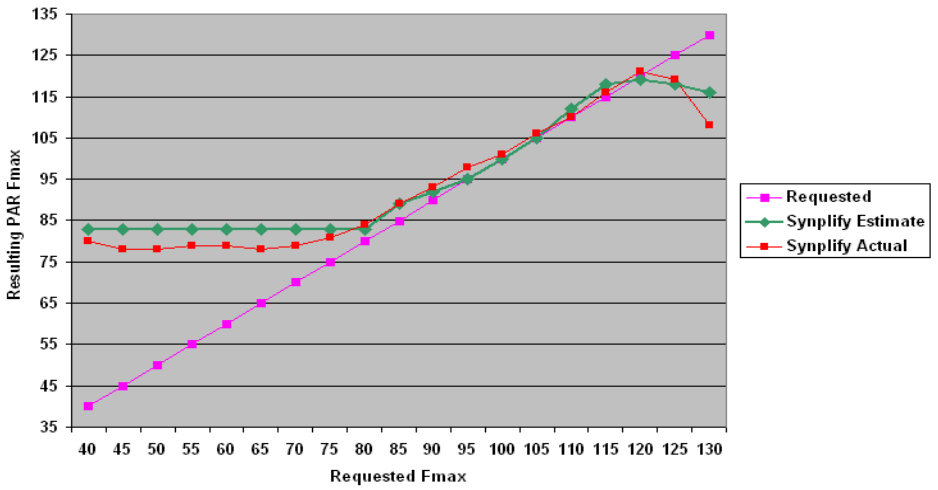
- Use timing constraints to drive the optimization of your design
  - Define accurate individual clock constraints
  - Create clock constraints in the appropriate style
    - Specify related clocks using related constraints
    - Specify unrelated clocks using independent constraints
      - Use different clock groups in Synplicity
  - Based on your performance objectives, the tools will try several algorithms to attempt to meet performance while keeping the amount of resources in mind
  - Performance objectives are provided to the synthesis tool via timing constraints
  - Do not over-constrain your design
    - This will disable your synthesis tool from helping you

Timing Constraints

- Apply proper timing constraints to the synthesis tool, but do not pass them to the implementation tools
  - Synthesis constraints will also be passed (by default) on to the Xilinx implementation tools via a Netlist Constraints File (NCF) when using Synplify
    - This should be turned off
  - Synthesis constraints can be passed (not by default) on to the Xilinx implementation tools via the Xilinx NGC file when using XST
- Synplify
  - Specify constraints in the SDC file or use the SCOPE GUI
- XST
  - Specify constraints in the XCF file
    - See the *Synthesis Constraints* section of Chapter 3 in the Constraints Guide
      - Software Manuals: **Help → Software Manuals → Constraints Guide**

Timing Constraint Example

- Use constraints
  - Synplify stops optimizing when the constraints are met
- Over-constraining clocks can yield poorer results
  - Over-constraining means specifying a constraint that is tighter than what your system needs
- Using the global frequency field can deteriorate results



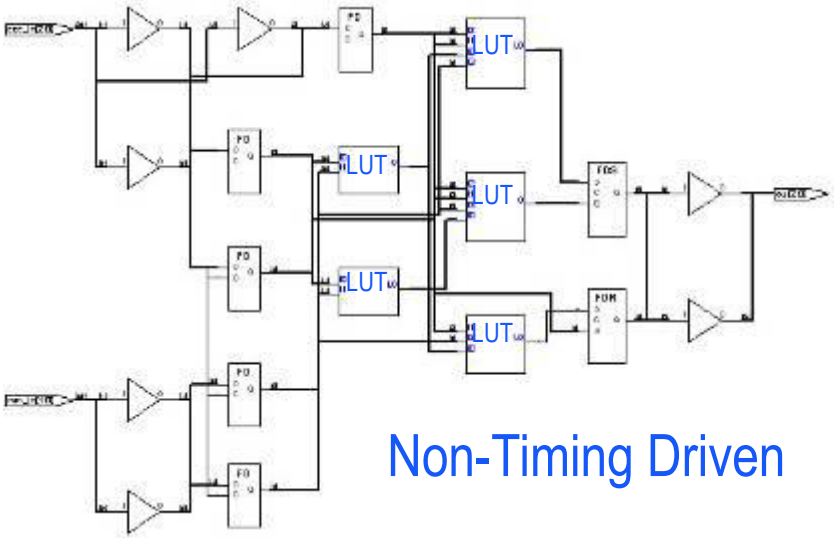
(*) Synplicity's data

	Enabled	Clock	Frequency (MHz)	Period (ns)	Clock Group	Rise At (ns)	Fall At (ns)	Duty Cycle (%)	Route (ns)	Virtual Clock
1	☑	clock1	285.714	**3.5**	group1			50	1.2	☐
2	☑	clock2	232.558	**4.3**	group2			50	0.5	☐
3	☑	clock3	178.571	**5.6**	group3			50	0	☐
4	☑									☐

Clocks | Clock to Clock | Inputs/Outputs | Registers | Multi-Cycle Paths | False Paths | Max Delay Paths | Attributes | Compile Points | Other

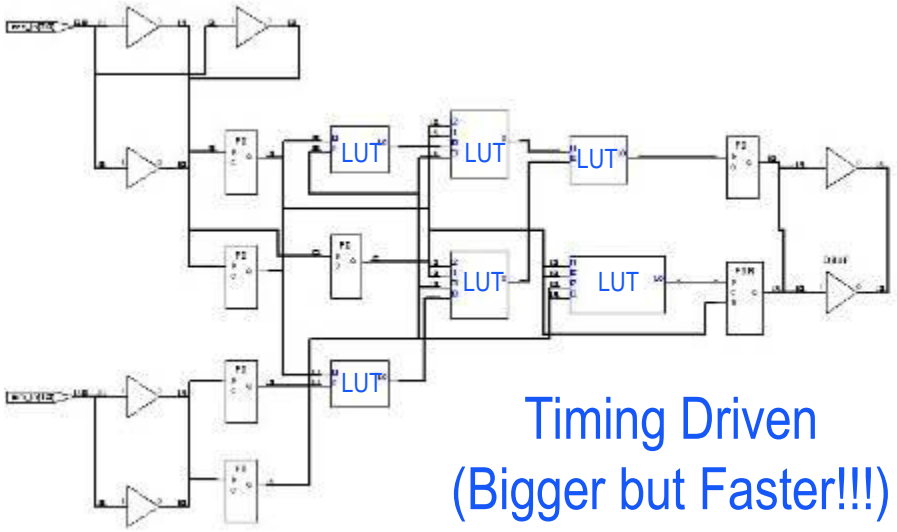
Impact of Synthesis Constraints

- Non-timing-constrained designs can be optimized for area rather than performance



Non-Timing Driven

Total LUTs: 5
Clock Freq: 423.7 MHz



Timing Driven
(Bigger but Faster!!!)

Total LUTs: 6
Clock Freq: 591.7 MHz (+ 40%)

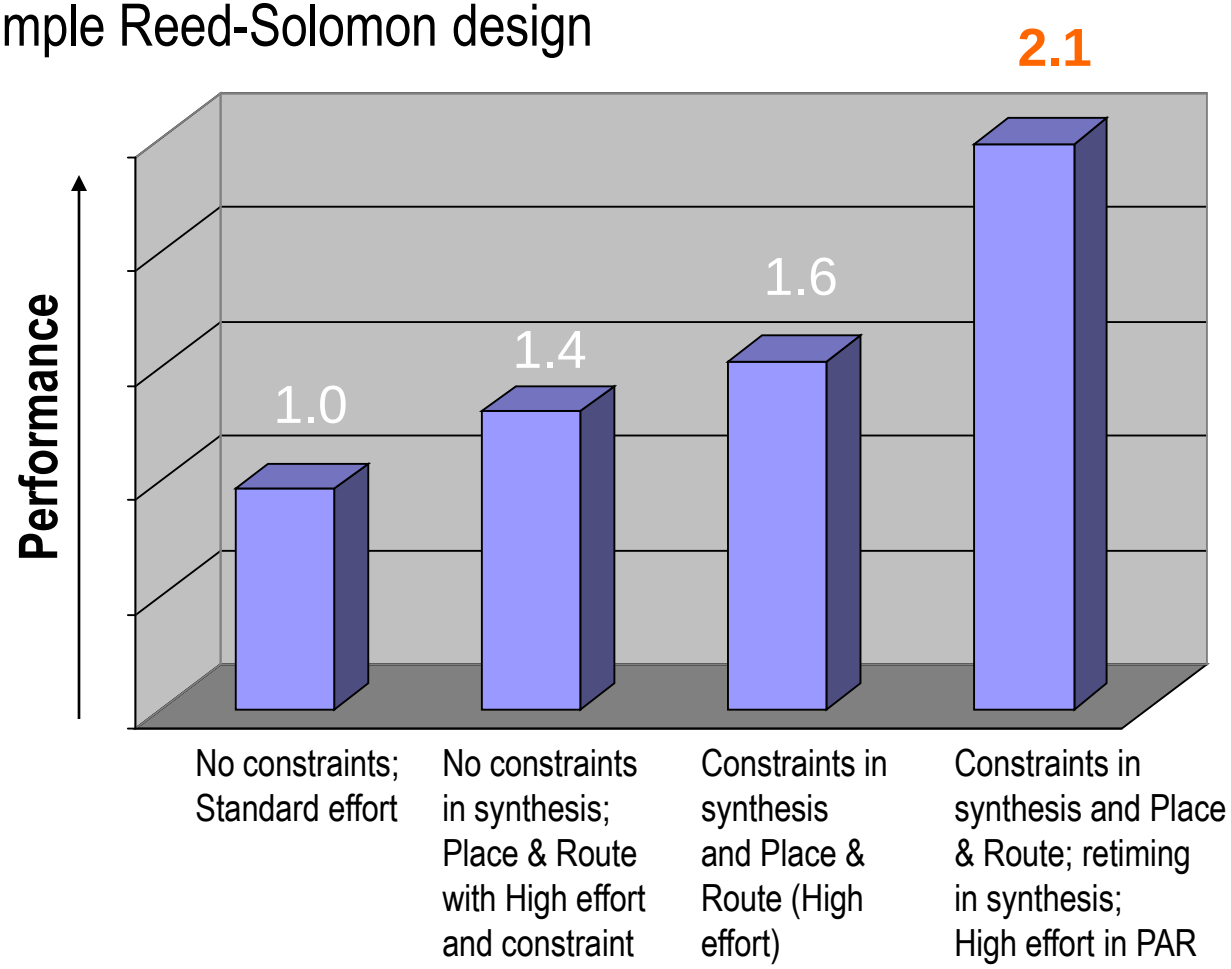
Place & Route Guidelines

- Timing constraints
  - It is essential to use accurate constraints for the implementation tools
- Implementation tool options
  - The implementation tools have many options that can affect design performance
- Area Constraints
  - Especially with the use of the PlanAhead™ tool

Using the correct Place & Route options can have a dramatic impact on design performance

Impact of Constraints in Tools

- Example Reed-Solomon design

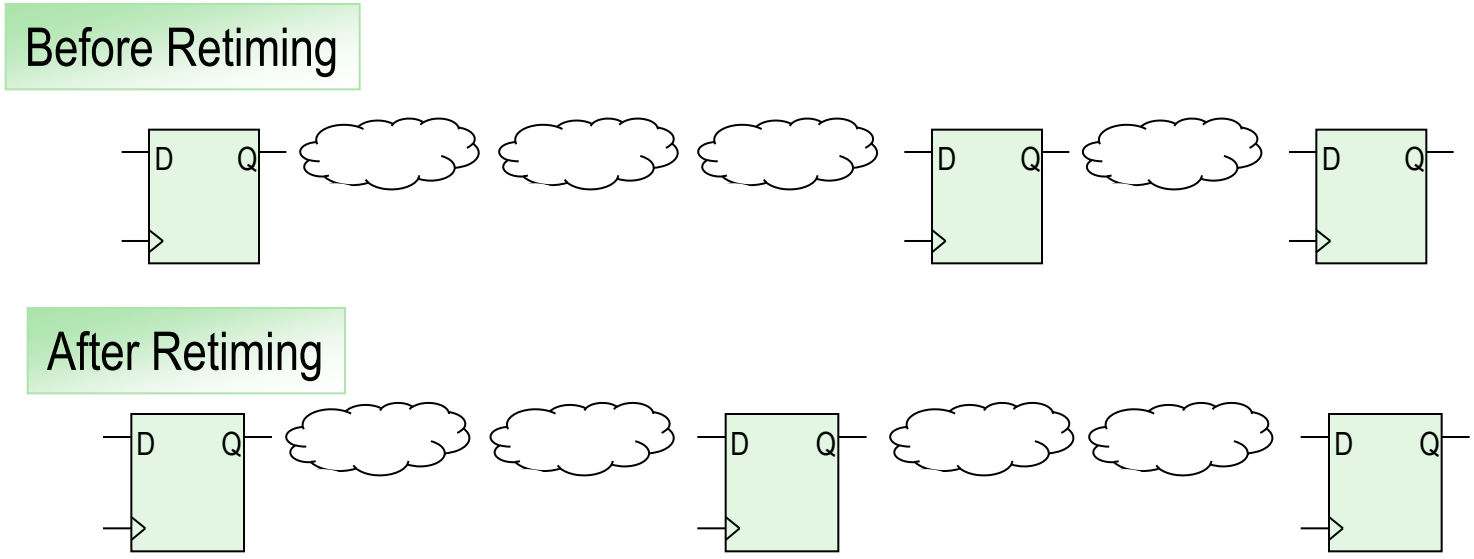


FSM Extraction

- Finite State Machine (FSM) extraction optimizes your state machine by re-encoding and optimizing your design based on the number of states and inputs
  - By default, the tools will use FSM extraction
  - Can be enabled or disabled globally, or using attributes in your HDL code
- Safe state machines
  - By default, the synthesis tools will remove all decoding for illegal states (when FSM extraction is enabled)
    - Even if you include VHDL “when others” or Verilog “default” cases
  - Must be turned ON to use “safe” FSM implementation
    - See Notes for more information

Retiming

- Retiming: The synthesis tool automatically tries to move register stages to balance combinatorial delay on each side of the registers



Register Duplication

- Register duplication is used to reduce fanout on registers (to improve delays)
- Registered output signals that are used internally
- Xilinx recommends manual register duplication
  - Not all high fanout nets will give you a timing problem
  - Most synthesis vendors create signals <*signal_name*>*_rep0*, *_rep1*, etc.
    - Implementation tools pack logic with related names into the same slice, which can prohibit a register from being moved closer to its destination
  - When manually duplicating registers, do not use a number at the end
    - Example: <*signal_name*>*_0dup*, <*signal_name*>*_1dup*
  - Use synthesis options to prevent duplicate registers from being re-merged

Hierarchy Management

- The basic settings are
  - Flatten the design: Allows total combinatorial optimization across all boundaries (XST default)
  - Maintain hierarchy: Preserves hierarchy without allowing optimization of combinatorial logic across boundaries (Xilinx recommended)
- If you have followed the synchronous design guidelines, use the setting *-maintain hierarchy*
- If you have not followed the synchronous design guidelines, use the setting *-flatten the design*
- Your synthesis tool may have additional settings
  - Refer to your synthesis documentation for details on these settings

Hierarchy Preservation Benefits

- Easily locate problems in the code based on the hierarchical instance names contained within static timing analysis reports
- Enables floorplanning and incremental design flow
- The primary advantage of flattening is to optimize combinatorial logic across hierarchical boundaries
  - If the outputs of leaf-level blocks are registered, there is generally no need to flatten
    - However, preserving hierarchy can limit register retiming (balancing) and register duplication

Resource Sharing

- Resource sharing allows arithmetic operator resources to be shared with other functions
  - By default, this property is set to True with XST
- If your design has a significant amount of math functions, it can decrease the size of your design
  - Resource sharing is the opposite of logic replication
  - This option can increase the net delays for those nets whose fanout increases

Schematic Viewers

- Allows you to view synthesis results graphically
  - Check the number of logic levels between flip-flops
  - Locate net and instance names quickly
  - View the design as generic RTL or technology-specific components
- Works best when hierarchy has been preserved during synthesis

Cross-Probing

- From the Timing Analyzer, click a reported worst-case path and that path will be highlighted in the synthesis schematic viewer
  - Cross-probe to the code
    - Review the code to determine whether or not it can be rewritten to improve performance
    - Apply timing constraints in your synthesis tool to optimize this path better
  - You may need to set some environment variables for this to work
    - For more information, see Application Note XAPP406: *Cross-Probing to Synplify and Exemplar*

Physical Optimization

- *Synopsys Synplify Premier* or *Mentor Precision Physical* software (add-on tools)
- Based on the critical paths in the design, the tools will attempt to optimize and physically locate the associated logic closely together to minimize the routing delays
- Essentially, this is a way to provide critical path information to the synthesis tool so that it can attempt to optimize those paths further

Summary

- Your HDL coding style can affect synthesis results
- Infer resources whenever possible
- Most resources are inferable, either directly or with an attribute and the appropriate coding style
- If you cannot infer the resource you need, instantiate the necessary component with the aid of the Core Generator
- Take advantage of the synthesis options provided to help you meet your timing objectives
- Use synchronous design techniques and timing-driven synthesis to achieve higher performance

Where Can I Learn More?

- ***Software Manuals***
  - **Start → Xilinx ISE Design Suite 12.1 → ISE Design Tools → Documentation → Software Manuals**
  - This includes the *Synthesis & Simulation Design Guide*
    - This guide has example inferences of many architectural resources
  - *XST User Guide*
    - HDL language constructs, coding recommendations, and synthesis options
  - *Constraints Guide*
    - All Synthesis and Implementation constraints
- **Xilinx Training**
  - **www.xilinx.com/training**
    - Xilinx tools and architecture courses
    - Hardware description language courses
    - Basic HDL Coding Techniques, Spartan-6 and Virtex-6 Coding Techniques and other Free training videos!

Recommended REL Modules

- Additional FREE training videos are available for you to improve your HDL coding style
  - *Basic HDL Coding Techniques, part 1 and 2*
    - Design guidelines (good design practices)
    - Best ways to pipeline your design and Finite State Machine design
  - *Virtex-6 and Spartan-6 HDL Coding Techniques, part 1 and 2*
    - Coding for hardware resources
      - SRL, multiplexers, carry logic, and GSR
    - Coding to reduce your design size and improve your speed
      - Managing your control signals (sets, resets, clocks, clock enables)
      - Block RAM and DSP slice
  - *XST Synthesis Options*
    - Detailed instruction on how to use XST for synthesis

Trademark Information

Xilinx is disclosing this Document and Intellectual Propery (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS" WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.