Targeting Xilinx FPGAs

Objectives

After completing this module, you will be able to:

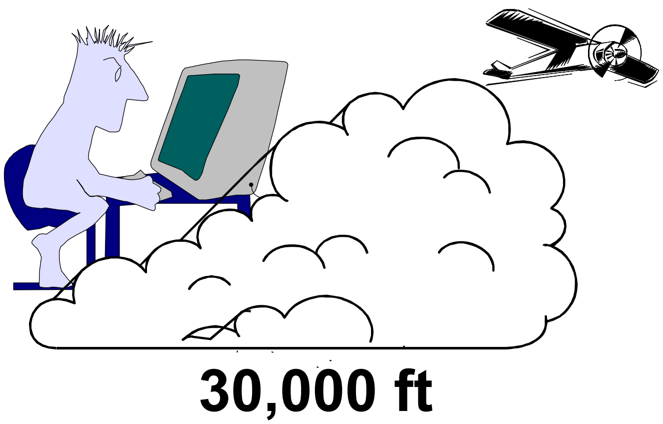
- Select a proper coding style to create efficient FPGA designs
- Specify Xilinx resources that need to be instantiated for various FPGA synthesis tools
- Identify synthesis tool options that can be used to increase performance
- Describe an approach to using your synthesis tool to obtain higher performance
- Describe unique coding requirements for optimal Virtex™-5 FPGA implementation

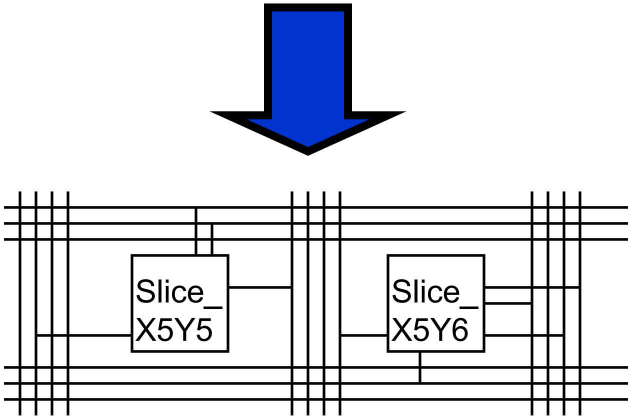
Outline

- **Recognizing the FPGA Challenge**
- Timing and Performance Guidelines
- Inference versus Instantiation
- Synthesis Tools and Options
- Virtex-5 FPGA Power and Verification Considerations
- Summary

High-Level Code versus Device Level Optimization

- HDL design entry allows high-level hardware modeling, without device-level considerations
- Nonetheless, the need for performance and device-level optimization still exists





XILINX®

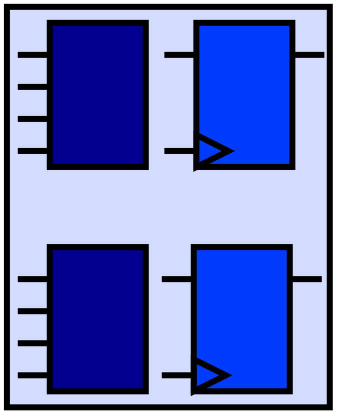
Device-Level Challenges

- Optimization within Xilinx devices via VHDL synthesis requires that you address the following areas
  - 1. Effective use of I/O resources
  - 2. Using DCM, BUFG, and BUFGMUX
  - 3. Using SRLE, the SelectIO™ technology standard, and MGT
  - 4. Block RAM and block MULT
  - 5. DSP48E and ISERDES/OSERDES
  - 6. DDR and IDELAY
  - 7. Accessing other dedicated resources

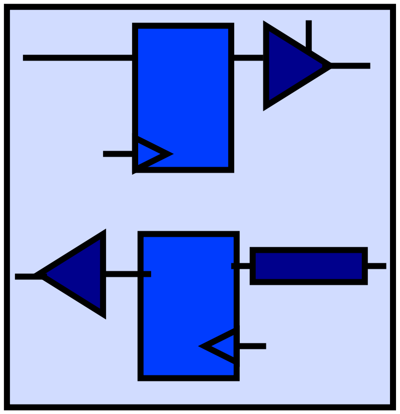
- Your success in addressing these issues depends largely on your coding style and your choice of VHDL synthesis tools

Xilinx Logic Implementation

- Within Xilinx devices, most logic is implemented in Input /Output Blocks (IOBs) or Configurable Logic Blocks (CLBs). CLBs contain dedicated registers and SRAM-based look-up tables for comb logic



SLICES and IOBs

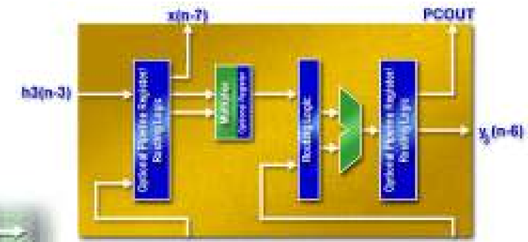


- Xilinx ISE™ software, and other synthesis tools that target Xilinx, use specific algorithms and directives optimized for the unique architecture

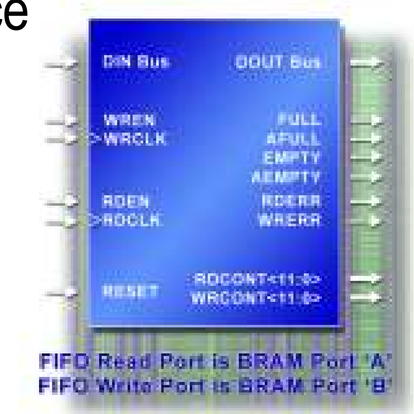
XILINX®

Use Embedded Blocks

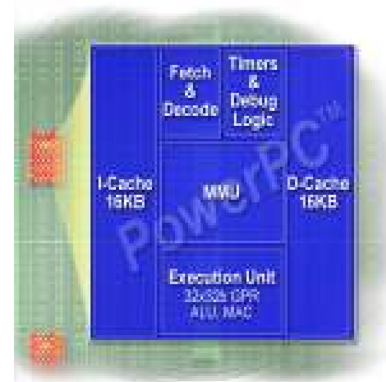
- Embedded block timing is *correct by construction*
  - Not dependent on programmable routing
- Offers as much as 3x the performance of soft implementations
- Examples
  - FIFO at 500 MHz
  - DSP slices at 500 MHz
  - PowerPC® processor at 702 DMIPS



XtremeDSP™ Technology Slice



Smart RAM FIFO



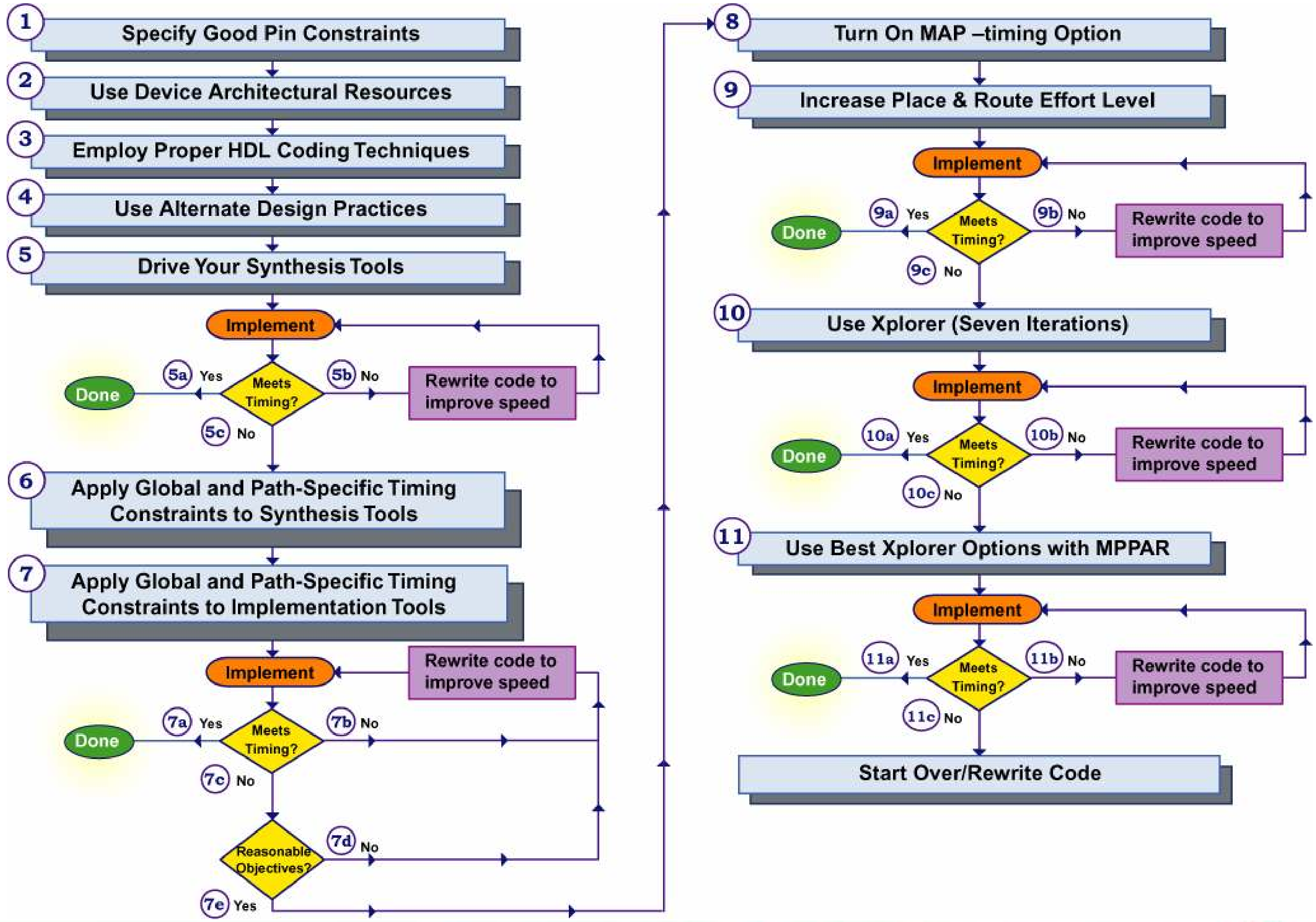
PowerPC Processor

Outline

- Recognizing the FPGA Challenge
- **Timing and Performance Guidelines**
- Inference versus Instantiation
- Synthesis Tools and Options
- Virtex-5 FPGA Power and Verification Considerations
- Summary

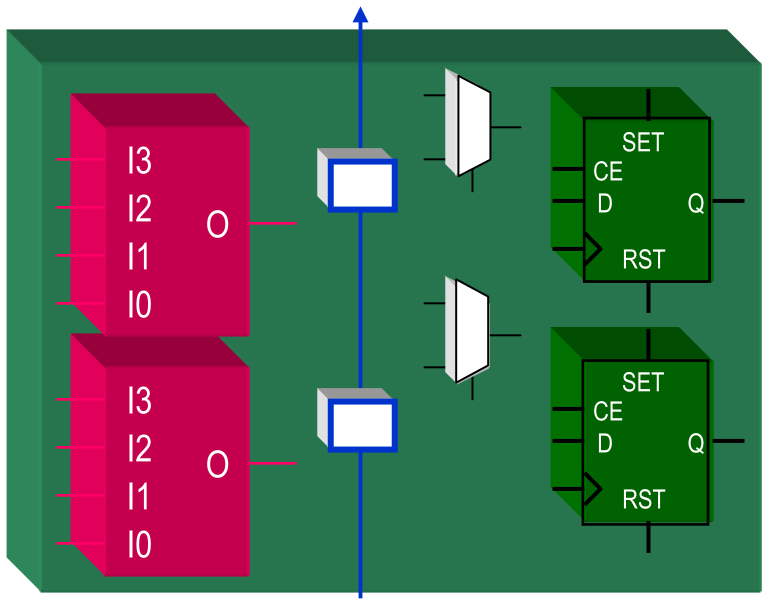
XILINX®

Timing Closure



XILINX

500-MHz Fabric Guidelines



One Level of Logic Only

- For the fabric to achieve maximum performance, note the following important considerations
  - 1. Do not exceed more than one level of logic. Use the existing registers
  - 2. Carry chains should not exceed 14 before being registered*
  - 3. You may need placement constraints to keep functions together

Simple Coding Steps Yield 3x Performance

- 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
- Avoid high-level constructs (i.e., loops)—may yield slow implementations
- Avoid deeply nested *if-then-else* statements
  - Usually parallel; however, deep nesting can result in priority encoded logic
- Use *case* statements for large decoding
  - Rather *than if-then-else*

See the *Synthesis and Simulation Design Guide*:
toolbox.xilinx.com/docsan/xilinx9/books/docs/sim/sim.pdf

Targeting Xilinx Resources

- Some resources must be instantiated as a library primitive or an IP core (Architecture Wizard and CORE Generator™ software)
  - FIFO16, ISERDES, OSERDES, and clock resources
- Certain resources require specific coding
  - DSP48 registers have synchronous set/reset only
  - Distributed RAM/ROM and SRL do not have set/reset functionality after configuration
- Synchronous resets are preferred over asynchronous reset
  - Xilinx FPGAs have a configuration reset (GSR) that is used during the configuration stage to bring up the FPGA in a known state
    - GSR is also accessible to designers after configuration via the STARTUP block
    - No need for asynchronous initialization power-on reset

XILINX®

Synchronous Design Techniques

- Minimize the number of clocks in the design
- Make sure that internally created resets are synchronous
- Use only one edge of the clock
- Use edge-triggered flip-flops (avoid latches)
- Cross-clock domains via synchronization circuits
- Register top-level inputs and outputs for fastest performance and increased pin-locking capability
- Register leaf-level outputs
- Use hierarchy to separate functionality and clock domains
- Employ pipelining for critical paths
- Comment your code to draw attention to multicycle paths and critical paths

Using Resets

- Resets can affect DSP48 register packing
  - DSP48Es only have synchronous resets and no presets
    - Use synchronous resets (or no reset at all) whenever possible
- Resets can affect RAM use
  - RAM output registers have a single synchronous set/reset
  - Ability to map ROMs, look-up tables, state machines, and other logic into unused block RAMs require either no reset or a synchronous reset
- SRL inference
  - Do not use a reset for shift register code unless absolutely necessary

Use resets sparingly and synchronous resets whenever possible

Always Use Signed Data

- A common mistake is to specify an unsigned data type and expect optimal usage of DSP48Es
- DSP48Es only compute signed data
  - You will lose a bit for unsigned types
  - Can result in more resources used than expected
  - Use the std_logic_signed package and a signed data type

Use signed types when using multipliers

State Machine Encoding

- Use enumerated types to define state vectors (VHDL)
  - Most synthesis tools have commands to extract and re-encode state machines described in this way
- Use one-hot encoding for high-performance state machines
  - Uses more registers, but simplifies next-state logic
  - Examine trade-offs: Gray and Johnson encoding styles can also improve performance
  - Refer to the documentation of your synthesis tool to determine how your synthesis tool chooses the default encoding scheme
- Register state machine outputs for higher performance

Knowledge Check

Knowledge Check

- Describe some simple coding techniques to improve performance. What is the benefit provided by each technique?

Answer

- Describe some simple coding techniques to improve performance. What is the benefit provided by each technique?
  - Use pipeline stages for more bandwidth
  - Use synchronous reset for better system control and optimization
  - Use Finite State Machine (FSM) optimizations for optimization and performance
  - Use inferable resources. Using hard blocks reduces area, improves performance, and decreases power
  - Avoid high-level constructs (i.e., loops), which can yield slow implementations
  - Avoid deeply nested *if-then-else* statements, which can yield slow implementations from priority-encoded logic
  - Use *case* statements for large decoding, which yield parallel implementation for higher performance

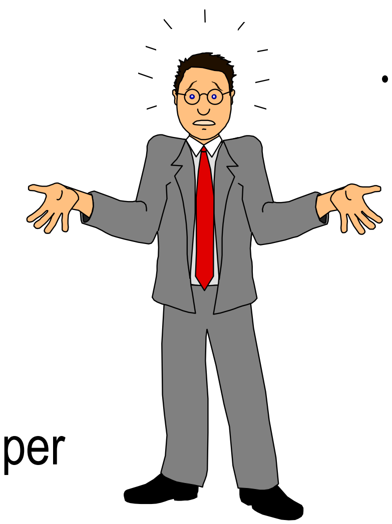
Outline

- Recognizing the FPGA Challenge
- Timing and Performance Guidelines
- **Inference versus Instantiation**
- Synthesis Tools and Options
- Virtex-5 FPGA Power and Verification Considerations
- Summary

Inference versus Instantiation

- The key to Xilinx optimization is accessing and controlling device-level resources, as well as overall place & route results
  - From an HDL perspective, there are only two means to access any resource

- **Inference**
  - Generic HDL code
    - Synthesis tool decides which vendor library to use
  - Device optimization as per tool ability
  - Maximum portability



- **Instantiation**
  - Create “instance” of
  - Designer references specific vendor macro
  - Maximum device optimization
  - May be required
  - Limits portability

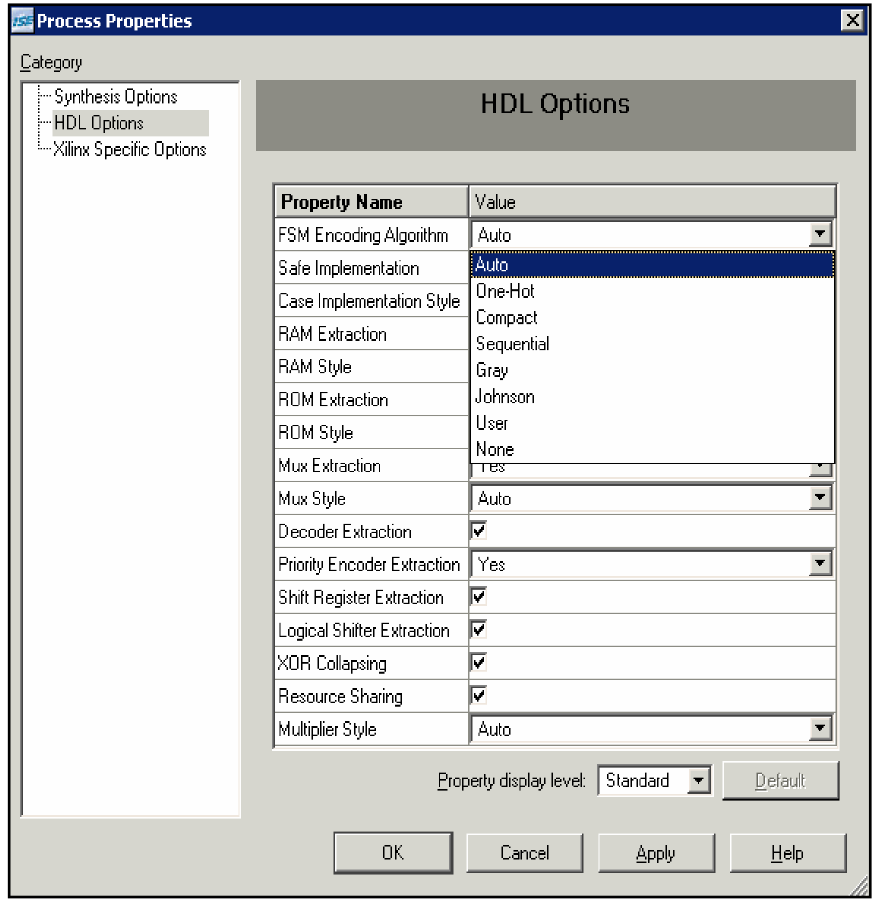
FPGA Resources

- Can be inferred by all synthesis tools
  - Shift register LUT (SRL16/ SRLC16)
  - F5, F6, F7, and F8 multiplexers
  - Carry logic
  - MULT_AND
  - Multipliers and counters using the DSP48
  - Global clock buffers (BUFG)
  - SelectIO™ (single-ended) standard
  - I/O registers (single data rate)
  - Input DDR registers
- Can be inferred by some synthesis tools
  - Memories
  - Global clock buffers (BUFGCE, BUFGMUX, BUFGDLL)
  - Some complex DSP functions
- Cannot be inferred by any synthesis tools
  - SelectIO (differential) standard
  - Output DDR registers
  - DCM/PMCD
  - Local clock buffers (BUFIO, BUFR)

XST Compiler Synthesis Directives

- Increasingly, the ability to infer certain resources also requires using a compiler-specific directive
  - This keeps the HDL source code generic and, thus, more portable. But results are not necessarily consistent across various tools

When using strategic menu options, add comments to the HDL source-code header; otherwise, reuse of the same module code may yield different results

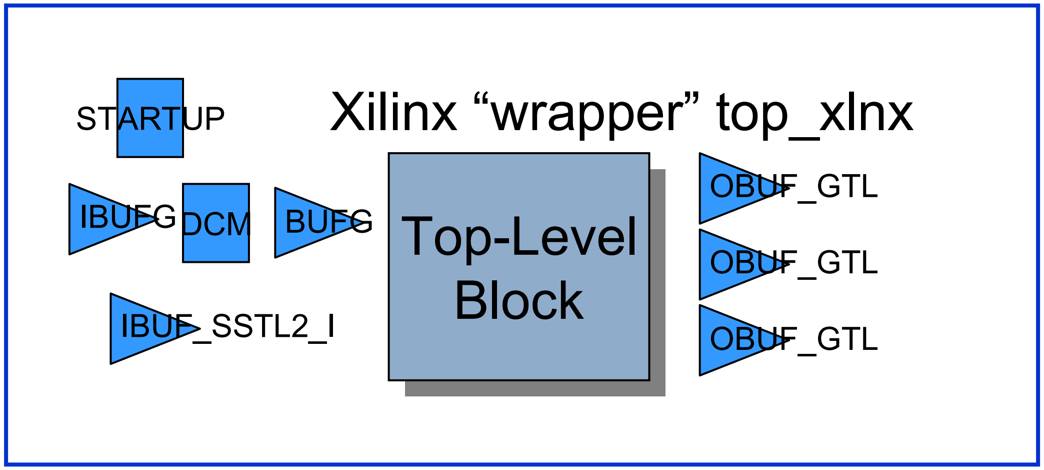


Suggested Instantiation

- Xilinx recommends that you instantiate the following elements
  - Memory resources
    - Block RAMs specifically (use the CORE Generator™ software system to build large memories)
  - SelectIO™ technology standard resources
  - Clocking resources
    - DCM, PMCD (use the Architecture Wizard)
    - IBUFG, BUFGMUX, BUFGCE
    - BUFIO, BUFR

Suggested Instantiation

- Why does Xilinx suggest this?
  - Easier to change (port) to other and newer technologies
  - Fewer synthesis constraints and attributes to pass on
    - Keeping most of the attributes and constraints in the Xilinx User Constraints File (UCF) keeps it simple—one file contains critical information



- Create a separate hierarchical block for instantiating these resources
  - Above the top-level block, create a Xilinx "wrapper" with instantiations specific to Xilinx

Knowledge Check

- In general, which Xilinx resources does Xilinx recommend be instantiated? Why?

Answer

- In general, which Xilinx resources does Xilinx recommend be instantiated?
  - Memory resources
    - Specifically, block RAMs (use the CORE Generator™ software to build large memories)
  - SelectIO™ technology standard resources
  - Clocking resources
    - DCM, PMCD (use the Architecture Wizard)
    - IBUFG, BUFGMUX, BUFGCE
    - BUFIO, BUFR
- Why?
  - Easier to change (port) to other and newer technologies
  - Fewer synthesis constraints and attributes to pass on

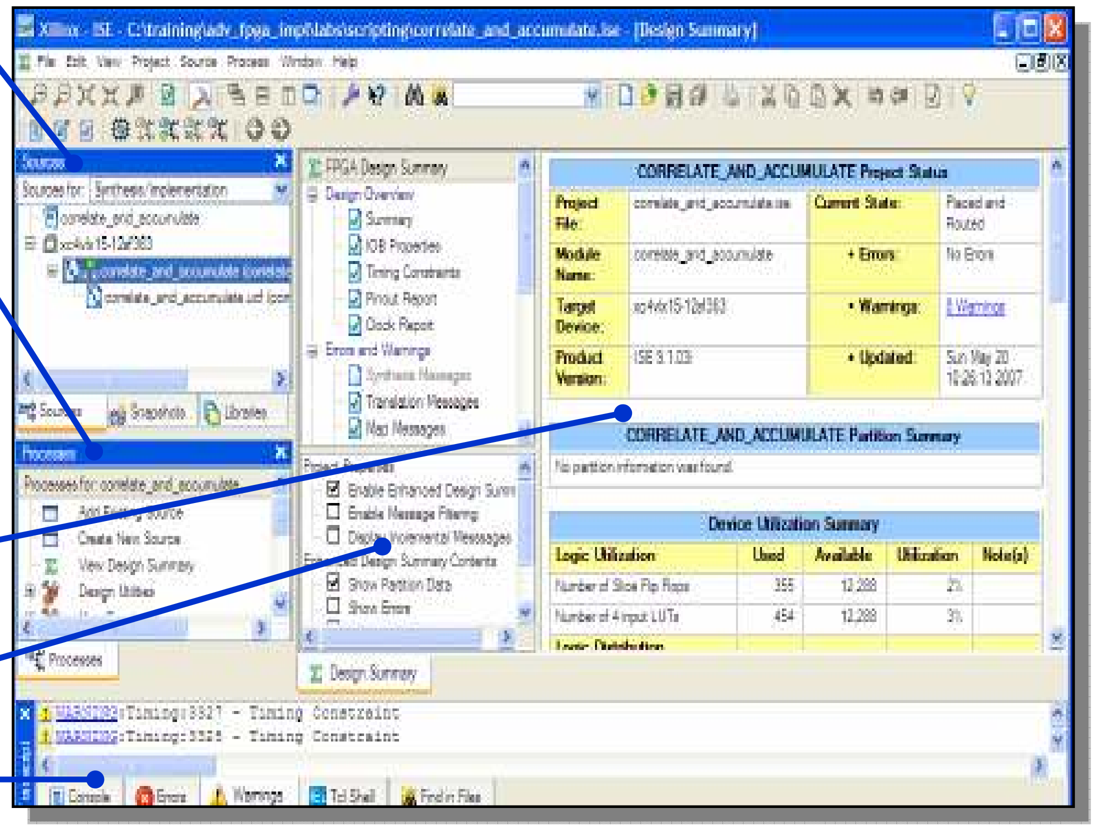
Outline

- Recognizing the FPGA Challenge
- Timing and Performance Guidelines
- Inference versus Instantiation
- **Synthesis Tools and Options**
- Virtex-5 FPGA Power and Verification Considerations
- Summary

XILINX®

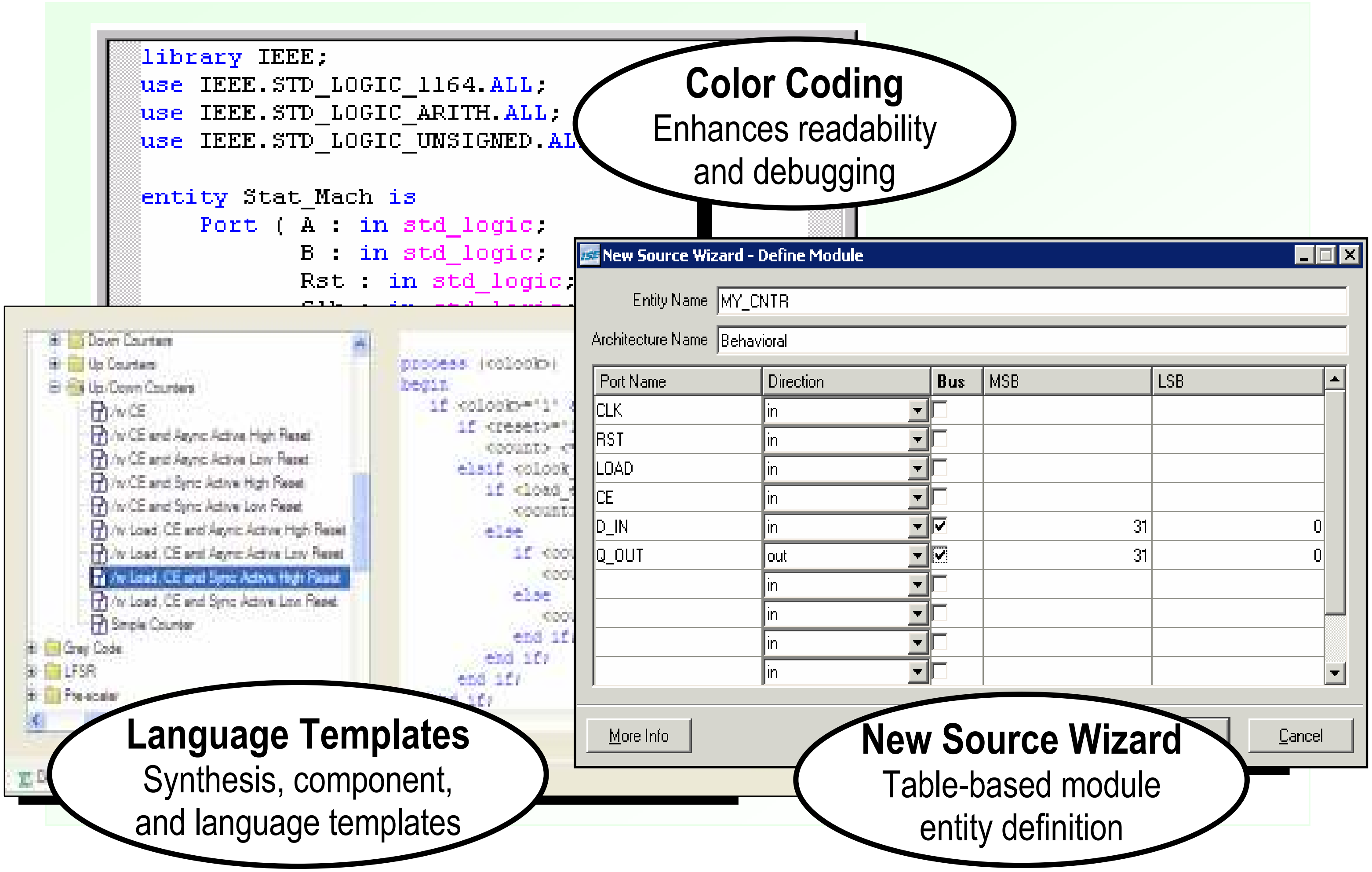
ISE 9 Software Project Navigator

- Sources window
  - Lists all source files
  - Shows design hierarchy
- Processes window
  - Shows current status and possible design flows for selected source file
- File editing area and Design Summary
- Project Properties
- Console window
  - Displays errors, warnings, and informational messages



XILINX®

ISE Software Editing Resources



The Alliance

- Xilinx recognizes the challenges and works directly with EDA companies such as Synopsys, Synplicity, Exemplar, Aldec, Model Technology, Viewlogic, and others
- The Alliance partners are a consortium of the top EDA, HDL synthesis, and simulation-tool providers
- In the process, Xilinx assists in creating efficient interfaces, translators, netlist optimizers, benchmarking standards, and qualitative objectives

Synthesis Options

- Many synthesis options can help you obtain your performance and area objectives
  - Timing-driven synthesis
  - FSM extraction
  - Retiming
  - Register duplication
  - Hierarchy management
  - Resource sharing
  - Physical optimization

Timing-Driven Synthesis

- Synplify, Precision, and XST software
- Timing-driven synthesis uses performance objectives to drive the optimization of the design
  - 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

Timing Constraints Editor

- Synplify and Precision software
- The timing constraints editor allows you to apply timing constraints for your tool
  - These constraints will be used to drive synthesis optimization (for those tools that use constraint-driven synthesis)
  - These constraints will also be passed (by default) on to the Xilinx implementation tools via a Netlist Constraints File (NCF)
- XST constraints
  - Communicated via the XCF
    - See the *XST User Guide* in Software Manuals (toolbox.xilinx.com/docsan/xilinx9/books/manuals.pdf)

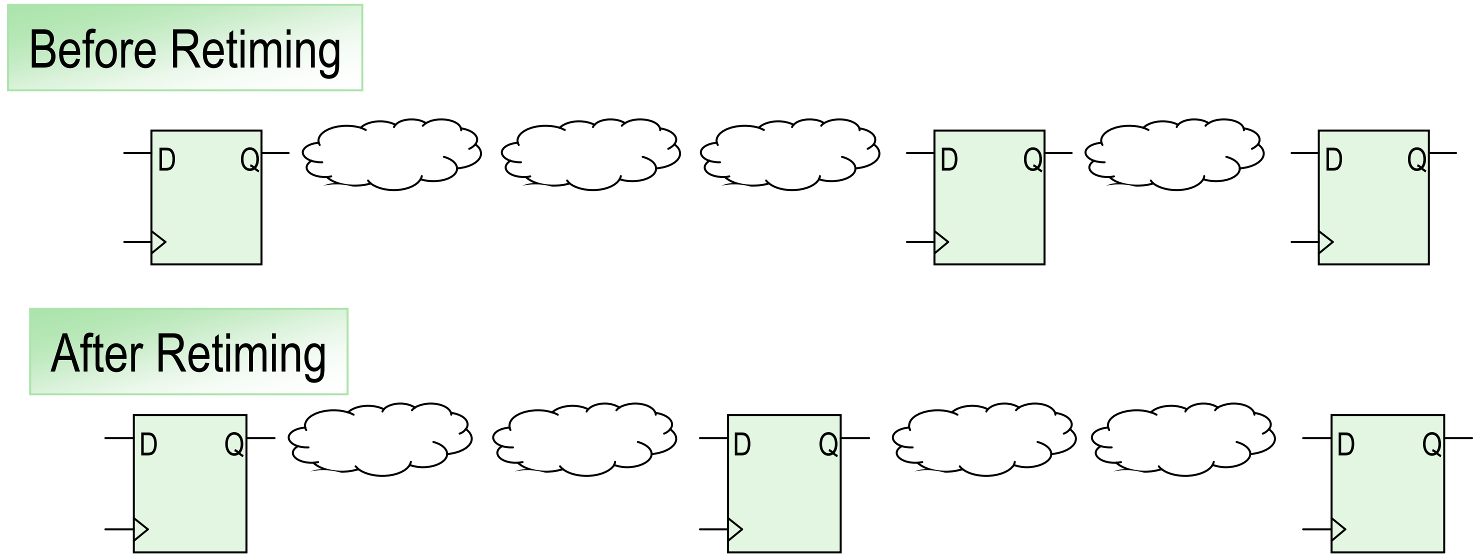
FSM Extraction

- Synplify, Precision, and XST software
- 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

- Safe state machines
  - By default, the synthesis tools will remove all decoding for illegal states
    - Even if you include VHDL *when others* or Verilog *default* cases
  - Must be turned on to use "safe" FSM implementation
    - See the Notes section for more information

Retiming

- Synplify, Precision, and XST software
- Retiming: The synthesis tool automatically tries to move register stages to balance combinatorial delay on each side of the registers



Register Duplication

- Synplify, Precision, and XST software
- Register duplication is used to reduce fanout on registers (to improve delays)
- Register duplication of the output 3-state register is used so that the IOB 3-state register can be moved inside the IOB to reduce clk-to-output delays
- Xilinx recommends manual register duplication
  - Most synthesis vendors create signals <signal_name>_rep0, _rep1, etc.
    - Implementation tools pack these signals into the same slice
    - 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

- Synplify, Precision, and XST software
- The basic settings are
  - Flatten the design: Allows total combinatorial optimization across all boundaries
  - Maintain hierarchy: Preserves hierarchy without allowing optimization of combinatorial logic across boundaries
- If you have followed synchronous design guidelines, use the setting *-maintain hierarchy*
- If you have not followed 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

XILINX®

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

Schematic Viewers

- Synplify, Precision, and XST software
- 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

Knowledge Check

Knowledge Check

- List a few of the options in the synthesis tools that help you increase performance

Answer

- List a few of the options in the synthesis tools that help you increase performance
  - Timing-driven synthesis
  - FSM extraction
  - Retiming
  - Register duplication
  - Physical optimization

Outline

- Recognizing the FPGA Challenge
- Timing and Performance Guidelines
- Inference versus Instantiation
- Synthesis Tools and Options
- **Virtex-5 FPGA Power and Verification Considerations**
- Summary

Virtex-5 FPGA Control Signals
(CE, RST, SET)

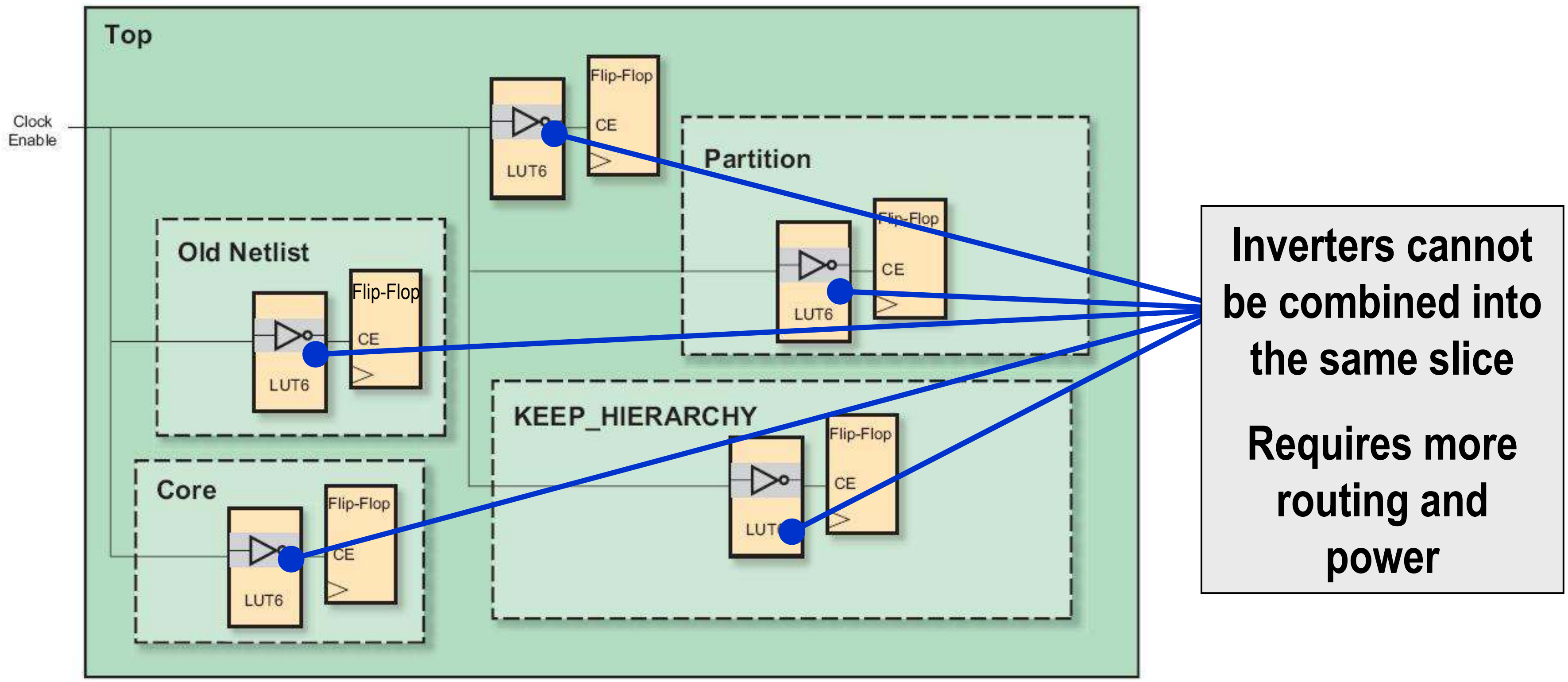
- Problem: Active-low control signals can produce sub-optimal results
  - Less than optimal utilization
    - More LUTs
    - Less dense slice packing
    - More routing resources necessary
  - Longer run times
    - Prohibits hierarchical design flows
    - More difficult timing
  - Impacts timing and power
- Why?
  - Architectural differences in the Virtex™-5 FPGA
  - Proliferation of hierarchical design methods

Better to describe inverted control signals in the top level

XILINX®

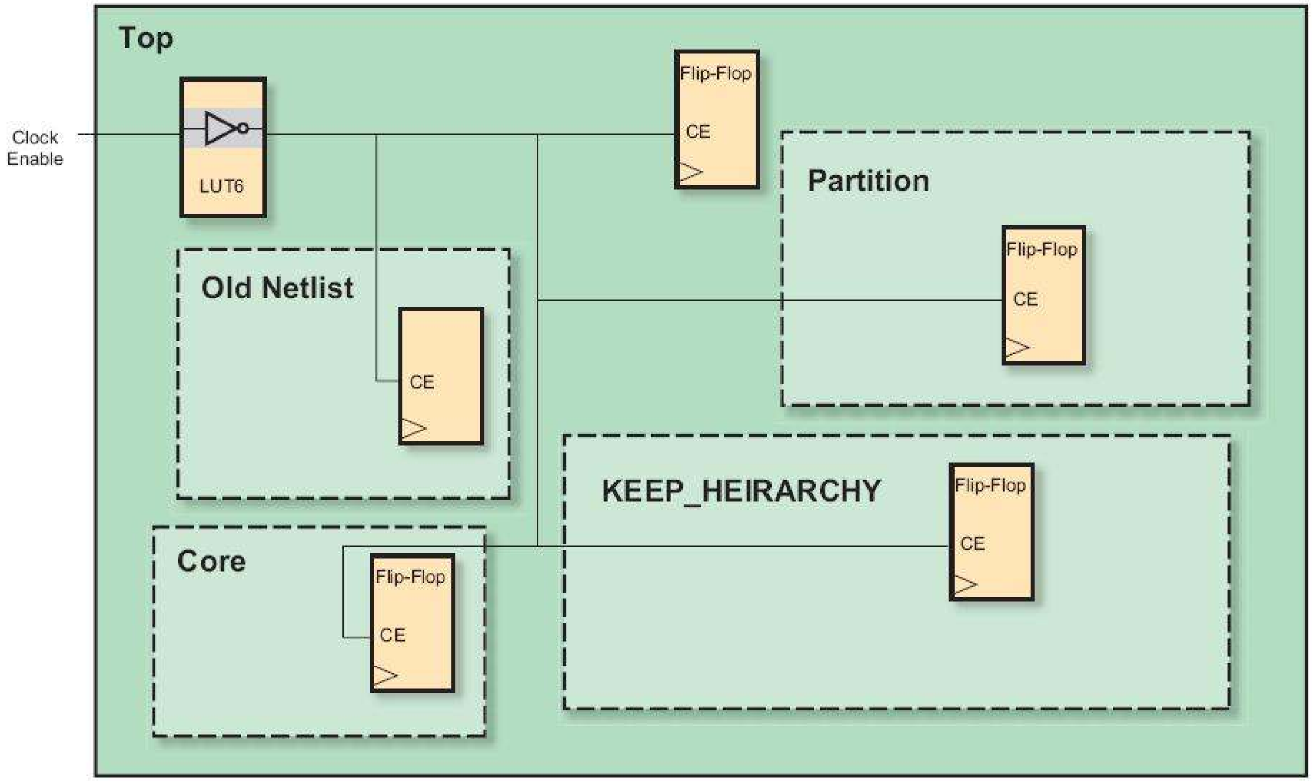
Hierarchical Design Methods

…and the effect on control signals



Hierarchical preservation may exacerbate issue…

Control Signal Polarity



Use active-high control signals
OR
Invert signal at the top level
OR
Do not preserve hierarchy

Pipelining for Performance and Power

- Maximum performance is seen when
  - There are six inputs to a logic function
    - This is different than previous architectures due to the 6-LUT
    - General rule for high-speed designs should be if more than 10 inputs are used for a logic function
  - Extra caution is taken around multipliers and RAMs
    - Many critical timing paths originate or contain block RAM or DSP48s
      - Functions mapped into DSP48E are properly pipelined
      - Output registers on RAMs
- "Good" pipelining can help improve power
  - DSP48E gets best power when pipelined
    - Not using MREG can add 15 percent power
  - Good pipelining reduces "glitch energy"

Summary

DSP48E Power

Module	DSP Slices	Clock (MHz)	Toggle Rate	MREG Used?	MULT Used?	Power (W)
Using MREG	500	445.0	12.5%	Yes	Yes	1.001
Not using MREG	500	445.0	12.5%	No	Yes	1.151

Coding for Power

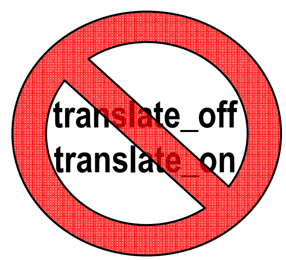
- Code for area, then performance in order to code for power
  - Code to reduce resources
    - Use dedicated resources whenever possible (hard rather than soft cores)
      - SERDES, DSP48Es, SRLs, and RAMs, for example
    - Enable resource sharing, register pushing, and use area reduction switches
  - More timing slack leads to better power optimization
    - This gives more operating room for power algorithms
    - Reduces area due to duplication and other performance optimizations
    - Pipeline and maintain balanced pipeline
  - Choose coding styles that “fit well” into the FPGA architecture
- Only enable RAMs when you need to read/write data
  - Do not tie RAM port enables to static 1 unless needed

Less area results in less power

Coding for Verification (1)

- Initialize all registers
  - RTL code can infer this behavior
    - signal Q_INT : std_logic := ‘0’ ;
  - Helps avoid VHDL simulation warnings
    - WARNING:Simulator:29 - at 110.000 ns(1): Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
  - Avoids the need for global reset for simulation initialization

- Avoid use of translate_off and translate_on
  - Rarely necessary and can cause simulation or synthesis mismatches



Code for consistency in simulation and synthesis

Coding for Verification (2)

- Synthesis attributes and switches can create synthesis or simulation mismatches
  - FSM, retiming, register/logic duplication, and treat_async_reset_as_sync, for example
- Use parameters and generics to pass primitive attributes
  - Do not use UCF or synthesis attributes
- Use KEEP_HIERARCHY constraints to help debug
  - Use KEEP_HIERARCHY at partition boundaries or where it will not have a drastic affect on performance or area

Consider broader impact of synthesis attributes/options

Knowledge Check

- Describe some unique requirements to obtain optimal results in a Virtex™-5 FPGA
- Analyze these requirements. Which of these requirements do you believe are applicable to other Xilinx FPGAs?

Answer

- Describe some unique requirements to obtain optimal results in a Virtex™-5 FPGA
- Analyze these requirements. Which of these requirements do you believe are applicable to other Xilinx FPGAs?
  - Use active-high control signals or invert the active-low signal at the top level prior to distributing throughout the design (specific to the Virtex-5 FPGA)
  - Pipelining
    - Even in low-performance designs, this will reduce long, highly buffered, routing and "glitch" energy
    - Use DSP48E and block RAM registers
    - Not Virtex-5 FPGA specific
  - Use dedicated resources to greatly reduce power requirements and increase performance (not Virtex-5 FPGA specific)

Answer

- Describe some unique requirements to obtain optimal results in a Virtex™-5 FPGA
- Analyze these requirements. Which of these requirements do you believe are applicable to other Xilinx FPGAs?
  - Only enable block RAMs as needed (not Virtex-5 FPGA specific)
  - Use parameters and generics to pass primitive attributes (not Virtex-5 FPGA specific)
  - Use KEEP_HIERARCHY to improve debug (not Virtex-5 FPGA specific)

XILINX®

Outline

- Recognizing the FPGA Challenge
- Timing and Performance Guidelines
- Inference versus Instantiation
- Synthesis Tools and Options
- Virtex-5 FPGA Power and Verification Considerations
- **Summary**

XILINX®

Summary

- Your HDL coding style can affect synthesis results
- Infer functions whenever possible
- Use one-hot encoding to improve design performance
- When coding a state machine, separate the next-state logic from the state machine output equations
- Most resources are inferable, either directly or with an attribute
- 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