Basic FPGA Architectures

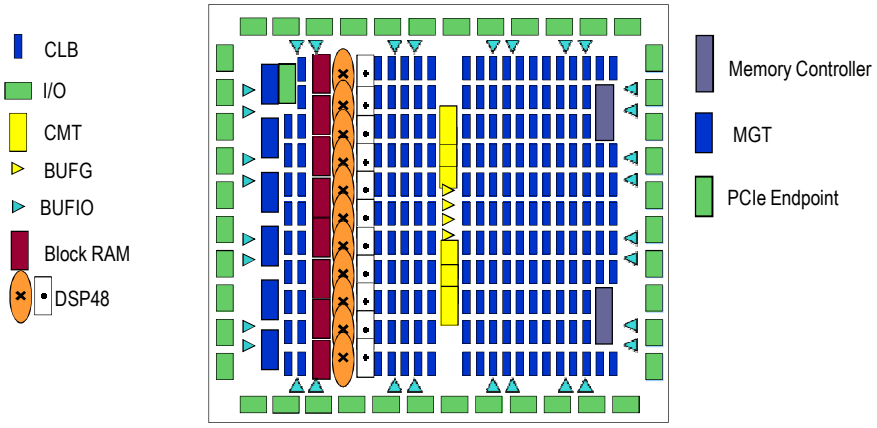
Outline

- **Overview**
- Logic Resources
- I/O Resources
- Memory and DSP48
- Clocking Resources
- Summary

Overview

- All Xilinx FPGAs contain the same basic resources
  - Logic Resources
    - Slices (grouped into CLBs)
      - Contain combinatorial logic and register resources
    - Memory
    - Multipliers
  - Interconnect Resources
    - Programmable interconnect
    - IOBs
      - Interface between the FPGA and the outside world
  - Other resources
    - Global clock buffers
    - Boundary scan logic

Spartan-6 FPGA



Spartan-6 Lowest Total Power

- 45 nm technology
- Static power reductions
  - Process & architectural innovations
- Dynamic power reduction
  - Lower node capacitance & architectural innovations
- More hard IP functionality
  - Integrated transceivers & other logic reduces power
  - Hard IP uses less current & power than soft IP
- Lower IO power
- Low power option -1L reduces power even further
- Fewer supply rails reduces power
- Two families: LX and LXT

Spartan-6 LX / LXT FPGAs

Part Number		LX4	LX9	LX16	LX25	LX45	LX100	LX150	LX25T	LX45T	LX100T	LX150T
Logic Cells		3.4K	9K	15K	24K	43K	101K	147K	24K	43K	101K	147K
CLB Flip-Flops		4.2K	11K	18K	30K	54K	126K	184K	28K	54K	126K	184K
Maximum Distributed RAM (Kbits)		32	90	136	228	401	975	1,358	228	401	975	1,385
Block RAM (18K bits each)		8	32	32	52	116	268	268	52	116	268	268
Total Block RAM (Kbits)		144	576	576	936	2,088	4,824	4,824	936	2,088	4,824	4,824
Clock Manager Tiles (CMT)		1	2	2	2	4	6	6	2	4	6	6
DSP48A1 Slices		4	16	32	38	58	182	182	38	58	182	182
PCI Express® Endpoint Block		–	–	–	–	–	–	–	1	1	1	1
Memory Controller Blocks		0	2	2	2	2	4	4	2	2	4	4
GTP Low-Power Transceivers		–	–	–	–	–	–	–	2	4	8	8
Package	Area (Pitch)	Maximum User I/O: Select IO™ Interface Pins (GTP transceivers)										
TQG144	20 x 20 mm (0.5 mm)	100	100									
CSG225	13 x 13 mm (0.8 mm)	120	160	160	160							
FTG256	17 x 17 mm (1.0 mm)			186	186							
CSG324	15 x 15 mm (0.8 mm)		200	232	226				190 (2)	190 (4)		
FGG484	23 x 23 mm (1.0 mm)				264	319	341	341	250 (4)	296 (4)	296 (4)	296 (4)
FGG676	27 x 27 mm (1.0 mm)					370	498	498		370 (4)	396 (8)	396 (8)

* Preliminary product information, subject to change. Please contact your Xilinx representative for the latest information.

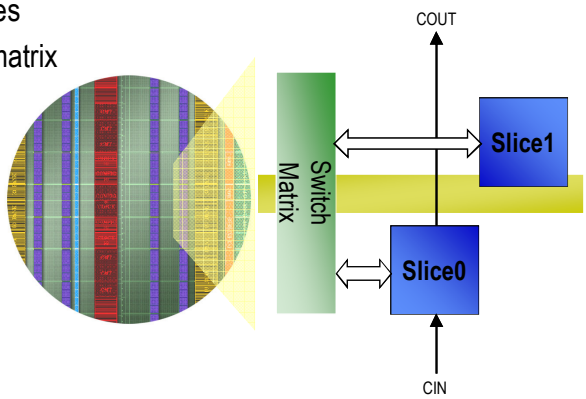
** All memory controller support x16 interface, except in CS225 package where x8 only is supported

Outline

- Overview
- **Logic Resources**
- I/O Resources
- Memory and DSP48
- Clocking Resources
- Summary

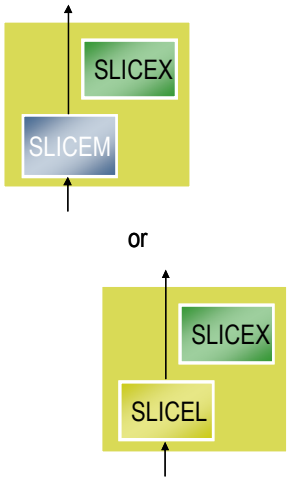
Spartan-6 FPGA CLB

- CLB contains two slices
- Connected to switch matrix for routing to other FPGA resources
- Carry chain runs vertically through Slice0 only



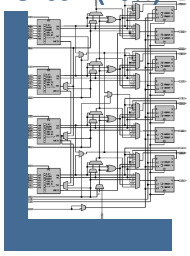
Three Types of Slices in Spartan-6 FPGAs

- SLICEM: Full slice
  - LUT can be used for logic and memory/SRL
  - Has wide multiplexers and carry chain
- SLICEL: Logic and arithmetic only
  - LUT can only be used for logic (not memory)
  - Has wide multiplexers and carry chain
- SLICEX: Logic only
  - LUT can only be used for logic (not memory)
  - No wide multiplexers or carry chain



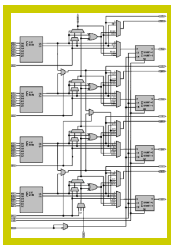
Spartan-6 CLB Logic Slices

SliceM (25%)



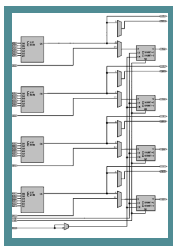
- LUT6
- 8 Registers
- Carry Logic
- Wide Function Muxes
- Distributed RAM / SRL logic

SliceL (25%)



- LUT6
- 8 Registers
- Carry Logic
- Wide Function Muxes

SliceX (50%)

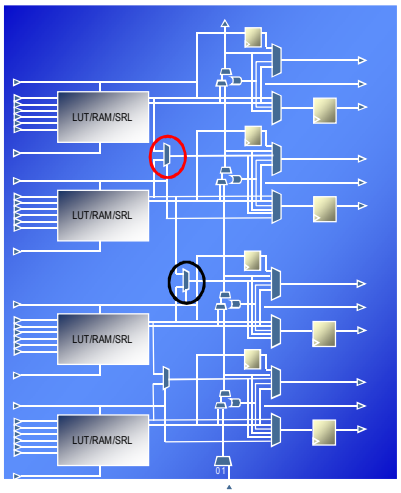


- LUT6
- Optimized for Logic
- 8 Registers

Slice mix chosen for the optimal balance of Cost, Power & Performance

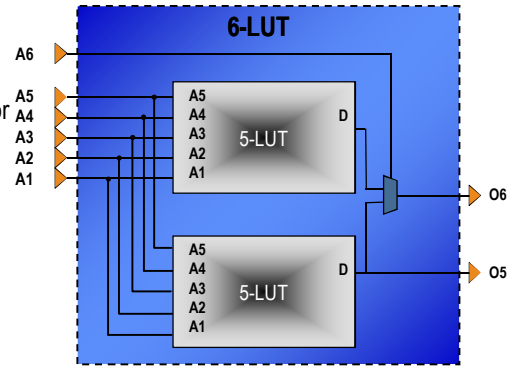
Spartan-6 FPGA SLICE

- Four LUTs
- Eight storage elements
  - Four flip-flop/latches
  - Four flip-flops
- F7MUX and F8MUX
  - Connects LUT outputs to create wide functions
  - Output can drive the flip-flop/latches
- Carry chain (Slice0 only)
  - Connected to the LUTs and the four flip-flop/latches



6-Input LUT with Dual Output

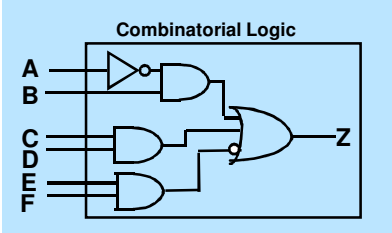
- 6-input LUT can be two 5-input LUTs with common inputs
  - Minimal speed impact to a 6-input LUT
  - One or two outputs
  - Any function of six variables or two independent functions of five variables



Combinatorial logic

Boolean logic is stored in Look-Up Tables (LUTs)

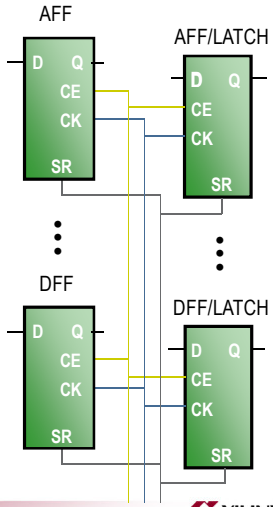
- Also called Function Generators (FGs)
- Capacity is limited by the number of inputs, not by the complexity
- Delay through the LUT is constant



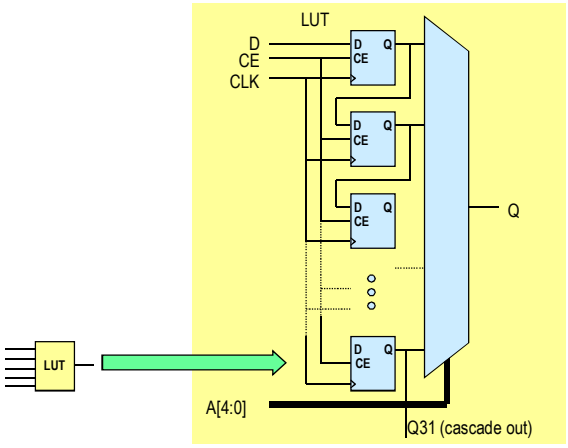
A	B	C	D	E	F	Z
0	0	0	0	0	0	1
0	0	0	0	0	1	1
0	0	0	0	1	0	1
0	0	0	0	1	1	0
0	0	0	1	0	0	1
0	1	0	1	0	1	1
	.	.	.	.	.	
1	1	0	0	0	0	1
1	1	0	1	0	1	0
.	.	.	.	.	.	.
1	1	1	1	1	1	1

Slice Flip-Flop and Flip-Flop/Latch Control

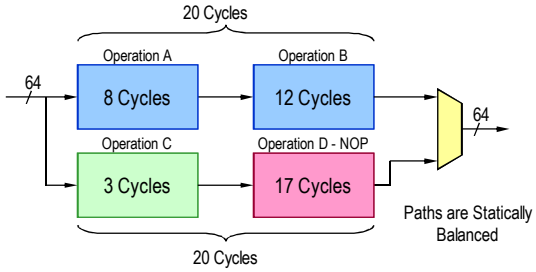
- All flip-flops and flip-flop/latches share the same CLK, SR, and CE signals
  - This is referred to as the “control set” of the flip-flops
  - CE and SR are active high
  - CLK can be inverted at the slice boundary
- Set/Reset (SR) signal can be configured as synchronous or asynchronous
  - All four flip-flop/latches are configured the same
  - All four flip-flops are configured the same
- SR will cause the flip-flop to be set to the state specified by the SRINIT attribute



Configuring LUTs as a Shift Register (SRL)



Shift Register LUT Example

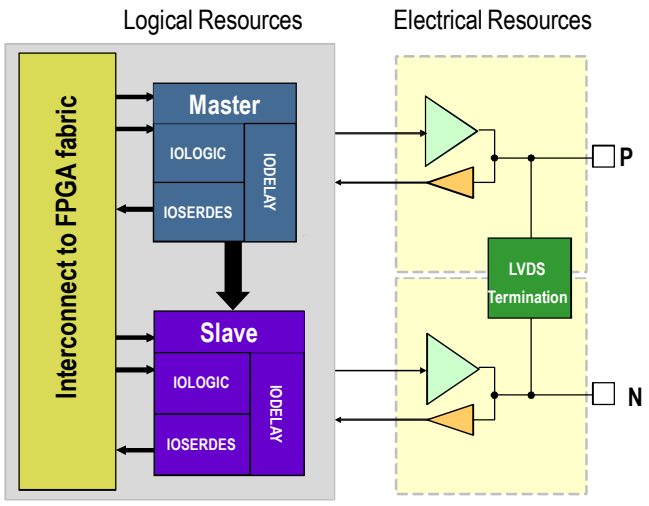


- Operation D - NOP must add 17 pipeline stages of 64 bits each
  - 1,088 flip-flops (hence 136 slices) or
  - 64 SRLs (hence 16 slices)

Outline

- Overview
- Logic Resources
- **I/O Resources**
- Memory and DSP48
- Clocking Resources
- Summary

I/O Block Diagram



Spartan-6 FPGA Supports 40+ Standards

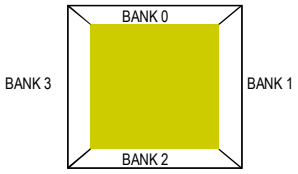
- Each input can be 3.3 V compatible
- LVCMOS (3.3 V, 2.5 V, 1.8 V, 1.5 V, and 1.2 V)
- LVCMOS_JEDEC
- LVPECL (3.3 V, 2.5 V)
- PCI
- **I2C***
- HSTL (1.8 V, 1.5 V; Classes I, II, III, IV)
  - DIFF_HSTL_I, DIFF_HSTL_I_18
  - **DIFF_HSTL_II***
- SSTL (2.5 V, 1.8 V; Classes I, II)
  - DIFF_SSTL_I, DIFF_SSTL18_I
  - **DIFF_SSTL_II***
- LVDS, Bus LVDS
- RSDS_25 (point-to-point)

Easier and More Flexible I/O Design!

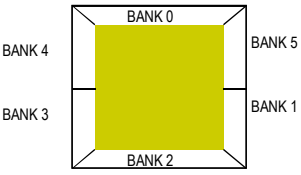
*** Newly added standards**

Spartan-6 FPGA I/O Bank Structure

- All I/Os are on the edges of the chip
- I/Os are grouped into banks
  - 30 ~ 83 I/O per banks
  - Eight clock pins per edge
  - Common VCCO, VREF
    - Restricts mixture of standards in one bank
- The differential driver is only available in Bank0 and Bank2
  - Differential receiver is available in all banks
  - On-chip termination is available in all banks



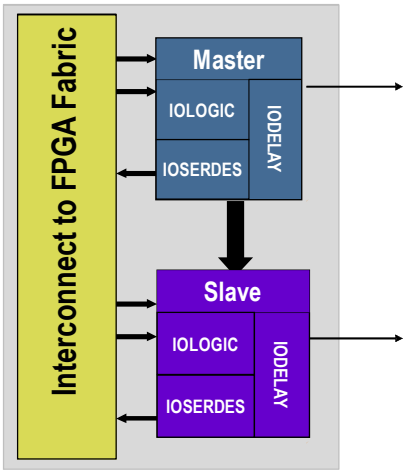
Chip View
(LX45/T and Smaller)



Chip View
(LX100/T and Larger)

I/O Logical Resources

- Two IOLOGIC block per I/O pair
  - Master and slave
  - Can operate independently or concatenated
- Each IOLOGIC contains
  - IOSERDES
    - Parallel to serial converter (serializer)
    - Serial to parallel converter (De-serializer)
  - IODELAY
    - Selectable fine-grained delay
  - SDR and DDR resources



Outline

- Overview
- Logic Resources
- I/O Resources
- **Memory and DSP48**
- Clocking Resources
- Summary

SLICEM Used as Distributed SelectRAM Memory

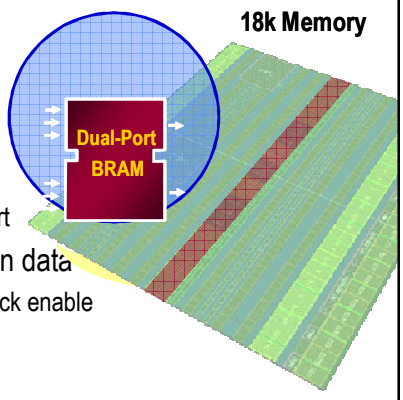
Single Port	Dual Port	Simple Dual Port	Quad Port
32x2	32x2**D**	32x6**SDP**	32x2**Q**
32x4	32x4**D**	64x3**SDP**	64x1**Q**
32x6	64x1**D**		
32x8	64x2**D**		
64x1	128x1**D**		
64x2			
64x3			
64x4			
128x1			
128x2			
256x1			

Each port has independent address inputs

- Uses the same storage that is used for the look-up table function
- Synchronous write, asynchronous read
  - Can be converted to synchronous read using the flip-flops available in the slice
- Various configurations
  - Single port
    - One LUT6 = 64x1 or 32x2 RAM
    - Cascadable up to 256x1 RAM
  - Dual port (**D**)
    - 1 read / write port + 1 read-only port
  - Simple dual port (**SDP**)
    - 1 write-only port + 1 read-only port
  - Quad-port (**Q**)
    - 1 read / write port + 3 read-only ports

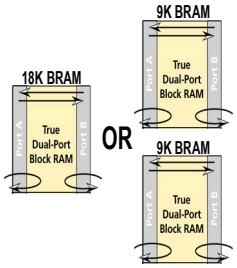
Spartan-6 FPGA Block RAM Features

- 18 kb size
  - Can be split into two independent 9-kb memories
- Performance up to 300 MHz
- Multiple configuration options
  - True dual-port, simple dual-port, single-port
- Two independent ports access common data
  - Individual address, clock, write enable, clock enable
  - Independent widths for each port
- Byte-write enable



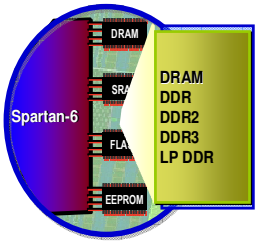
Better, More BRAM

- More Block RAMs
  - 2x higher BRAM to Logic Cell ratio than Spartan-3A platform
- More port flexibility
  - 18K can be split into two 9K BRAM blocks and can be independently addressed
- Improves buffering, caching & data storage
  - Excellent for embedded processing, communication protocols
  - Enables DSP blocks to provide more efficient video and surveillance algorithms
- Lower Static Power



Memory Controller

- Only low cost FPGA with a “hard” memory controller
- Guaranteed memory interface performance providing
  - Reduced engineering & board design time
  - DDR, DDR2, DDR3 & LP DDR support
  - Up to 12.8Mbps bandwidth for each memory controller
- Automatic calibration features
- Multiport structure for user interface
  - Six 32-bit programmable ports from fabric
  - Controller interface to 4, 8 or 16 bit memories devices

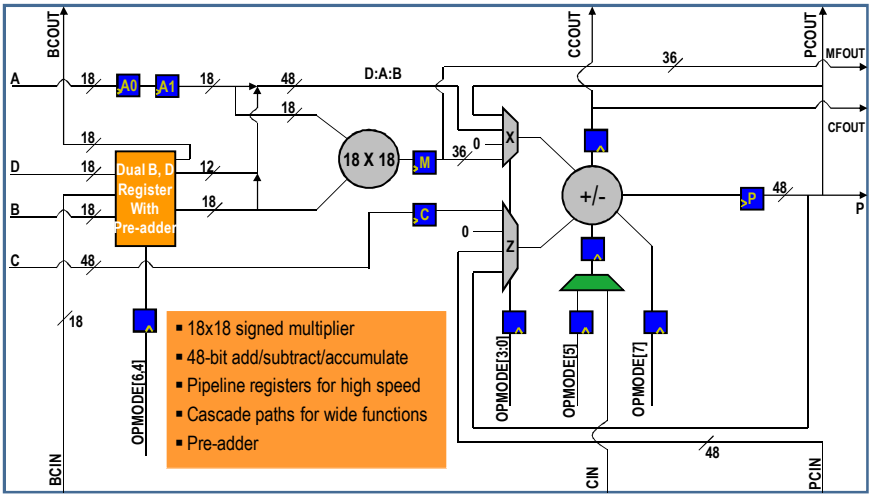


Spartan-6 Hard Memory Controller

- New Hard Block Memory Controller
  - Up to 4 controllers per device
- Why a Hard Memory Block?
  - Very common design component
  - Multiple customer benefits

Customer Requests	Spartan-6 Hard Block Memory Controller Benefits
Higher performance	• Up to 800 Mbps
Lower cost	• Saves soft logic, smaller die
Lower power	• Dedicated logic
Easier designs	• Timing closure no longer an issue • Configurable MultiPort user interface • CoreGen/MIG wizard & EDK support

Spartan-6 FPGA DSP48A1 Slice

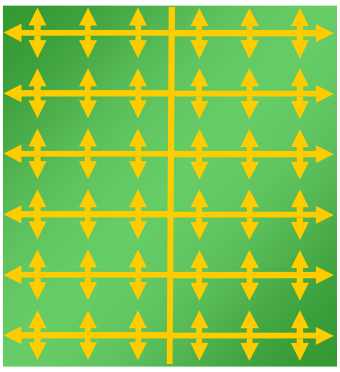


Outline

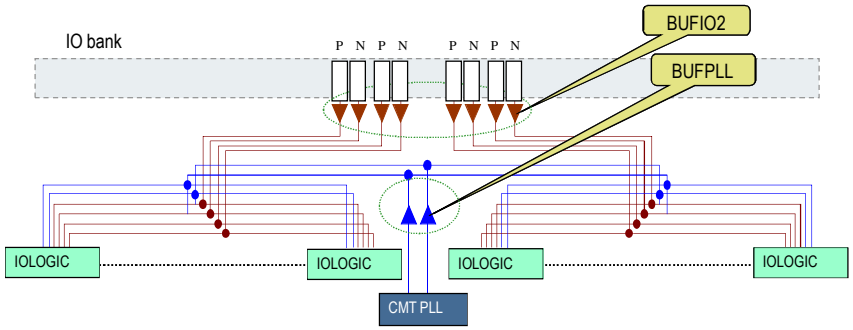
- Overview
- Logic Resources
- I/O Resources
- Memory and DSP48
- **Clock Resources**
- Summary

Spartan-6 FPGA Global Clock Network

- 16 global clock buffers in the Spartan-6 FPGA allow clocks to be distributed to potentially every clocked element on the die
- 16 HCLK lines connect clock signals resources in each row
- HCLK lines can be driven by
  - Global clock buffers
  - DCM outputs
  - PLL outputs

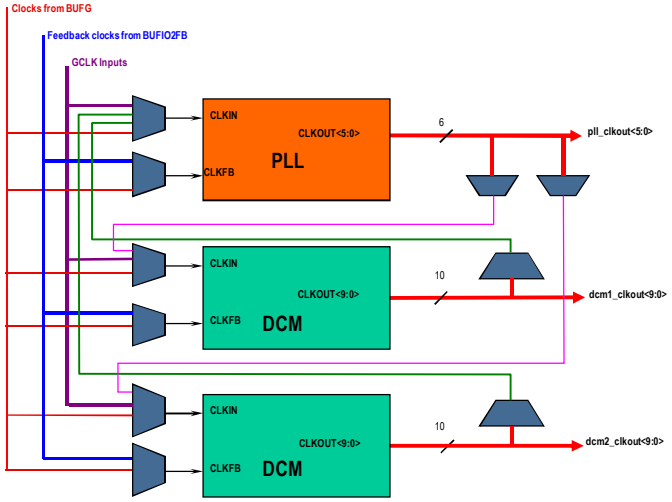


Spartan-6 FPGA I/O Clock Network



- Special clock network dedicated to I/O logical resources
  - Independent of global clock resources
  - Speeds up to 1 GHz
- Multiple sources for clocking I/O logic
  - BUFIO2: for high-speed dedicated I/O clock signals
  - BUFPLL: for clocks driven by the PLL in the CMT

Spartan-6 FPGA Clock Management Tile (CMT)



Outline

- Overview
- Logic Resources
- I/O Resources
- Memory and DSP48
- Clocking Resources
- **Summary**

Summary

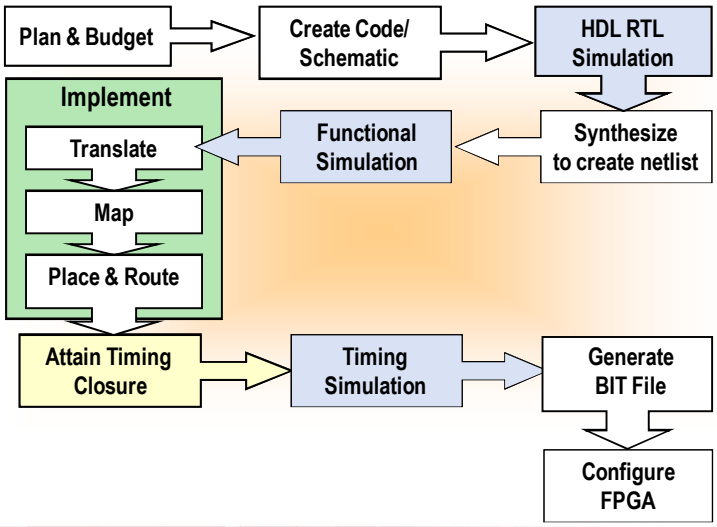
- The Spartan-6 FPGA slices contain four 6-input LUTs, eight registers, and carry logic
  - LUTs can perform any combinatorial function of up to six inputs
  - LUTs are connected with dedicated multiplexers and carry logic
  - Some LUTs can be configured as shift registers or memories
- The Spartan-6 FPGA IOBs contain DDR registers as well as SERDES resources
- The SelectIO™ interfaces enable direct connection to multiple I/O standards
- The Spartan-6 FPGA includes dedicated block RAM and DSP slice resources
- The Spartan-6 FPGA includes dedicated DCMs, PLLs, and routing resources to improve your system clock performance and generation capability
- Latest introduced families are architected for power efficiencies
  - Consists of Artix, Kintex, and Virtex devices

Xilinx Tool Flow

Outline

- **Overview**
- ISE Foundation
- Summary

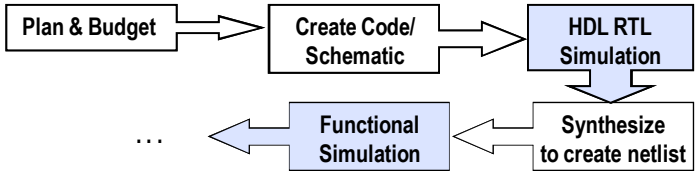
Xilinx Design Flow



Design Entry

Create designs in HDL or Schematic

- Plan and budget
- Whichever method you use, you will need a tool to generate an EDIF or NGC netlist to bring into the Xilinx implementation tools
  - Popular synthesis tools include: Synplify, Precision, FPGA Compiler II, and XST
- Tools available to assist in design entry
  - Architecture Wizard, CORE Generator™ system, and StateCAD tools
- Simulate the design to ensure that it works as expected!



Synthesis

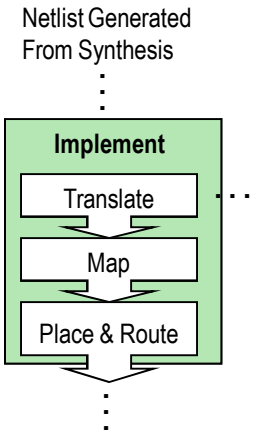
Generate a netlist file

- After coding up your HDL code, you will need a tool to generate a netlist (NGC or EDIF)
  - Xilinx Synthesis Tool (XST) included
  - Support for Popular Third Party Synthesis tools: Synplify and Synplify Pro from Synplicity, and Precision from Mentor Graphics

Implementation

Process a netlist file

- Consists of three phases
  - **Translate:** Merge multiple design files into a single netlist
  - **Map:** Group logical symbols from the netlist (gates) into physical components (slices and IOBs)
  - **Place & Route:** Place components onto the chip, connect the components, and extract timing data into reports
- Access Xilinx reports and tools at each phase
  - Timing Analyzer, Floorplanner, FPGA Editor, XPower



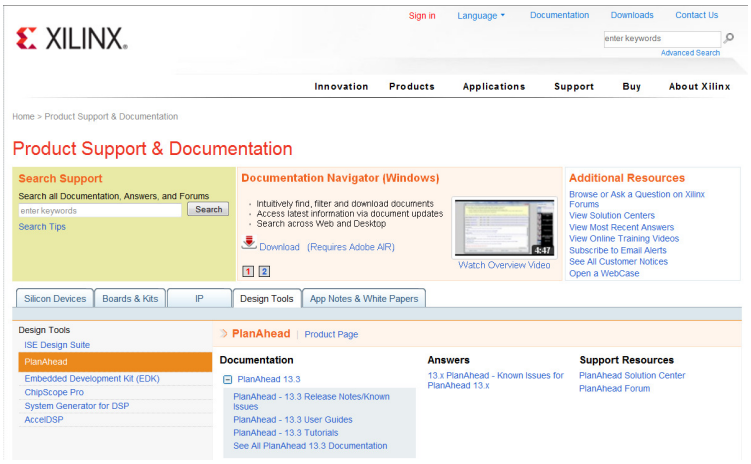
Configuration

Testing and Verification

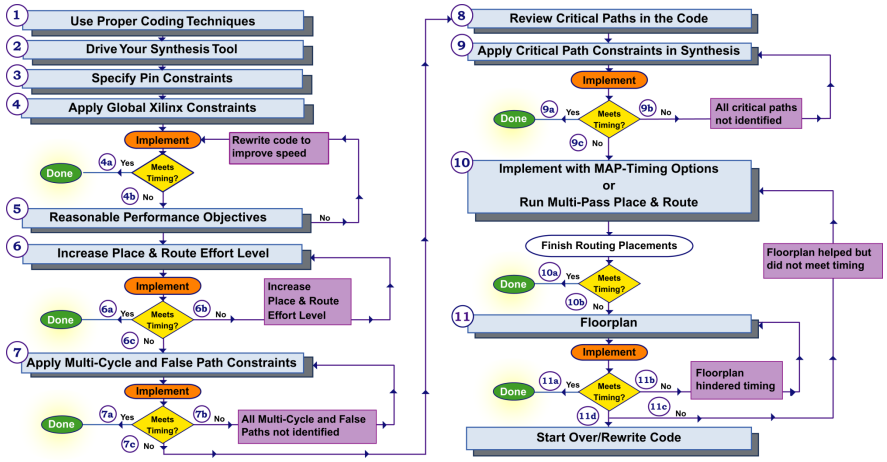
- Once a design is implemented, you must create a file that the FPGA can understand
  - This file is called a bitstream: a BIT file (.bit extension)
- The BIT file can be downloaded
  - Directly into the FPGA
    - Use a download cable such as Platform USB
  - To external memory device such as a Xilinx Platform Flash PROM
    - Must first be converted into a PROM file

Online Software Manuals

See Development System Reference Guide for Flow Diagrams



Timing Closure



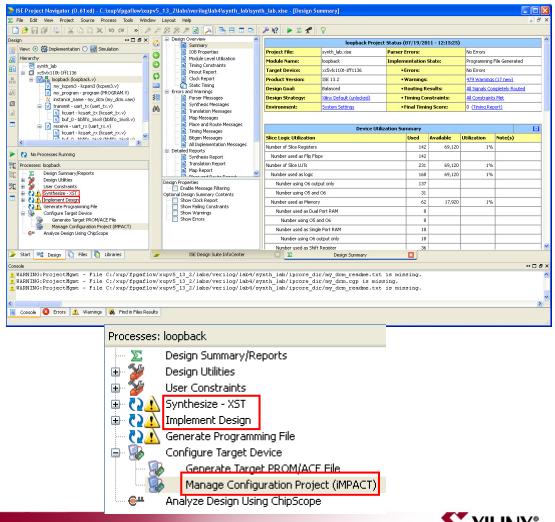
Outline

- Overview
- **ISE Foundation**
- Summary

ISE Project Navigator

Xilinx ISE Foundation is built around the Xilinx Design Flow

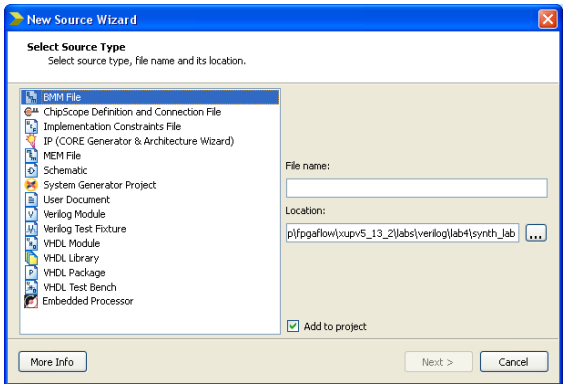
- Enter Designs
- Access to synthesis tools
  - Including third-party synthesis tools
- Implement your design with a simple double-click
  - Fine-tune with easy-to-access software options
- Download
  - Generate a bitstream
  - Configure FPGA using iMPACT



Entering Designs

Source Wizard available to assist with design entry

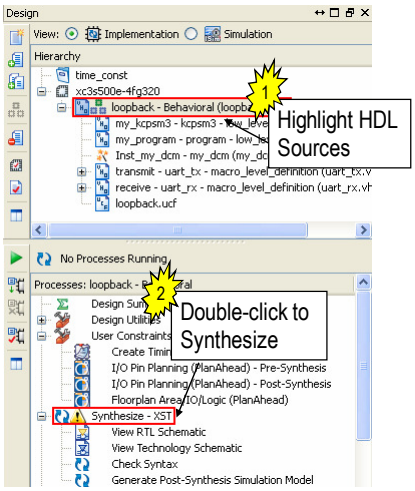
- Select source type
  - Design Entry Methods
    - Schematic
    - HDL source (VHDL and Verilog)
  - Design Entry Tools
    - Architecture Wizard
    - BMM/MEM/UCF Files
    - Core Generator
    - ChipScope
    - Embedded Processor
    - System Generator
  - Simulation Test Bench
    - VHDL
    - Verilog



Synthesizing Designs

Generate a netlist file using XST (Xilinx Synthesis Technology)

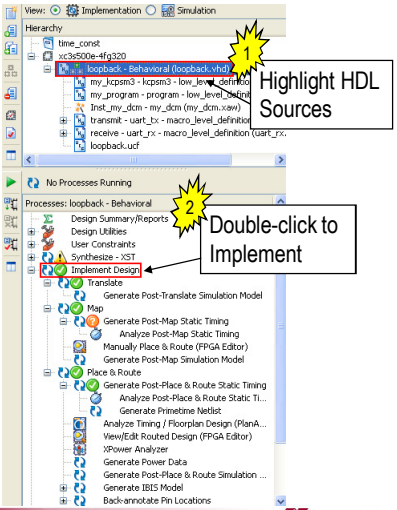
- Synthesis Processes and Analysis
  - Access report
  - View Schematics (RTL or Technology)
  - Check Syntax
  - Generate Post-Synthesis Simulation Model



Implementing Designs

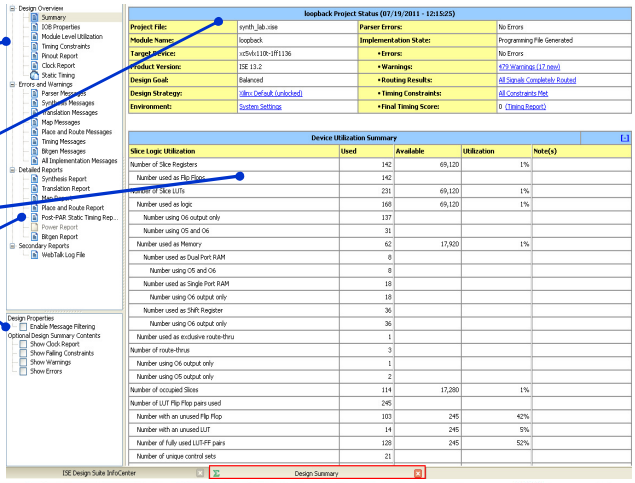
Process netlist generated from synthesis

- Implement a design
  - Translate
    - Access reports
    - Post-Translate Simulation Model
  - Map
    - Access reports
    - Post-Map Static Timing
    - Manually place components
    - Post-Map Simulation Model
  - Place & Route
    - Access reports
    - Analyze timing/Floorplan (PlanAhead)
    - Manually place & route components
    - And more



The Design Summary Displays Design Data

- Quick View of Reports, Constraints
- Project Status
- Device Utilization
- Detailed Reports
- Design Properties
- Performance Summary (not shown)



loopback Project Status (07/19/2011 - 12:15:25)

Project File:	synth_lab.xise	Parser Errors:	No Errors
Module Name:	loopback	Implementation State:	Programming File Generated
Target Device:	xc5vlx110t-1ff1136	• Errors:	No Errors
Product Version:	ISE 13.2	• Warnings:	429 Warnings (17 new)
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

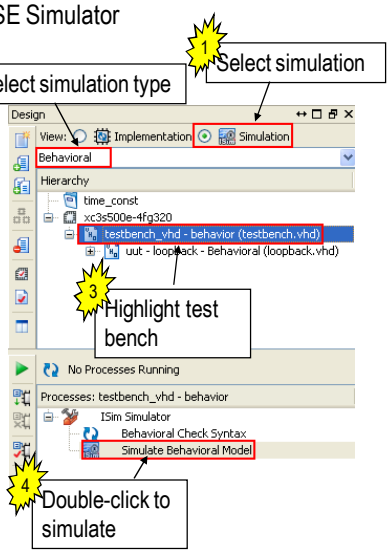
Device Utilization Summary

Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	142	69,120	1%	
Number used as Flip Flops	142			
Number of Slice LUTs	231	69,120	1%	
Number used as logic	168	69,120	1%	
Number using O6 output only	137			
Number using O5 and O6	31			
Number used as Memory	62	17,920	1%	
Number used as Dual Port RAM	0			
Number using O5 and O6	0			
Number used as Single Port RAM	18			
Number using O6 output only	18			
Number used as SRL Register	36			
Number using O6 output only	36			
Number used as exclusive route-thru	1			
Number of route-thrus	3			
Number using O6 output only	1			
Number using O5 output only	2			
Number of occupied Slices	114	17,280	1%	
Number of LUT Flip Flop pairs used	245			
Number with an unused Flip Flop	103	245	42%	
Number with an unused LUT	14	245	5%	
Number of fully used LUT-FF pairs	128	245	52%	
Number of unique control sets	21			

Simulating Designs

Verify the design with the ISE Simulator

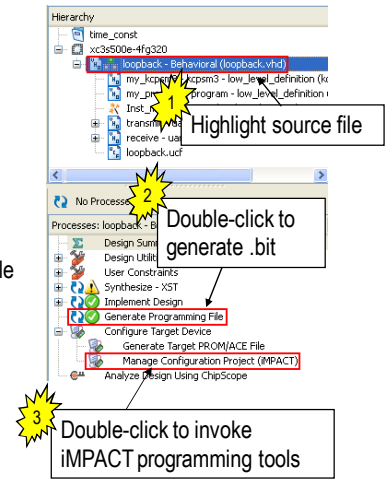
- Add a test bench
  - VHDL, Verilog, or Xilinx waveform file
- Perform a Behavioral Simulation
  - Use UNISIM/UniMacro library when FPGA primitives are instantiated in the design
  - Use XilinxCoreLib library when IP cores are instantiated in the design
- Perform a timing simulation
  - Use Xilinx SIMPRIM library when FPGA primitives are instantiated in the design
- SmartModels
  - Simulation library for both functional and timing simulation of Xilinx Hard-IP such as PPC, PCIe, GT, TEMAC are used in the design



Configuring FPGAs

Generate PROM files and download to devices using iMPACT

- Configure FPGAs from computer
  - Use iMPACT to download bitstream from computer to FPGA via Xilinx download cable (ie. Platform USB)
- Configure FPGAs from External Memory
  - Xilinx Platform Flash
    - Use iMPACT to generate PROM file and download to PROM using Xilinx download cable
  - Generic Parallel PROM
    - Use iMPACT to generate PROM file - no support for programming
  - Compact Flash (Xilinx System ACE required)
    - Use iMPACT to generate SysACE file - no support for programming



Outline

- Overview
- ISE
- **Summary**

Summary

- Implementation means more than Place & Route
- Xilinx provides a simple *pushbutton* tool to guide you through the Xilinx design process

Architecture Wizard and Pins Assignment

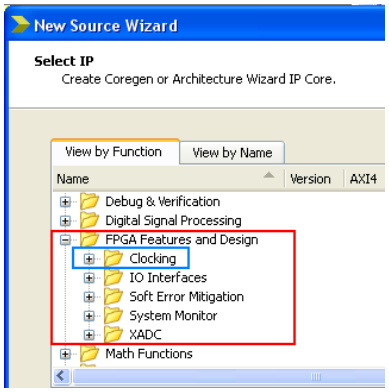
Outline

- **Architecture Wizard**
- Pins Assignment (PlanAhead)
- Summary

The Architecture Wizard

Configure and add FPGA resources in the user design

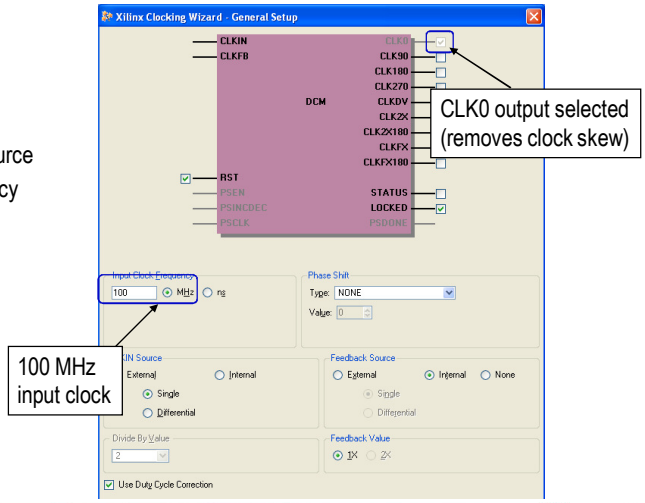
- Access through New Source Wizard
  - IP (CORE Generator & Architecture Wizard)
- Configure various resources
  - Depends on target device
    - Digital Clock Manager (see next slide)
    - IO Interfaces
    - Soft Error Mitigation
    - System Monitor
    - XADC



Configure a DCM

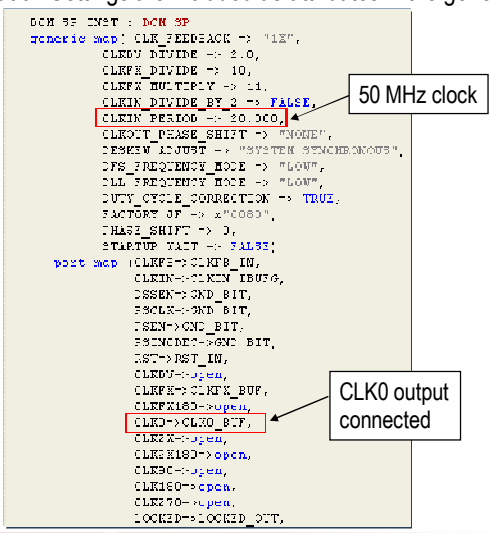
DCMs (Digital Clock Manager) are built into the latest FPGAs

- Main window
  - Select pins
  - Specify
    - Reference source
    - Clock frequency
    - Phase shift
  - Advanced button



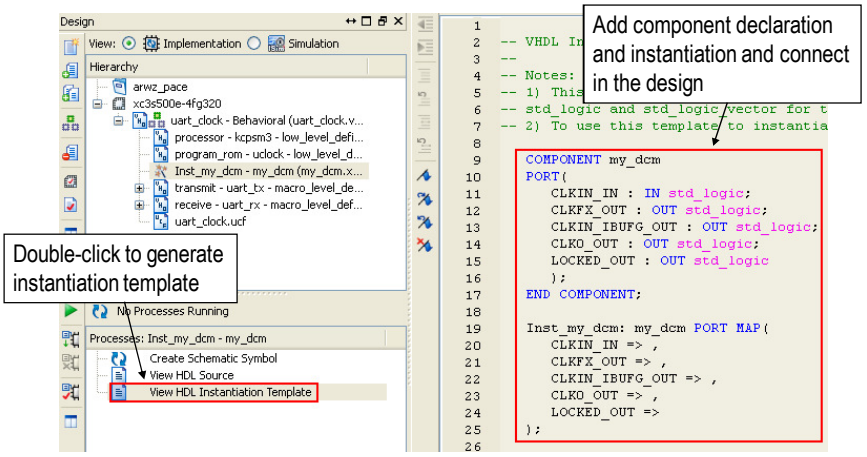
Generate DCM

The Configuration Settings are included as attributes in the generated HDL code



Instantiate DCM

Connect the DCM in the design with the automatically generated instantiation template



Outline

- Architecture Wizard
- **Pins Assignment (PlanAhead)**
- Summary

PlanAhead

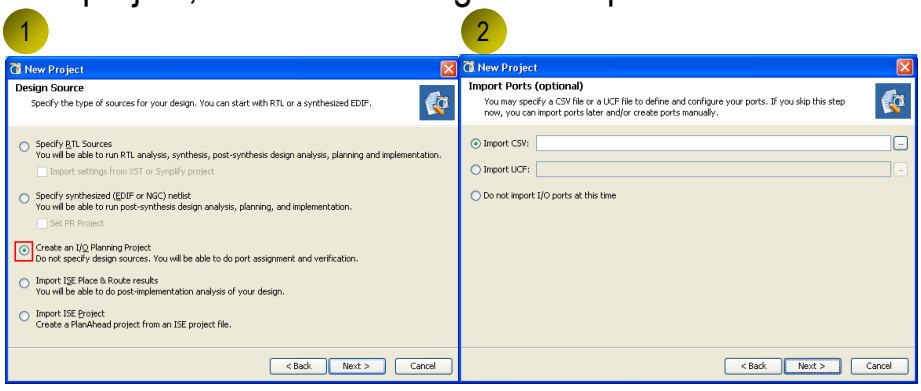
- Integrated in ISE
- Default installation with ISE
- Allows to import RTL sources
- Allows to import synthesized netlists
- Can import placed and routed design for further analysis
- Enables pin planning without requiring source files through I/O Planner

Two Design Flows

- Invoke PlanAhead separately to create an empty project to assign pins and perform early analysis from pins point of view
  - Use I/O Planner feature of PlanAhead
- Invoke PlanAhead from an existing ISE project
  - For pin assignment purpose
    - You can invoke either before synthesizing the design
    - You can invoke after synthesizing the design
  - For layout purpose

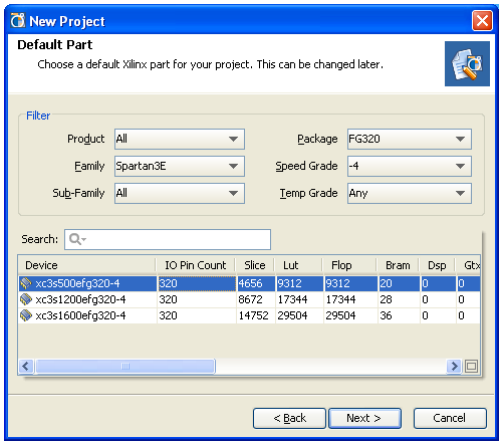
I/O Planner Flow (1)

- I/O Planner allows you to create a new empty project, define and configure new ports

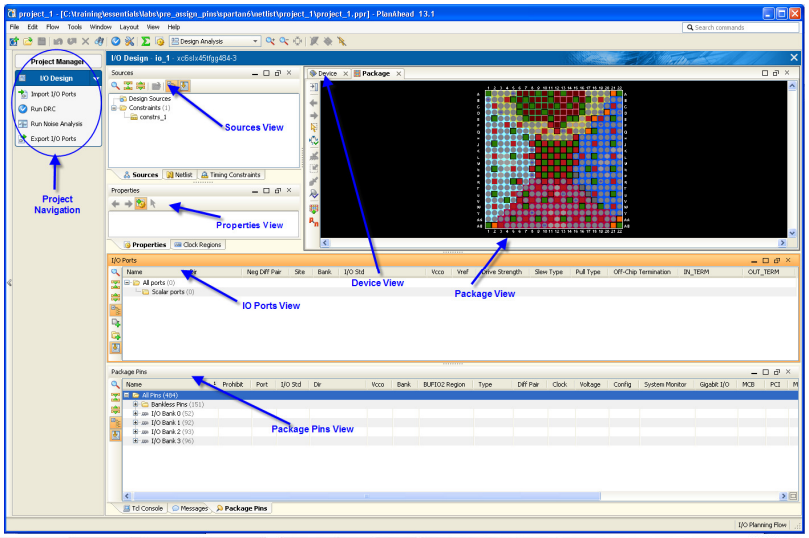


Device and Package (2)

- Select device, package, and speed grade

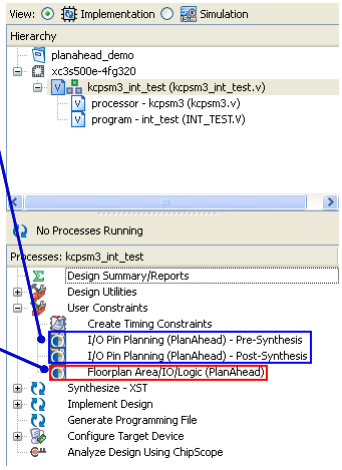


I/O Planner View

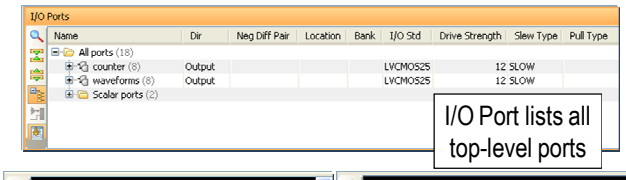


Invoking PlanAhead from ISE

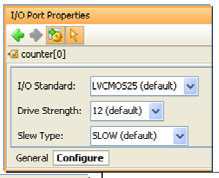
- Assign Package Pins
  - Assign I/O locations, specify I/O banks and I/O standards, prohibit I/O locations
  - Verify the pin type to the logic assignment
  - Perform a DRC check to prevent illegal placements
- Create Area Constraints
  - Create area constraints for logic and display I/Os on the periphery to show connectivity
  - Begin floorplanning early in the design flow
  - Verify area constraints



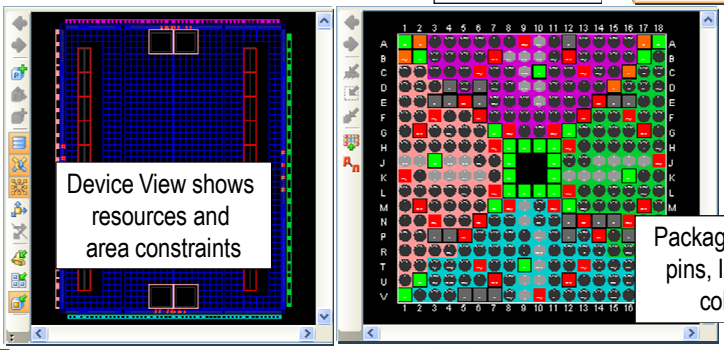
Pin Assignment Related Windows



I/O Port lists all top-level ports



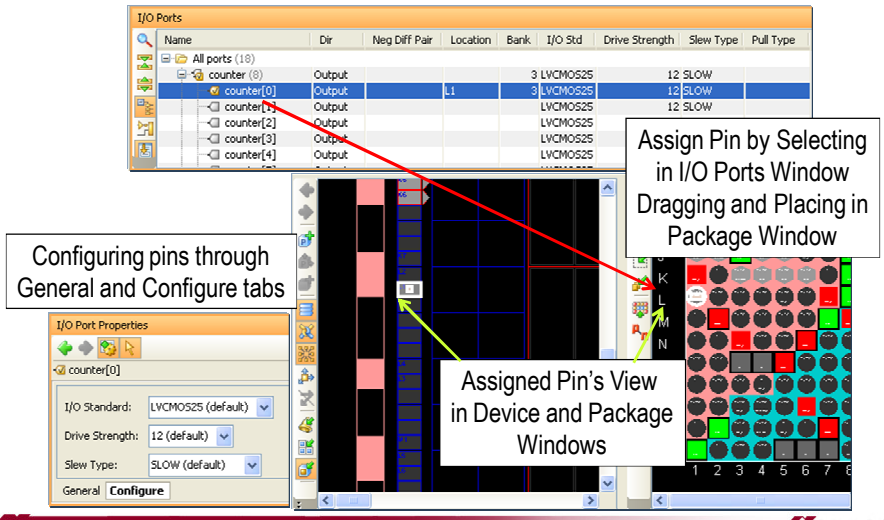
I/O Port Properties allows configuring pins



Device View shows resources and area constraints

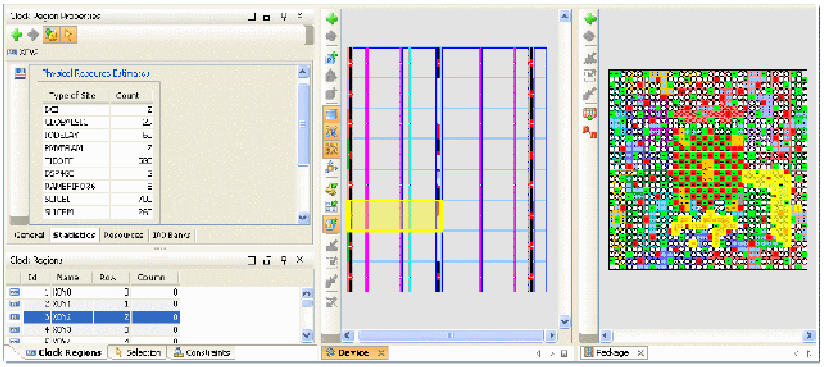
Package View shows pins, I/O banks are color-coded

Assigning Individual Pins



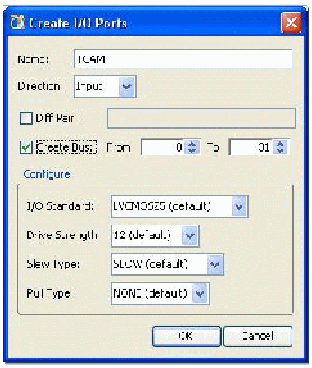
Viewing Clock Regions

Clock Regions window shows various clock regions available in device



Creating I/O Ports Interface

- Select the **Create I/O Ports** popup menu command in the I/O Ports view
  - Enter name
  - Select direction
  - Check Create Bus if needed
  - Configure I/O



PlanAhead Tools

- Perform DRC
- Run WASSO analysis
- Make pins compatible for package migration
  - Available for Virtex-5 only
- Perform timing analysis
- Run synthesis and implementation tools

Outline

- Architecture Wizard
- Pins Assignment (PlanAhead)
- **Summary**

Summary

- The Architecture Wizard enables the configuration of complex FPGA resources based on target family:
  - Digital Clock Manager (see next slide)
  - IO Interfaces
  - Soft Error Mitigation
  - System Monitor
  - XADC
- PlanAhead provides ease in I/O locations assignments and create area constraints for logic

Reading Reports

Outline

- **Introduction**
- Area Goals
- Performance Goals
- Summary

Introduction

- After you have implemented your design, how can you tell whether the implementation was successful?
- First and foremost, how do you define a successful design?
- Answer: A successful design:
  - Fits into the device
  - Achieves performance goals

Outline

- Introduction
- **Area Goals**
- Performance Goals
- Summary

Area Goals

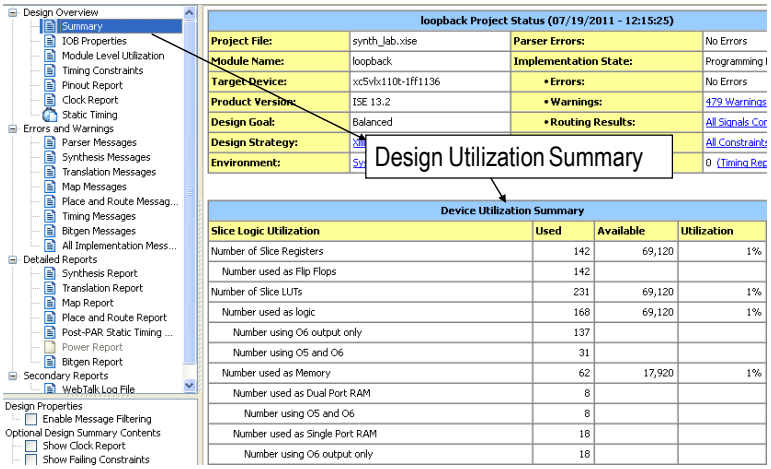
- How do we know whether the design fits into the device?
- How do we know whether there is enough room in the device for more logic? If there is enough room, exactly how much space is available?
- If the design fits into the device, was it able to route completely?

Area Goals

- How do we know whether the design fits into the device?
  - Information can be found in the Design Summary & Map Report
- How do we know whether there is enough room in the device for more logic? If there is enough room, exactly how much space is available?
  - Information can be found in the Design Summary, Map Report, or the Place & Route Report
- If the design fits into the device, was it able to route completely?
  - Information can be found in the Place & Route Report

Device Utilization Summary

Get quick access to used and available resources through the FPGA Design Summary

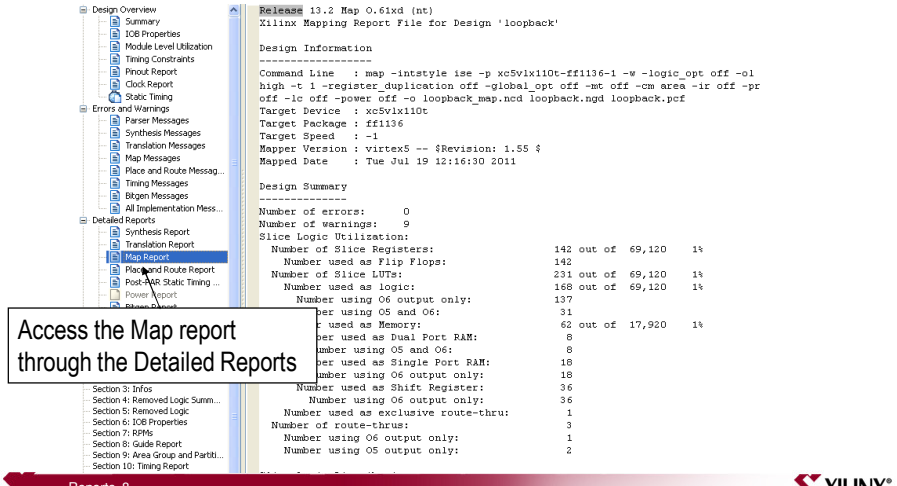


loopback Project Status (07/19/2011 - 12:15:25)			
Project File:	synth_lab.xise	**Parser Errors:**	No Errors
Module Name:	loopback	**Implementation State:**	Programming
Target Device:	xc5vlx110t-1ff1136	**• Errors:**	No Errors
Product Version:	ISE 13.2	**• Warnings:**	479 Warnings
Design Goal:	Balanced	**• Routing Results:**	All Signals Cor
Design Strategy:	Xil		All Constraints
Environment:	Sy		0 (Timing Rep

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	142	69,120	1%
Number used as Flip Flops	142		
Number of Slice LUTs	231	69,120	1%
Number used as logic	168	69,120	1%
Number using O6 output only	137		
Number using O5 and O6	31		
Number used as Memory	62	17,920	1%
Number used as Dual Port RAM	8		
Number using O5 and O6	8		
Number used as Single Port RAM	18		
Number using O6 output only	18		

Device Utilization Summary

Get quick access to used and available resources from the Map report

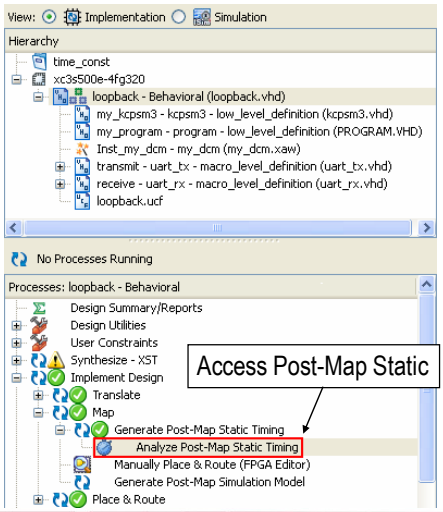


Outline

- Introduction
- Area Goals
- **Performance Goals**
- Summary

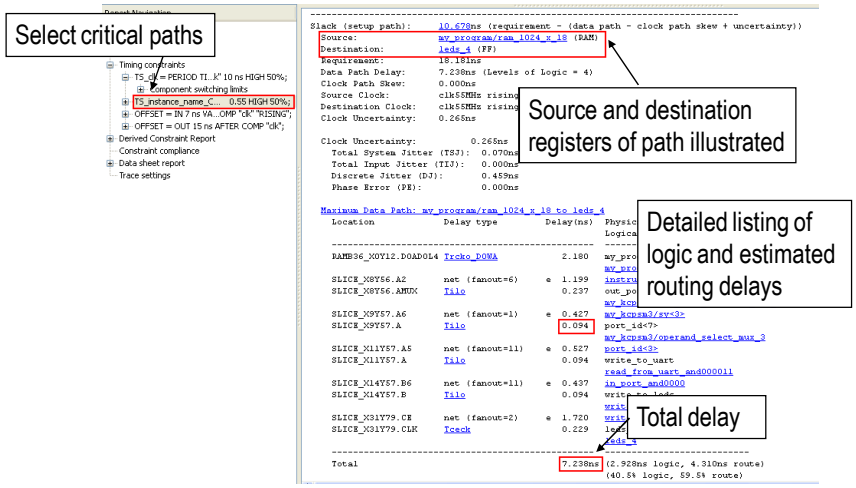
Post-Map Static Timing Report

Evaluate logic delays to see if you should proceed to place & route



Analyze Logic Timing

Look at critical paths to determine if timing is reasonable



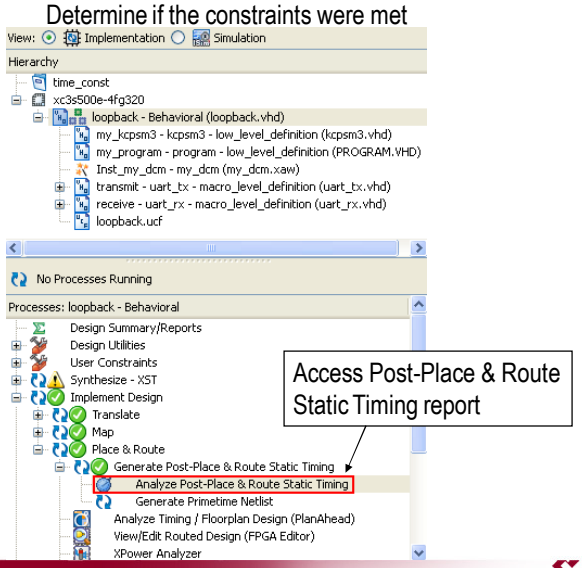
Reasonable Logic Delays

Recommended to evaluate using the 60/40 rule

- If less than 60 percent of the timing budget is used for logic delays, the Place & Route tools should be able to meet the constraint easily
- Between 60 to 80 percent, the software run time will increase
- Greater than 80 percent, the tools may have trouble meeting your goals

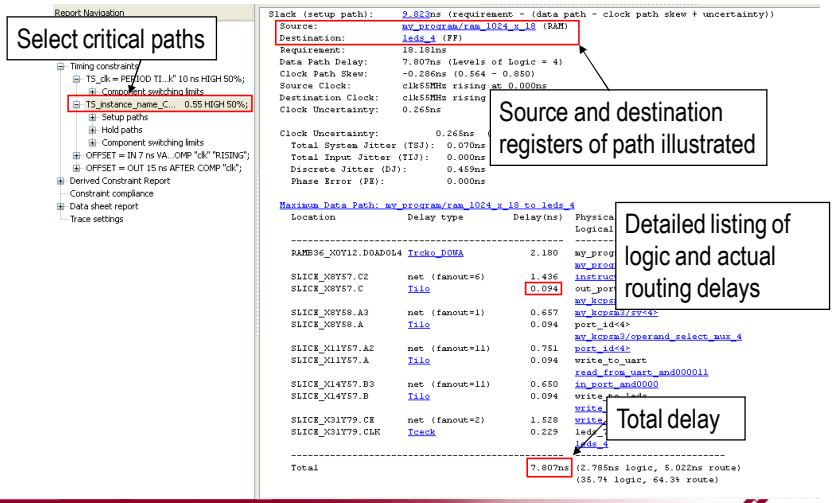
Post-Place & Route Static Timing Report

Determine if the constraints were met



Analyze Logic and Route Timing

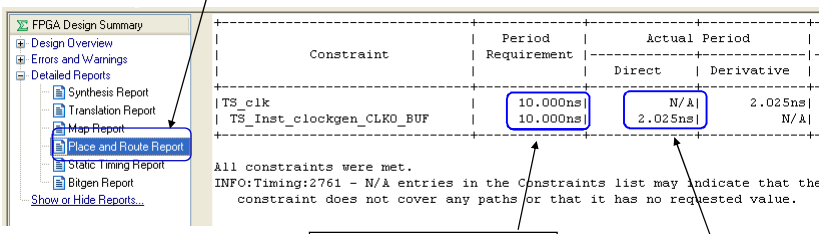
Look at critical paths to determine if timing goals are met



Timing Summary

Provides statistics on average routing delays and performance versus constraints

Access Place & Route report through Detailed Reports

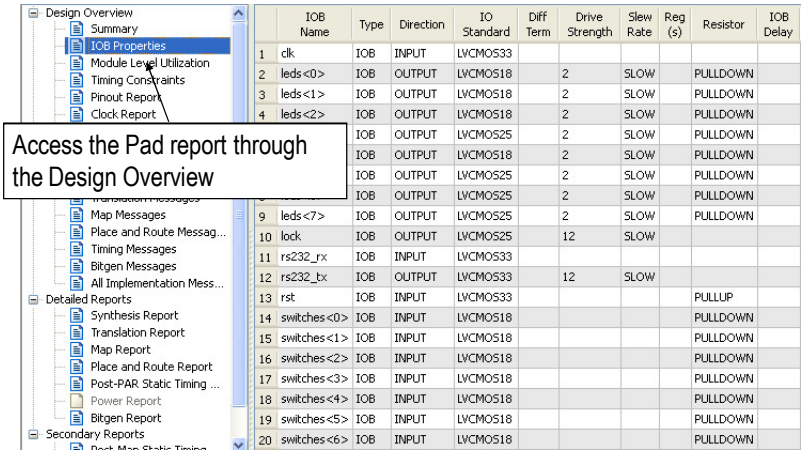


Period Requirement for global clock is 10 ns

Actual Period of 2.025 ns

Pad Report

Get detailed Post-Place & Route Pinout Information



Access the Pad report through the Design Overview

	IOB Name	Type	Direction	IO Standard	Diff Term	Drive Strength	Slew Rate	Reg (s)	Resistor	IOB Delay
1	clk	IOB	INPUT	LVCMOS33						
2	leds<0>	IOB	OUTPUT	LVCMOS18		2	SLOW		PULLDOWN	
3	leds<1>	IOB	OUTPUT	LVCMOS18		2	SLOW		PULLDOWN	
4	leds<2>	IOB	OUTPUT	LVCMOS18		2	SLOW		PULLDOWN	
		IOB	OUTPUT	LVCMOS25		2	SLOW		PULLDOWN	
		IOB	OUTPUT	LVCMOS18		2	SLOW		PULLDOWN	
		IOB	OUTPUT	LVCMOS25		2	SLOW		PULLDOWN	
		IOB	OUTPUT	LVCMOS25		2	SLOW		PULLDOWN	
9	leds<7>	IOB	OUTPUT	LVCMOS25		2	SLOW		PULLDOWN	
10	lock	IOB	OUTPUT	LVCMOS25		12	SLOW			
11	rs232_rx	IOB	INPUT	LVCMOS33						
12	rs232_tx	IOB	OUTPUT	LVCMOS33		12	SLOW			
13	rst	IOB	INPUT	LVCMOS33					PULLUP	
14	switches<0>	IOB	INPUT	LVCMOS18					PULLDOWN	
15	switches<1>	IOB	INPUT	LVCMOS18					PULLDOWN	
16	switches<2>	IOB	INPUT	LVCMOS18					PULLDOWN	
17	switches<3>	IOB	INPUT	LVCMOS18					PULLDOWN	
18	switches<4>	IOB	INPUT	LVCMOS18					PULLDOWN	
19	switches<5>	IOB	INPUT	LVCMOS18					PULLDOWN	
20	switches<6>	IOB	INPUT	LVCMOS18					PULLDOWN	

Outline

- Introduction
- Area Goals
- Performance Goals
- **Summary**

Summary

- A successful implementation means that your design meets your area and performance objectives
- The Map Report provides resource utilization and availability
- The Post-Map Static Timing Report gives you information to create reasonable timing constraints
- The Post-Place & Route Static Timing Report informs you whether or not your timing constraints were met
- The Design Summary screen provides quick access to many reports

Global Timing Constraints

XILINX®

Outline

- **Introduction**
- Global Constraints
- The Constraints Editor
- Summary

XILINX®

Timing Constraints

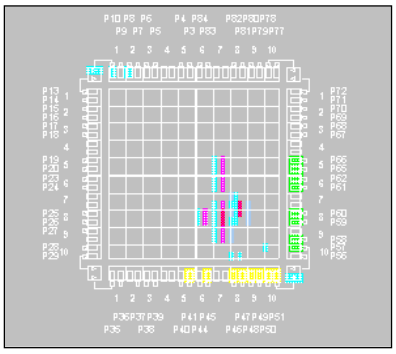
What effects do timing constraints have on your project?

- The implementation tools do not attempt to find the Place & Route that will obtain the best speed
  - Instead, the implementation tools try to meet your performance expectations
- Performance expectations are communicated with timing constraints
  - Timing constraints improve the design performance by placing logic closer together so that shorter routing resources can be used
  - Note: The Constraints Editor refers to the Xilinx Constraints Editor

Design Without and With Constraints

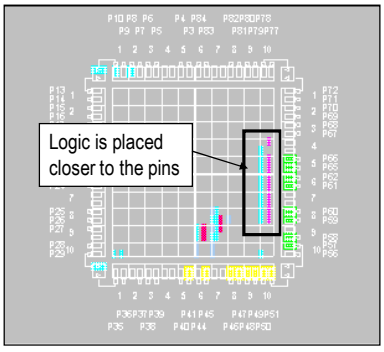
Logic Placement Can Be Very Different

Without global timing constraints



Logic is placed randomly

With global timing constraints (OFFSET)

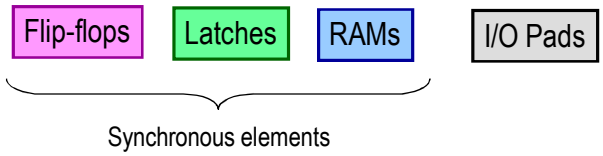


Logic is placed to result in a faster design

Create a Timing Constraint

Create groups of path end points and specify a timing requirement between groups

Step 1: Create groups of path end points



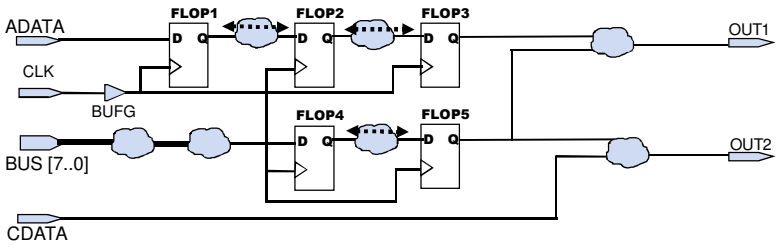
Step 2: Specify a timing requirement between the groups

Outline

- Introduction
- **Global Constraints**
- The Constraints Editor
- Summary
- Lab 3: Global Timing Constraints Lab

Period Constraints

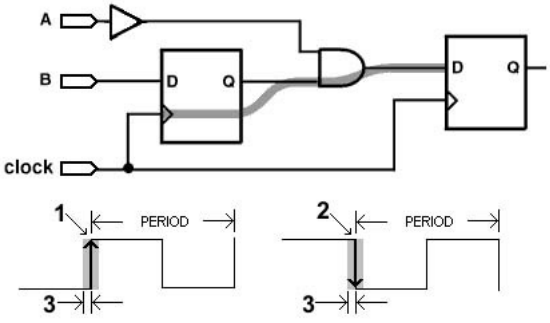
Cover Paths Between Synchronous Elements



PERIOD Constraint

Use the Most Accurate Timing Information

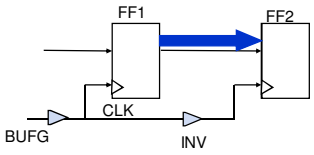
- ✓ Clock skew between the source and destination flip-flops
- ✓ Synchronous elements clocked on the negative edge
- ✓ Unequal clock duty cycles
- ✓ Clock input jitter



PERIOD Constraint

Calculation takes into account inverted clock edges

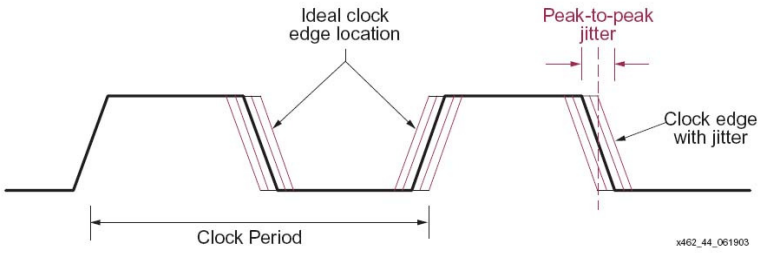
- Assume:
  - 50 percent duty cycle on CLK
  - PERIOD constraint of 10 ns
  - Because FF2 will be clocked on the falling edge of CLK, the path between the two flip-flops will be constrained to 50 percent of 10 ns = 5 ns



Period Constraint

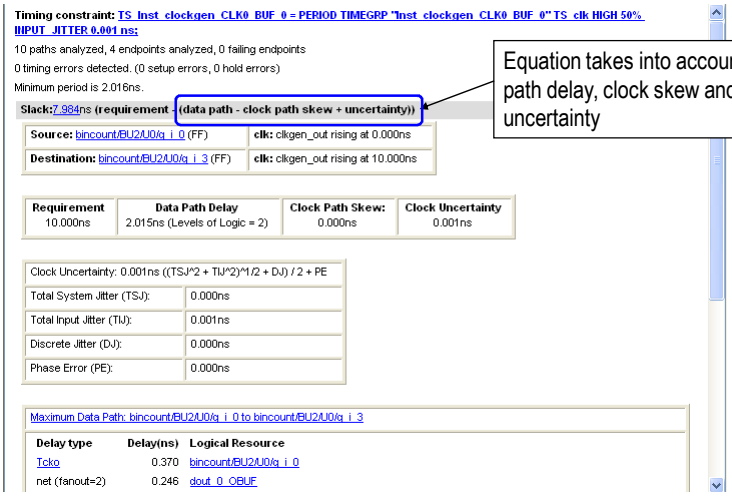
Clock uncertainty is automatically accounted for in global constraint calculations

Clock jitter is a form of clock uncertainty



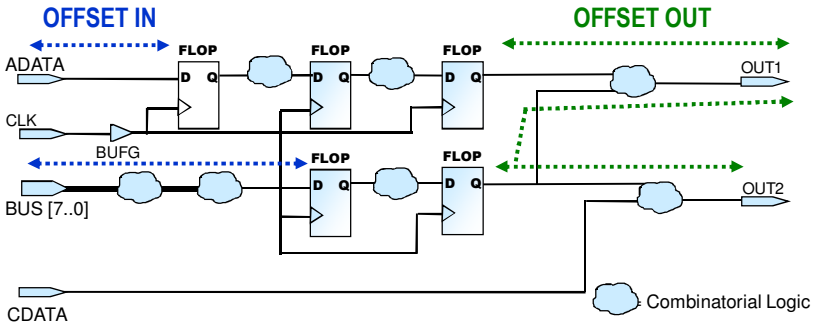
Period Constraint

Timing Analyzer calculation accounts for most accurate timing information



Offset Constraints

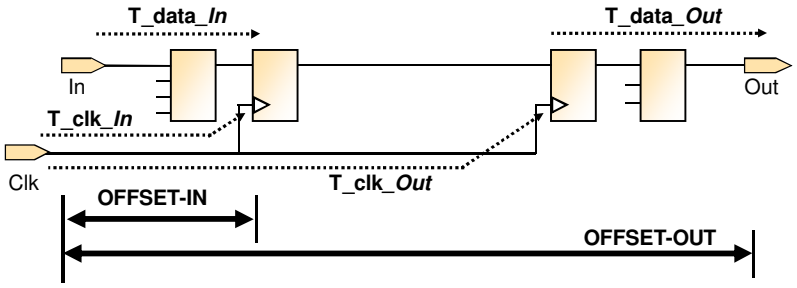
Constrains I/O Pads To/From Synchronous Elements relative to associated clock signal



Offset Constraints

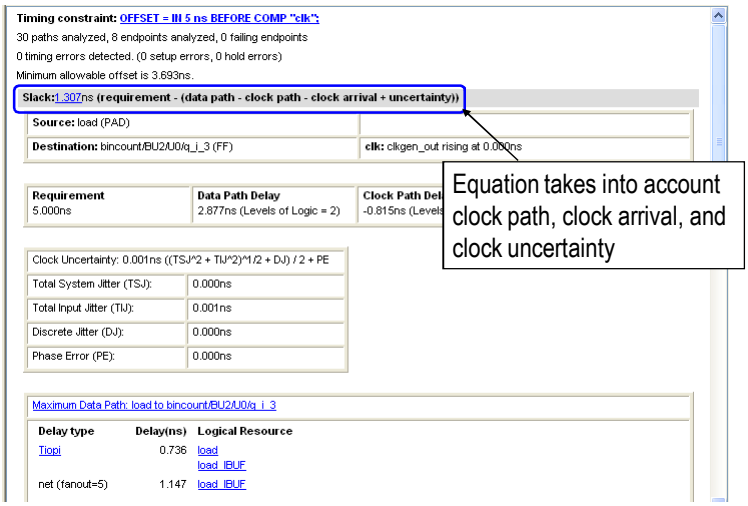
Accounts for Clock Delay

- OFFSET IN = T_data_In - T_clk_In
- OFFSET OUT = T_data_Out + T_clk_Out



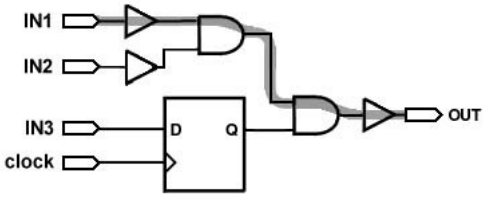
Offset Constraints

Timing Analyzer calculation accounts for most accurate timing information



Pad to Pad Constraints

Covers Purely Combinatorial Paths that start and end at I/O pads



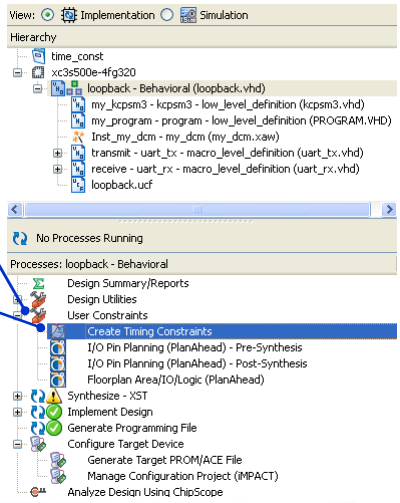
Outline

- Introduction
- Global Constraints
- **The Constraints Editor**
- Summary

Access the Constraints Editor

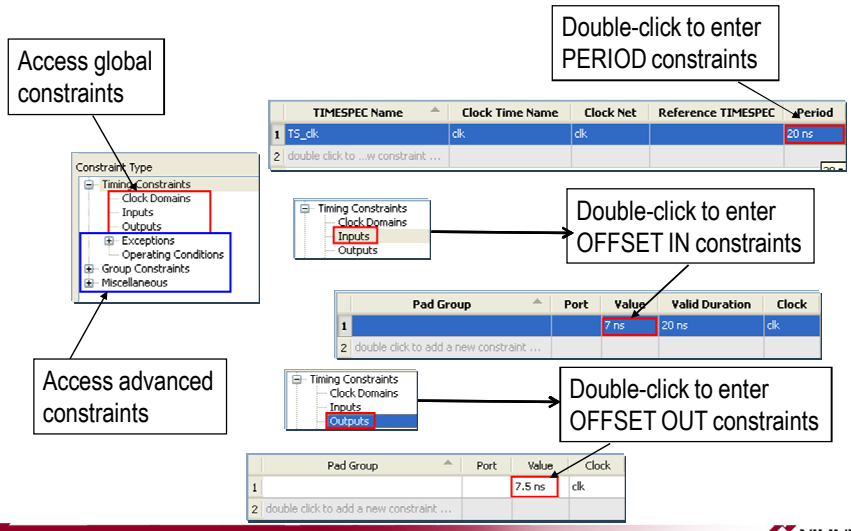
Enter constraints in the Constraints Editor GUI

- Expand **User Constraints** in the Processes for Source window
- Double-click **Create Timing Constraints**



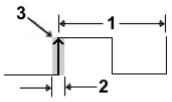
Enter Global Constraints

Specify PERIOD, OFFSET and Pad-to-Pad Constraints

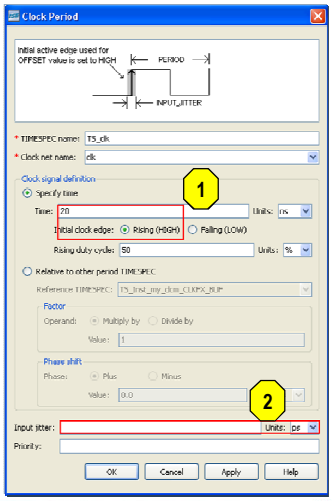


Entering PERIOD Constraint

Can enter text or specify options using the Clock Period GUI

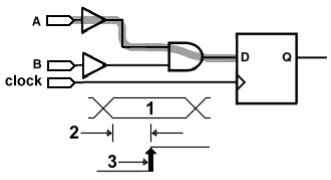


1. Period
2. Input Jitter
3. Default active edge used for OFFSET is RISING

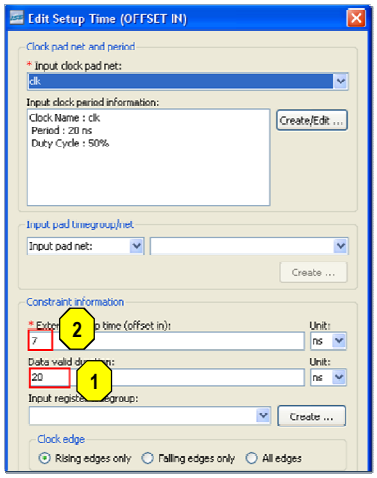


Entering OFFSET IN Constraints

Can enter text or specify options using the OFFSET IN GUI

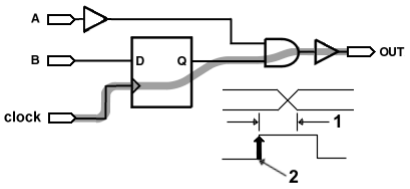


1. Data
2. OFFSET
3. Active edge defined by PERIOD

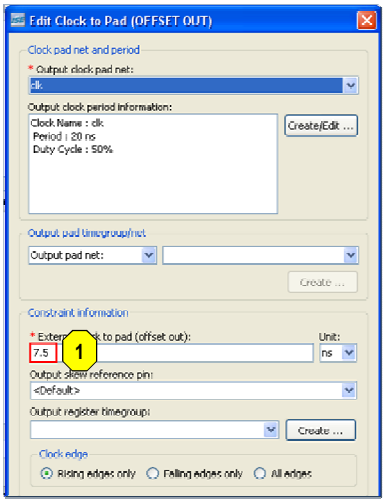


Entering OFFSET OUT Constraints

Can enter text or specify options using the OFFSET OUT GUI



1. OFFSET
2. Active Edge Defined by PERIOD



Outline

- Introduction
- Global Constraints
- The Constraints Editor
- **Summary**

Summary

- Performance expectations are communicated with timing constraints
- PERIOD constraints cover delay paths between synchronous elements
- OFFSET constraints cover delay paths from input pins to synchronous elements and from synchronous elements to output pins
- Use the Constraints Editor to create timing constraints

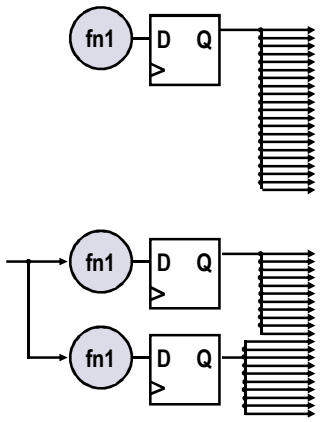
FPGA Design Techniques

Outline

- **Duplicating Flip-Flops**
- Pipelining
- I/O Flip-Flops
- Synchronization Circuits
- Summary

Duplicating Flip-Flops

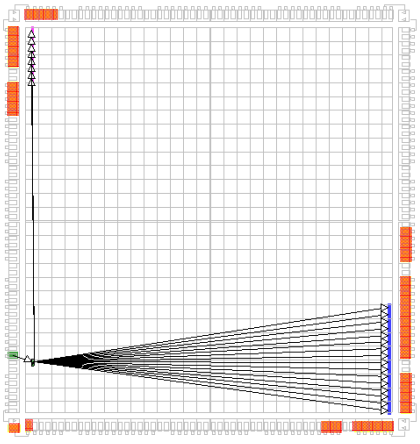
- High-fanout nets can be slow and hard to route
- Duplicating flip-flops can fix both problems
  - Reduced fanout shortens net delays
  - Each flip-flop can fanout to a different physical region of the chip to reduce routing congestion
- Design trade-offs
  - Gain routability and performance
  - Increase design area
  - Increase fanout of other nets



Duplicating Flip-Flops Example

The source flip-flop drives two register banks constrained to different regions of the chip

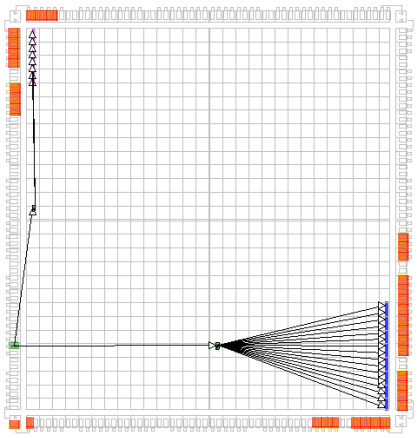
- The source flip-flop and pad are not constrained
- PERIOD = 5 ns timing constraint
- Implemented with default options
- Longest path = 6.806 ns
  - Fails to meet timing constraint



Duplicating Flip-Flops Example

The source flip-flop has been duplicated

- Each flip-flop drives a region of the chip
  - Each flip-flop can be placed closer to the register that it is driving
  - Shorter routing delays
- Longest path = 4.666 ns
  - Meets timing constraint



Tips on Duplicating Flip-Flops

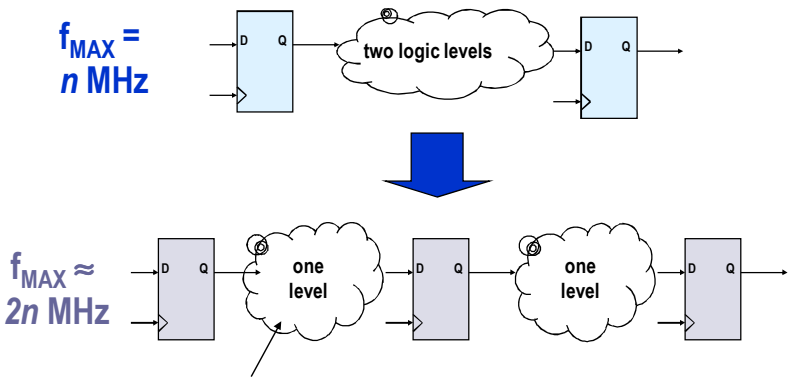
- Name duplicated flip-flops _a, _b; *NOT* _1, _2
  - Numbered flip-flops are mapped into the same slice by default
  - Duplicated flip-flops should be separated
    - Especially if the loads are spread across the chip
- Explicitly create duplicate flip-flops in your HDL code
  - Most synthesis tools have automatic fanout-control features
    - However, they do not always pick the best division of loads
    - Also, duplicated flip-flops will be named _1, _2
  - Many synthesis tools will optimize-out duplicated flip-flops
    - Set your synthesis tool to keep redundant logic
- Do not duplicate flip-flops that are sourced by asynchronous signals
  - Synchronize the signal first
  - Feed the synchronized signal to multiple flip-flops

Outline

- Duplicating Flip-Flops
- **Pipelining**
- I/O Flip-Flops
- Synchronization Circuits
- Summary

Pipelining Concept

Inserting flip-flops into a data path increases performance



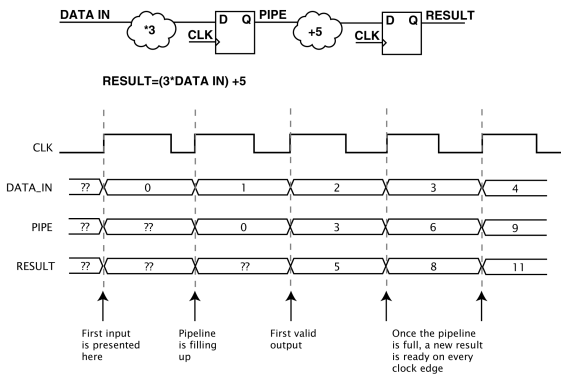
Reduces the number of logic levels (LUTs) between flip-flops

Pipelining Considerations

- Are enough flip-flops available?
  - Refer to the MAP Report
  - In general, you will not run out of flip-flops
- Are there multiple logic levels between flip-flops?
  - If there is only one logic level between flip-flops, pipelining will not improve performance
  - Refer to the Post-Map Static Timing Report or Post-Place & Route Static Timing Report
- Can the system tolerate *latency*?

Latency in Pipelines

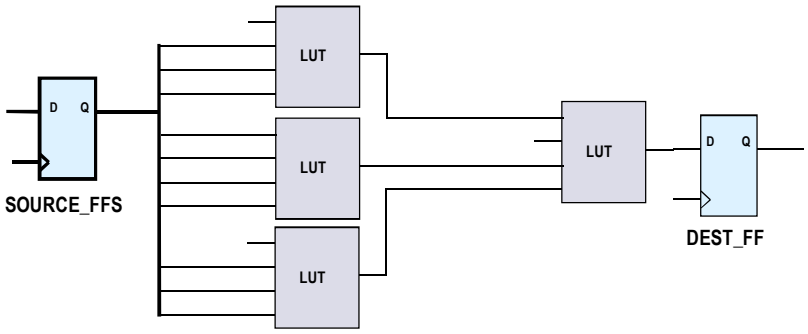
- Each pipeline stage adds one clock cycle of delay before the first output will be available
  - Also called "filling the pipeline"
- After the pipeline is filled, a new output is available every clock cycle



Pipelining Example

Circuit before pipelining

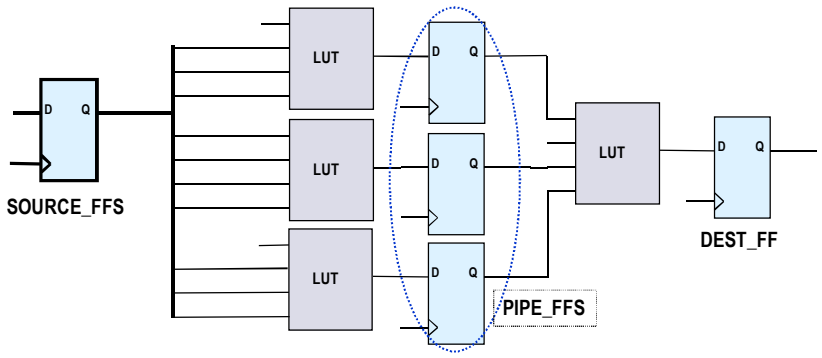
- Original circuit
  - Two logic levels between SOURCE_FFS and DEST_FF
  - f_{MAX} = ~233 MHz



Pipelining Example

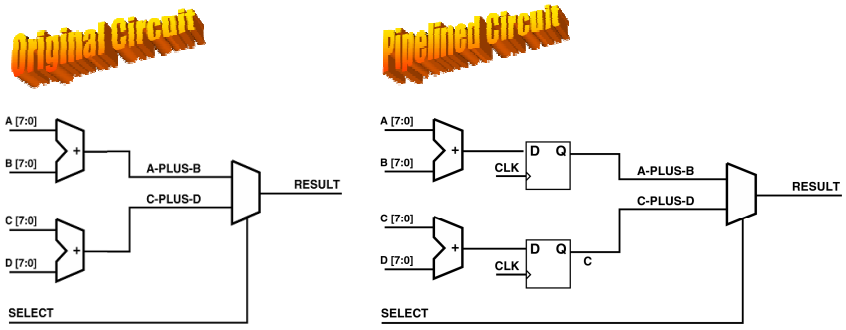
Circuit after pipelining

- Pipelined circuit
  - One logic level between each set of flip-flops
  - f_{MAX} = ~385 MHz



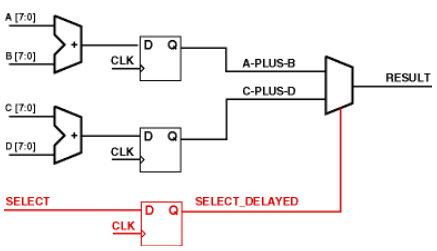
Review Questions

- Given the original circuit, what is wrong with the pipelined circuit?
- How can the problem be corrected?



Answers

- What is wrong with the pipelined circuit?
  - Latency mismatch
  - Older data is mixed with newer data
  - Circuit output is incorrect
- How can the problem be corrected?
  - Add a flip-flop on *SELECT*
  - All data inputs now experience the same amount of latency



Outline

- Duplicating Flip-Flops
- Pipelining
- **I/O Flip-Flops**
- Synchronization Circuits
- Summary

I/O Flip-Flop Overview

- Each I/O block contains six flip-flops
  - IN FF on the input, OUT FF on the output, EN FF on the 3-state enable*
  - Single data rate or double data rate support
- I/O flip-flops provide guaranteed setup, hold, and clock-to-out times when the clock signal comes from a BUFG

Accessing I/O Flip-Flops

- During synthesis
  - Timing-driven synthesis can force flip-flops into Input/Output Blocks (IOBs)
  - Some tools support attributes or synthesis directives to mark flip-flops for placement in an IOB
- Xilinx Constraint Editor
  - Select the Misc tab and specify registers that should be placed into IOBs
    - You need to know the instance name for each register
- During the MAP phase of implementation
  - In the Map Properties dialog box, the Pack I/O Registers/Latches into IOBs option is selected by default
  - Timing-driven packing will also move registers into IOBs for critical paths
- Check the MAP Report to confirm that IOB flip-flops have been used
  - IOB Properties section

Outline

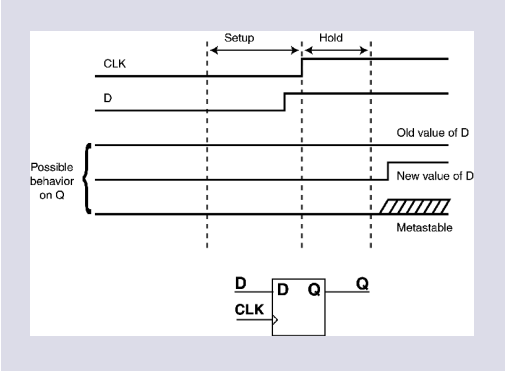
- Duplicating Flip-Flops
- Pipelining
- I/O Flip-Flops
- **Synchronization Circuits**
- Summary

Synchronization Circuits

- What is a synchronization circuit?
  - Captures an asynchronous input signal and outputs it on a clock edge
- Why do you need synchronization circuits?
  - To prevent setup and hold time violations
  - To ensure a more reliable design
- When do you need synchronization circuits?
  - Signals cross between *unrelated* clock domains
    - Between related clock domains, relative PERIOD constraints are sufficient
  - Chip inputs that are asynchronous

Setup and Hold Time Violations

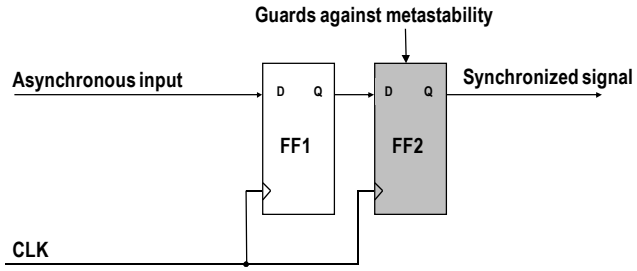
- Violations occur when the flip-flop input changes too close to a clock edge
- Three possible results:
  - Flip-flop clocks in an old data value
  - Flip-flop clocks in a new data value
  - Flip-flop output becomes *metastable*



Synchronization Circuit 1

Use when input pulses will always be at least one clock period wide

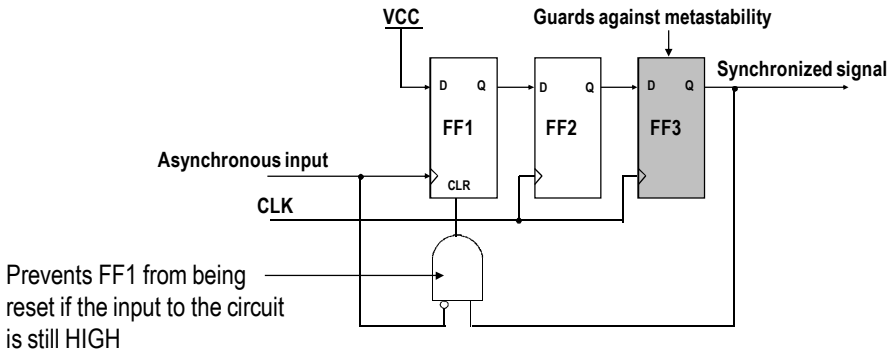
- The “extra” flip-flops guard against metastability



Synchronization Circuit 2

Use when input pulses may be less than one clock period wide

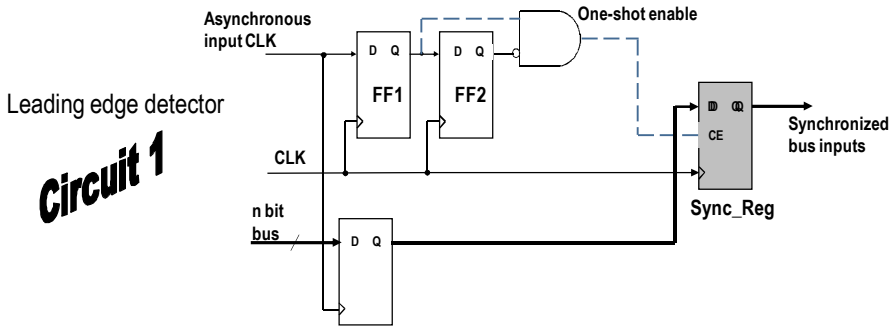
- FF1 captures short pulses
- FF2 and FF3 act in the same way as the Synchronization 1 circuit



Capturing a Bus

Use when input pulses must be at least one CLK period wide

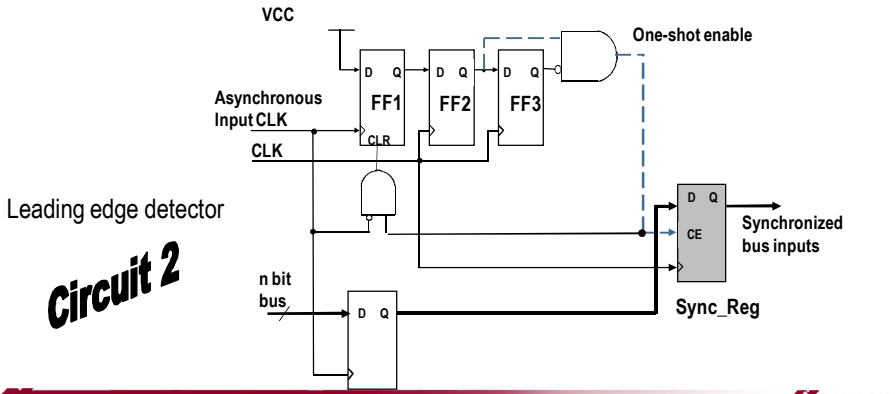
- First, the data bus is registered by the asynchronous clock.
- Second, the one-shot enable (synchronized to CLK) signals to the internal circuit that data is captured (via CE).



Capturing a Bus

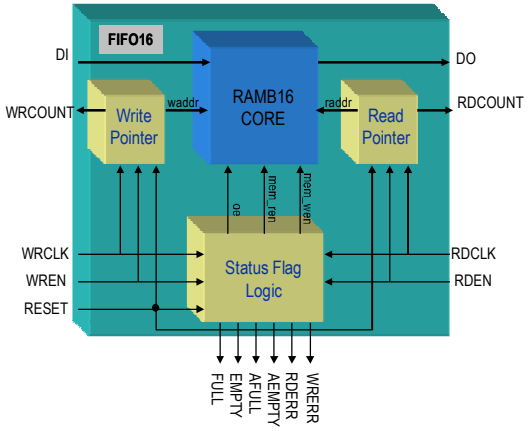
Use when Input pulses may be less than one CLK period wide

- First, the data bus is registered by the asynchronous clock.
- Second, the one-shot enable (synchronized to CLK) signals to the internal circuit that data is captured (via CE).



Synchronization Circuit 3

- Use a FIFO to cross domains



Outline

- Duplicating Flip-Flops
- Pipelining
- I/O Flip-Flops
- Synchronization Circuits
- **Summary**

Summary

- You can increase circuit performance by:
  - Duplicating flip-flops
  - Adding pipeline stages
  - Using I/O flip-flops
- Some trade-offs
  - Duplicating flip-flops increases circuit area
  - Pipelining introduces latency and increases circuit area
- Synchronization circuits increase reliability

Synchronous Design Techniques

XILINX®

Objectives

After completing this module, you will be able to:

- Use the hierarchy effectively
- Increase the circuit reliability and performance by applying synchronous design techniques

XILINX®

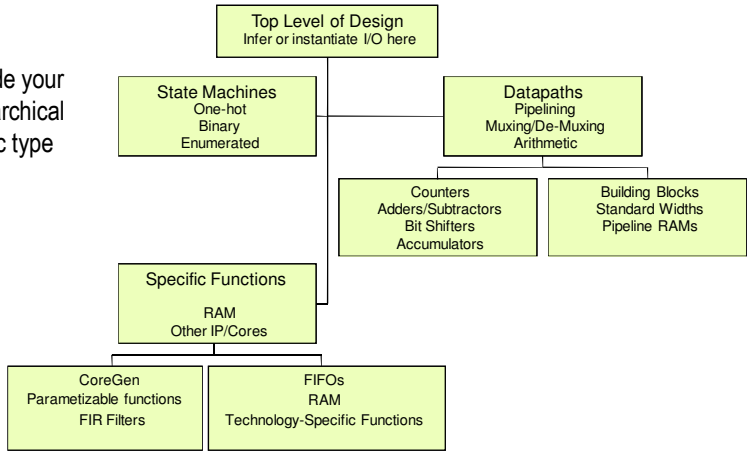
Outline

- **Hierarchical Design**
- Synchronous Design for Xilinx FPGAs
- Summary

Hierarchical Design

Using hierarchy leads to greater design readability, reuse, and debug

One way to divide your design into hierarchical blocks is by logic type



Benefits of Using Hierarchy

- Design readability
  - Easier to understand the design functionality and data flow
  - Easier to debug
- Easy to reuse parts of a design

Design Readability Tips

- Choose hierarchical blocks that have the following:
  - Logical data flow between blocks
  - Minimum of routing between blocks
- Choose descriptive labels for blocks and signals
- Keep clock domains separated
  - Makes the interaction between clocks very clear
- Keep the length of each source file manageable
  - Easier to read, synthesize, and debug

Design Reuse Tips

- Build a set of blocks that are available to all designers
  - Register banks
  - FIFOs
  - Other standard functions
  - Custom functions commonly used in your applications
- Name blocks by function and target Xilinx family
  - Easy to locate the block you want
  - Example: REG_4X8_S3 (bank of four 8-bit registers targeting Spartan-3)
- Store in a directory that is separate from the Xilinx tools
  - Prevents accidental deletion when updating tools

Outline

- Hierarchical Design
- **Synchronous Design for Xilinx FPGAs**
- Summary

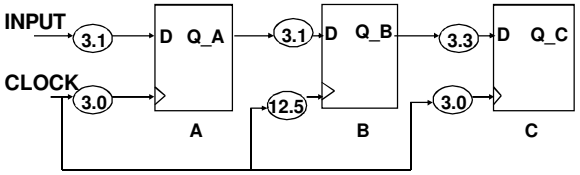
Why Synchronous Design?

- Synchronous circuits are more reliable
  - Events are triggered by clock edges that occur at well-defined intervals
  - Outputs from one logic stage have a full clock cycle to propagate to the next stage
    - Skew between data arrival times is tolerated within the same clock period
- Asynchronous circuits are less reliable
  - Delay may need to be a specific amount (for example, 12 ns)
  - Multiple delays may need to hold a specific relationship (for example, DATA arrives 5 ns before SELECT)

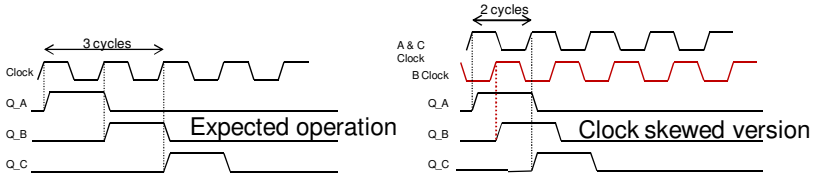
Asynchronous Design
Case Studies

- The design I created two years ago no longer works. What did Xilinx change in their FPGAs?
  - SRAM process improvements and geometry shrinks increase speed
  - Normal variations between wafer lots
- My design passes a timing simulation test but fails in-circuit. Is the timing simulation accurate? **YES**
  - Timing simulation uses worst-case delays
  - Actual board-level conditions are usually better

Clock Skew



- This shift register will not work because of clock skew!



Use Global Buffers to Reduce Clock Skew

- Global buffers are connected to dedicated routing
  - This routing network is balanced to minimize skew
- All Xilinx FPGAs have global buffers
  - Spartan™-3 devices have 8 BUFGMUXes
  - Virtex™-5 devices have 32 BUFGCTRLs
  - Spartan-6 devices have 16 BUFGMUXes
- You can always use a BUFG symbol, and the software will choose an appropriate buffer type
  - All major synthesis tools can infer global buffers onto clock signals that come from off-chip

Using Global Lines

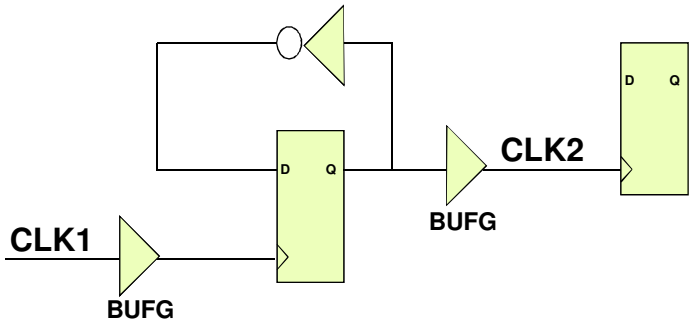
- Most synthesis tools will automatically infer a BUFG on clocks
  - Clock signals must come from a top-level port
  - Internally generated clocks are not automatically placed on a BUFG
- Sample instantiation of BUFGMUX (Verilog)

```
BUFGMUX  U_BUFGMUX
(.I0( ),  // insert clock input used when select(S) is Low
.I1( ),  // insert clock input used when select(S) is High
.S( ),   // insert Mux-Select input
.O( )    // insert clock output
      );
```

- Note: These templates, and templates for other BUFGMUX configurations, are available in the ISE™ language templates

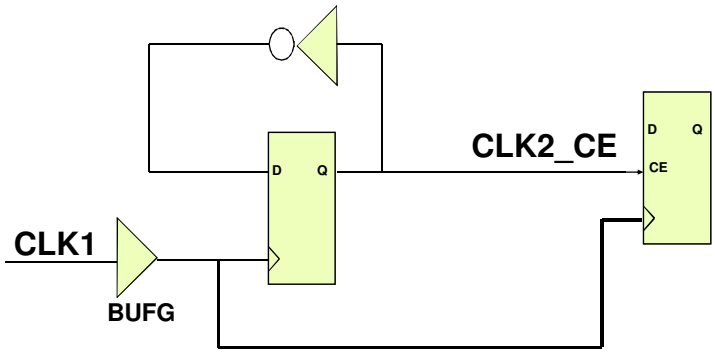
Traditional Clock Divider

- Introduces clock skew between CLK1 and CLK2
- Uses an extra BUFG to reduce skew on CLK2



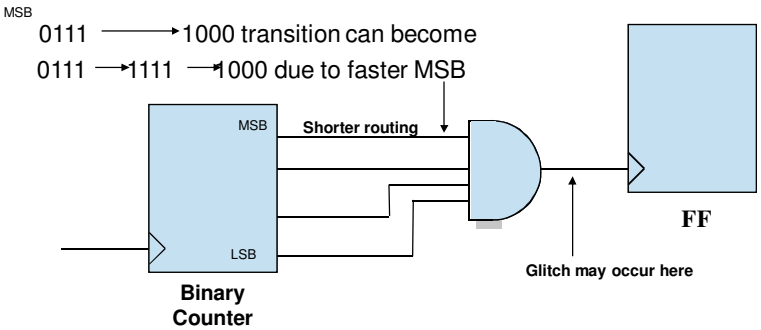
Recommended Clock Divider

- No clock skew between flip-flops



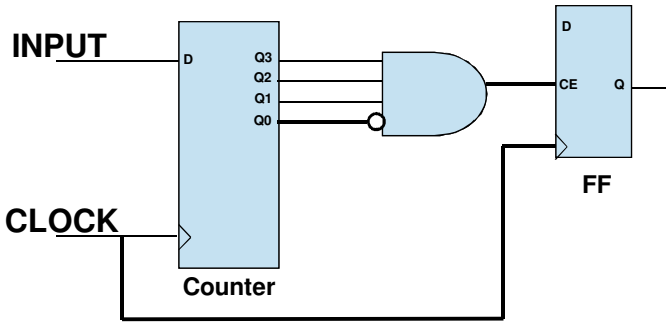
Avoid Clock Glitches

- Because flip-flops in today's FPGAs are very fast, they can respond to very narrow clock pulses
- Never source a clock signal from combinatorial logic
  - Also known as "gating the clock"



Avoid Clock Glitches

- This circuit creates the same function, but it creates the function without glitches on the clock



Coding Clock Enables

May infer extra logic if you do not follow the correct order of precedence

VHDL

```
FF_AR_CE: process(CLK)
begin
 if (CLK'event and CLK = '1') then
  if (ENABLE = '1') then
   Q <= D_IN;
  end if;
 end if;
end process
```

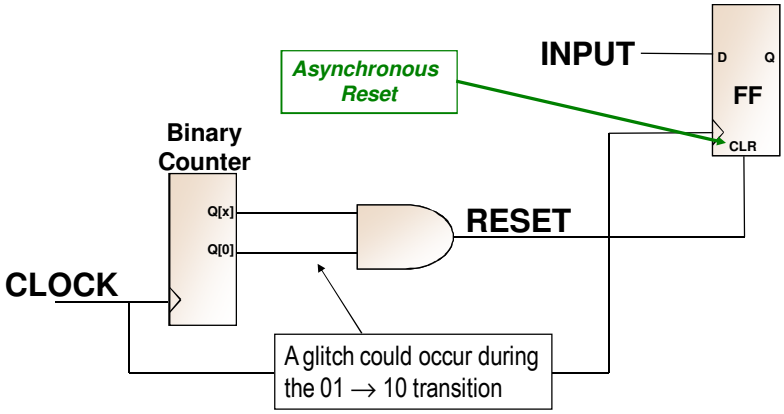
Verilog

```
always @(posedge CLOCK)
 if (ENABLE)
  Q = D_IN;
```

- If ENABLE is not a top-level port, write the code for ENABLE in another process
  - Makes the code more readable
  - Helps the synthesis tools create better netlists
- Control signal precedence: Reset, Set, Enable
  - Use this order in your code

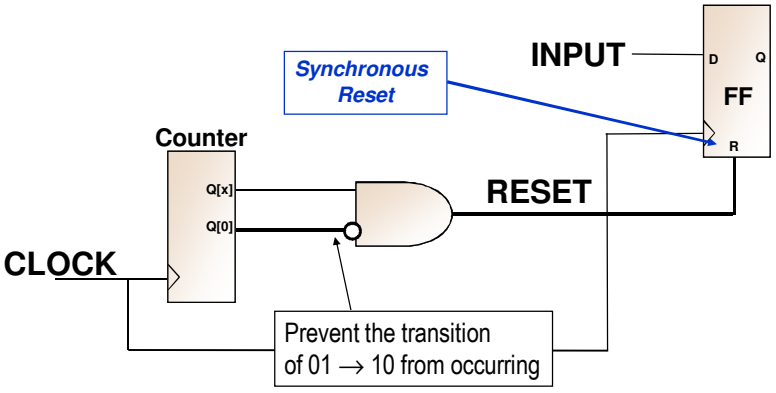
Avoid Set/Reset Glitches

- Glitches on asynchronous set and asynchronous reset inputs can lead to incorrect circuit behavior



Avoid Set/Reset Glitches

- Convert to synchronous set and synchronous reset when possible



Coding Synchronous Flip-Flops

- Asynchronous Reset

```
always @(posedge CLOCK or posedge
    RESET)
    if (RESET)
        Q = 0;
    else
        Q = D_IN;
```

- Synchronous Reset

```
always @(posedge CLOCK)
    if (RESET)
        Q = 0;
    else
        Q = D_IN;
```


Outline

- Hierarchical Design
- Synchronous Design for Xilinx FPGAs
- **Summary**

Summary

- Proper use of the hierarchy aids design readability and debug
- Synchronous designs are more reliable than asynchronous designs
- FPGA design tips
  - Global clock buffers and DLLs eliminate skew
  - Avoid glitches on clocks, asynchronous sets, and asynchronous resets

Synthesis Techniques

Outline

- **Coding Tips**
- Instantiating Resources
- Synthesis Options
- Summary

Simple Coding Steps Yield 3x Performance

- Use pipeline stages—more bandwidth
- Use synchronous reset—better system control
- Use Finite State Machine optimizations
- Use inferable resources
  - Multiplexer
  - Shift Register LUT (SRL)
  - Block RAM, LUT RAM
  - Cascade DSP
- Avoid high-level constructs (loops, for example) in code
  - Many synthesis tools produce slow implementations

Synthesis Guidelines

- Use timing constraints
  - Define tight but realistic individual clock constraints
  - Put unrelated clocks into different clock groups
- Use proper options and attributes
  - Turn off resource sharing
  - Move flip-flops from IOBs closer to logic
  - Turn on FSM optimization
  - Use the *retiming* option

Basic Performance Tips

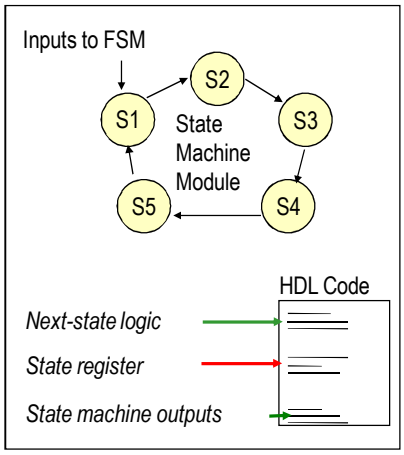
- Avoid high-level loop constructs
  - Synthesis tools may not produce optimal results
- Avoid nested if-then-else statements
  - Most tools implement these in parallel; however, multiple nested if-then-else statements can result in priority encoded logic
- Use case statements for large decoding
  - Rather than if-then-else
- Order and group arithmetic and logical functions and operators
  - A <= B + C + D + E; should be: A <= (B + C) + (D + E)
- Avoid inadvertent latch inference
  - Cover all possible outputs in every possible branch
    - Easily done by making default assignments before if-then-else and case

Targeting Xilinx

- Some resources must be instantiated (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 or 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 through the STARTUP block
    - No need for asynchronous initialization power-on reset

State Machine Design

- Put the next-state logic in one CASE statement
  - The state register can also be included here or in a separate process block or always block
- Put the state machine outputs in a separate process or always block
  - Prevents resource sharing, which can hurt performance



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
  - Experiment to discover how your synthesis tool chooses the default encoding scheme
- Register state machine outputs for higher performance

Coding Flip-Flops

Control signal precedence: Clear, Preset, Clock Enable

- The examples shown here implement synchronous reset and set

VHDL

```
FF_AR_CE: process(CLK)
begin
if (CLK'event and CLK = '1') then
  if (RST = '1') then Q <= '0';
  elsif (SET = '1') then Q <= '1';
  elsif (CE = '1') then
   Q <= D_IN;
  end if;
end if;
end process;
```

Verilog

```
always @(posedge CLK)
 if (RST)
  Q <= 1'b0;
 else if (SET)
  Q <= 1'b1;
 else if (CE)
  Q <= D_IN;
```


Outline

- Coding Tips
- **Instantiating Resources**
- Synthesis Options
- Summary

Instantiation versus Inference

- Xilinx recommends inferring FPGA resources whenever possible
  - Inference makes your code more portable
- In some cases, the synthesis tool is unable to infer or fails to infer resources
  - You can instantiate resources when you must dictate exactly which resource is needed
- Xilinx and Alliance Core partners offer IP cores that you can instantiate in your design
  - ISE includes the CORE Generator™ software system to create functions such as Arithmetic Logic Units (ALUs), fast multipliers, and Finite Impulse Response (FIR) filters for instantiation

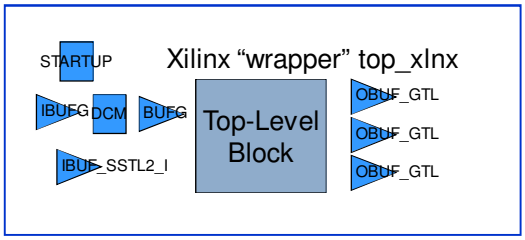
FPGA Resources

- Can be inferred by all synthesis tools:
  - Shift register LUT (SRL16/ SRLC16)
  - F5, F6, F7, and F8 multiplexers
  - Carry logic
  - MULT_AND
  - MULT18x18/MULT18x18S
  - 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 (ROM/RAM)
  - Global clock buffers (BUFGCE, BUFGMUX, BUFGDLL)
- Cannot be inferred by any synthesis tools:
  - SelectIO (differential) standard
  - Output DDR registers
  - DCM
  - Gigabit transceivers

Suggested Instantiation

Xilinx recommends that you isolate instantiated resources in the design

- 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 Xilinx-specific instantiations



Outline

- Coding Tips
- Instantiating Resources
- **Synthesis Options**
- Summary

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
- XST Constraints are entered in a .xcf file

Timing-Driven Synthesis

- 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

FSM Extraction

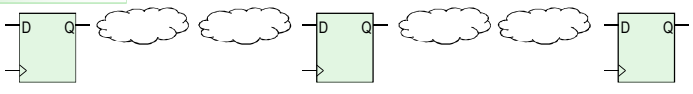
- Finite State Machine (FSM) extraction optimizes your state machine by re-encoding and optimizing your design based on the number of states and inputs
  - By default, the tools will use FSM extraction
- Safe state machines
  - By default, the synthesis tools will remove all decoding for illegal states
    - Even if you include VHDL “when others” or Verilog “default” cases
  - The “safe” FSM implementation option must be turned on

Retiming

- Retiming: The synthesis tool ca try to move register stages to balance combinatorial delay on each side of the registers provided the option is turned ON
  - Forward, reverse, or both directions optimization

Before Retiming

After Retiming



Register Duplication

- Register duplication is used to reduce fanout on registers (to improve 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
    - This 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

Hierarchy Management

- The basic settings are:
  - No: The design will be flattened
    - Allows total combinatorial optimization across all boundaries
  - Yes: The design hierarchy will be maintained
    - Preserves hierarchy without allowing optimization of combinatorial logic across boundaries
  - Soft: The design hierarchy is maintained at synthesis process level but is allowed to be flattened at the implementation process

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 no need to flatten

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

Increasing Productivity

- Use synchronous design techniques
- Preserve hierarchy during synthesis
  - Aids in debugging and cross-referencing to report files
- Use timing-driven synthesis if your tool supports it
  - Check the synthesis report for performance estimates
- After implementation, look at timing reports and identify critical paths
  - Double-check the HDL coding style for these paths
  - Try some of the synthesis options discussed earlier
- For paths that continually fail to meet timing, add path-specific constraints during synthesis
  - Add corresponding path-specific constraints for implementation

Outline

- Coding Tips
- Instantiating Resources
- Synthesis Options
- **Summary**

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

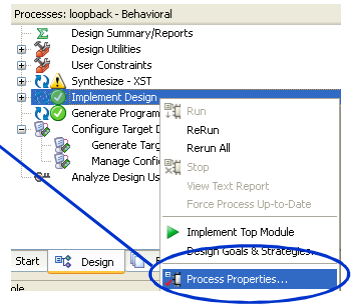
Implementation Options

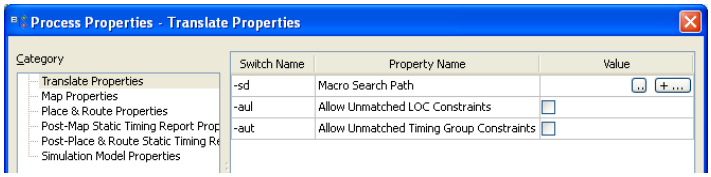
Outline

- **Basic Software Options**
- Accessing Advanced Software Options
- Summary

Accessing Implementation Options

- Right-click **Implement Design** and select **Process Properties…**
  - Or right-click the **specific process**
- This dialog has six categories

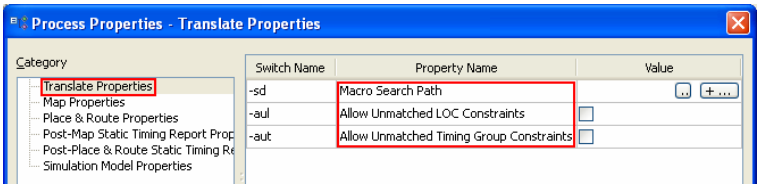




Translate Properties

Standard options

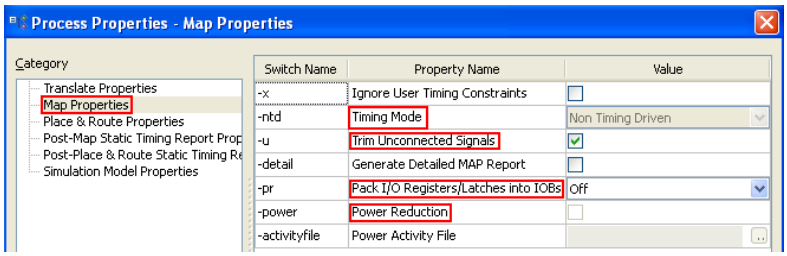
- Specify netlist libraries
  - Use the Macro Search Path to tells the tools where to search for your own netlist libraries of commonly used components, such as cores.
- Target the netlist to a new device or package
  - The Allow Unmatched LOC constraints option ignores invalid location constraints



Map Properties

Standard options

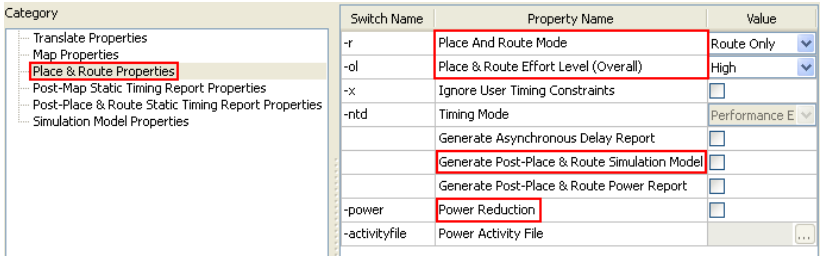
- Improve performance
  - Timing mode
  - Pack I/O Registers/Latches into IOBs
- Improve area
  - Trim Unconnected Signals
    - Uncheck for incomplete designs
- Improve power
  - Power reduction mode
    - Uses a power activity file for guiding placement



Place & Route Properties

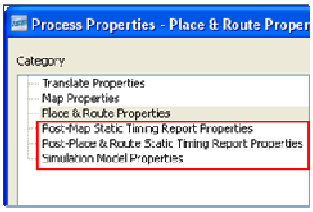
Standard options

- **Improve performance**
  - **General**
    - **Place & Route Effort Level**
      - **Standard, medium, high**
    - **Place & Route Mode**
      - **Place only, route only, Normal, Reentrant**
- **Generate simulation model**
  - **Post-Place & Route**
- **Improve power**
  - **Power reduction**
  - **Power activity file**



Other Properties

- Post-map static timing report
- Post-place & route static timing report
- Simulation model properties
  - Set simulation model target as VHDL or Verilog
  - Specify option to retain hierarchy

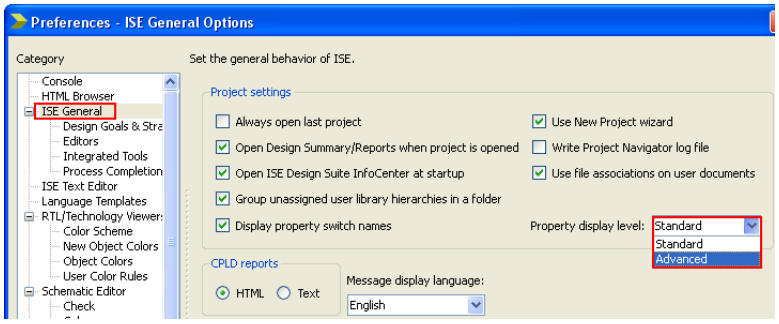


Outline

- Basic Software Options
- **Accessing Advanced Software Options**
- Summary

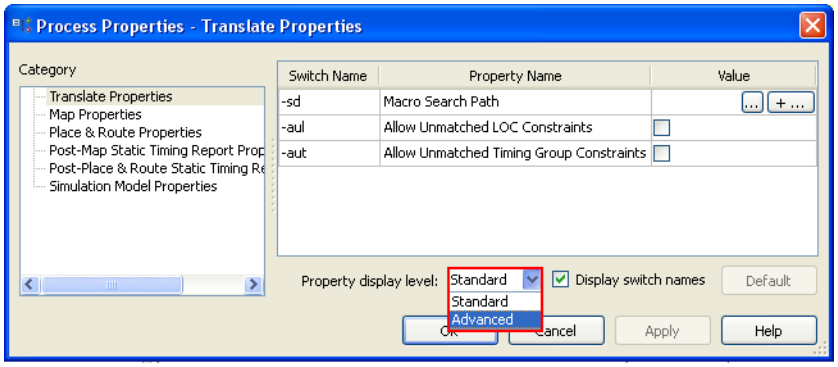
Viewing Advanced Options: Globally

Select **Edit → Preferences**



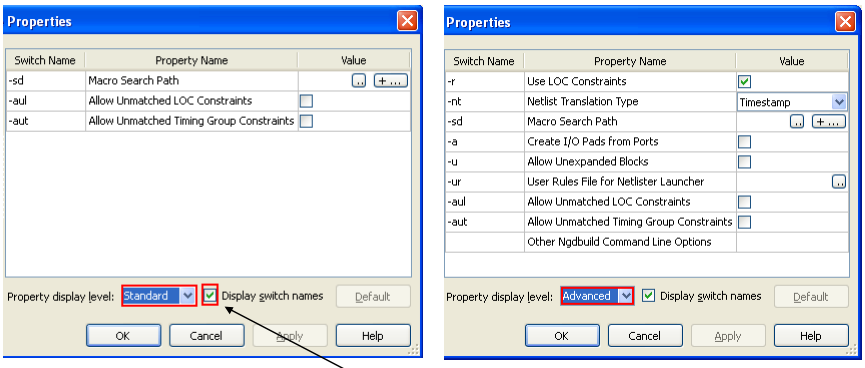
Viewing Advanced Options

From any Process Properties change from Standard to Advanced



Standard Versus Advanced Properties

Advanced display level shows more options



You can turn-off switch names display by unchecking the box

Outline

- Basic Software Options
- Accessing Advanced Software Options
- **Summary**

Summary

- Software options can improve performance, power, and area
- Software options are easy to access
  - View both standard and advanced properties to access software options

CORE Generator System

XILINX®

Outline

- **Introduction**
- Using the CORE Generator System
- CORE Generator Design Flows
- Summary

XILINX®

What are Cores?

- A *core* is a ready-made function that you can instantiate into your design as a *black box*
- Cores can range in complexity
  - Simple arithmetic operators, such as adders, accumulators, and multipliers
  - System-level building blocks, such as filters, transforms, and memories
  - Specialized functions, such as bus interfaces, controllers, and microprocessors
- Some cores can be customized

Benefits of Using Cores

- Save design time
  - Cores are created by expert designers who have in-depth knowledge of Xilinx FPGA architecture
  - Guaranteed functionality saves time during simulation
- Increase design performance
  - Cores that contain mapping and placement information have predictable performance that is constant over device size and utilization
  - The data sheet for each core provides performance expectations
    - Use timing constraints to achieve maximum performance

LogiCORE Solutions

- Typically customizable
- Fully tested, documented, and supported by Xilinx
- Many are pre-placed for predictable timing
- Many are unlicensed and provided for free with Xilinx software
  - More complex LogiCORE™ solution products are licensed
- VHDL and Verilog flow support for several EDA tools
- Schematic flow support for most cores

Outline

- Introduction
- **Using the CORE Generator System**
- CORE Generator Design Flows
- Summary

The CORE Generator System

- A Graphical User Interface (GUI) allows central access to the cores themselves, as well as:
  - Data sheets
  - Customizable parameters (available for some cores)
- Interfaces with design entry tools
  - Creates graphical symbols for schematic-based designs
  - Creates instantiation templates for HDL-based designs
- Web Links tab provides access to the Xilinx Website and the IP Center
  - The IP Center contains new cores to download and install
    - You always have access to the latest cores

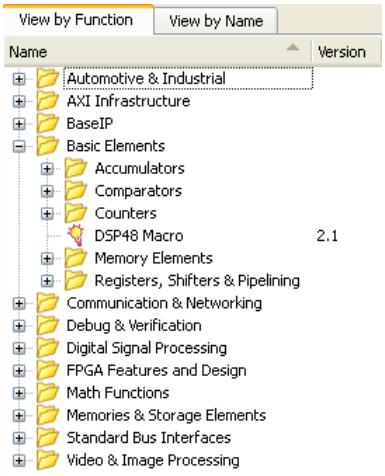
Invoking CORE Generator

Can access from within ISE using the New Source Wizard

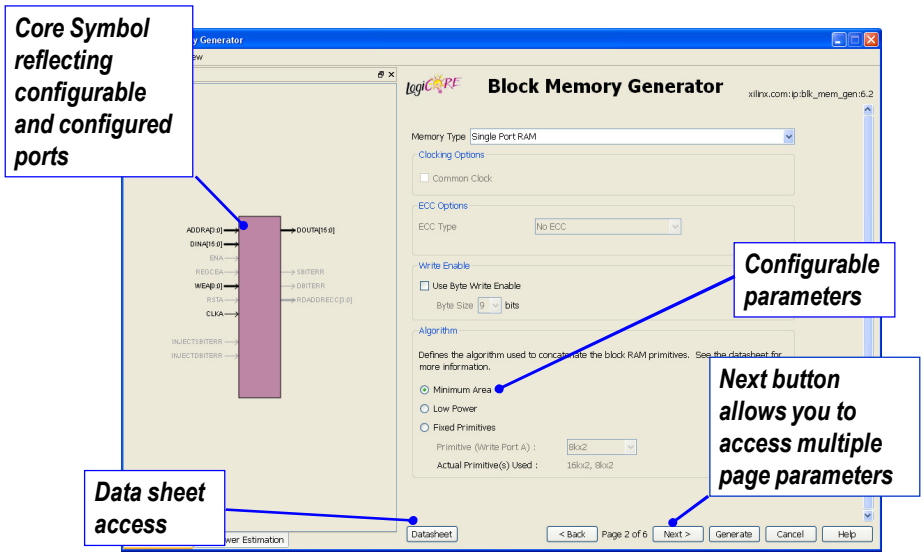
- From the Project Navigator, select **Project → New Source**
- Select **IP (CoreGen & Architecture Wizard)** and enter a ***filename***
- Click **Next** and then select the **type of core**

View Cores

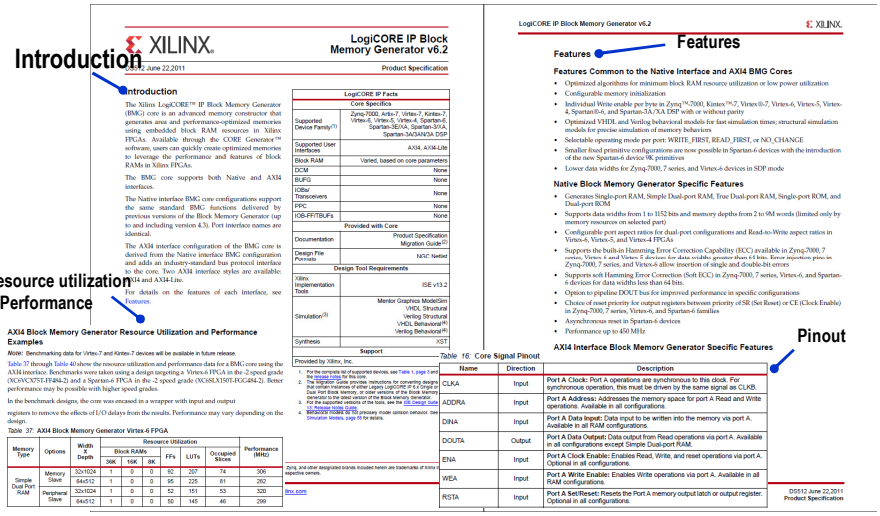
- View cores by Function
- View cores by Name



Core Customize Window



CORE Data Sheets



Outline

- Introduction
- Using the CORE Generator System
- **CORE Generator Design Flows**
- Summary

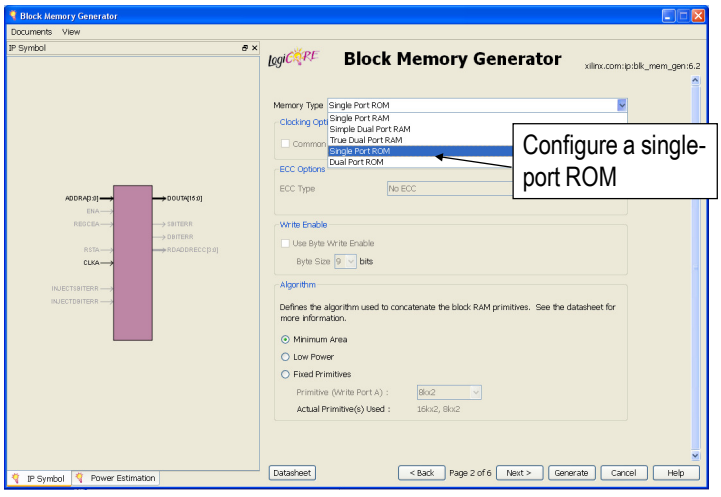
HDL Design Flow

Generate and integrate cores

- Generate a core
  - Netlist file (EDN, NGO)
  - Instantiation template files (VHO or VEO)
  - Behavioral simulation wrapper files (VHD or V)
- Instantiate the core
  - Cut and paste from the templates provided in the VEO or VHO file
- Perform a behavioral simulation
  - The ISE™ software automatically uses wrapper files when cores are present in the design
  - VHDL: Analyze the wrapper file for each core before analyzing the file that instantiates the core
- Synthesize and implement the design

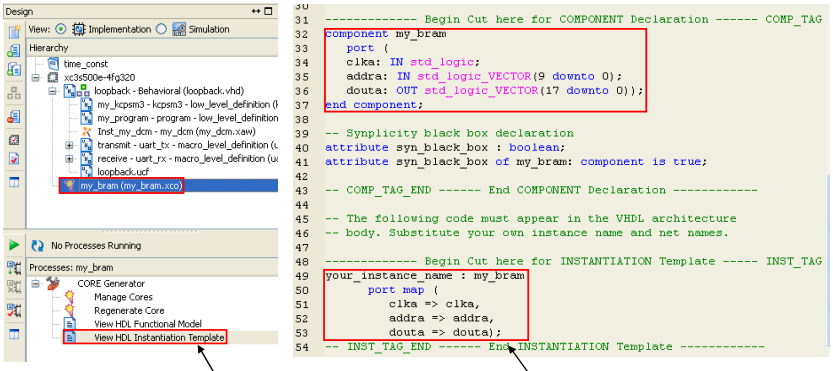
Generate Core

Parameterize the core using the GUI and generate the project files



Instantiate Core

Access HDL instantiation templates (VHO and VEO) from Processes for the Core

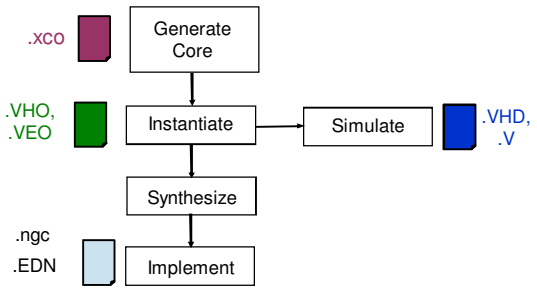


Select the core in the Hierarchy window, double-click View HDL Instantiation Template in Processes

Copy into the design and connect

Synthesize and Implement

The core netlist will merge with the design during the translate phase of Implementation



Outline

- Introduction
- Using the CORE Generator System
- CORE Generator Design Flows
- **Summary**

Summary

- A core is a ready-made function that you can insert into your design
- LogiCORE™ solution products are sold and supported by Xilinx
- Using cores can save design time and provide increased performance
- Cores can be used in schematic or HDL design flows

Xilinx On-Chip Debug Using ChipScope Pro

XILINX®

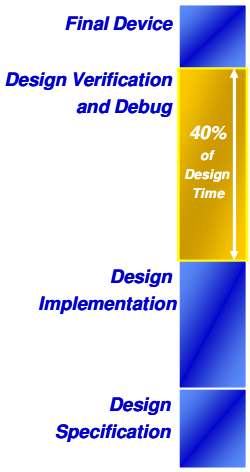
Outline

- **Introduction**
- Debugging Using ChipScope Pro
- Inserting ChipScope Core
- Summary

XILINX®

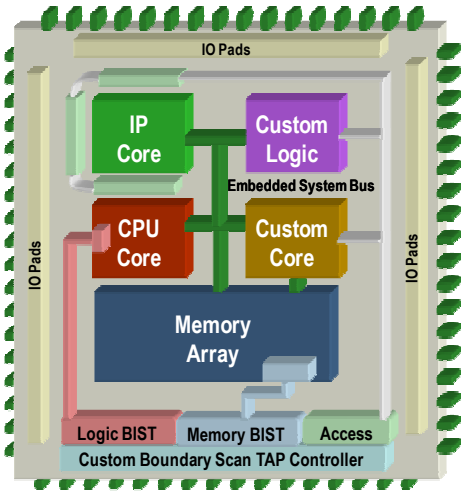
Debug and Verification is Critical

- Debug and verification can account for over 40% of an FPGA design time
- Serial nature of debug and verification can make it difficult to optimize
- Inefficient strategy may result in product launch delay
  - Loss in market share
  - Loss of first-to-market advantages



Traditional Debug

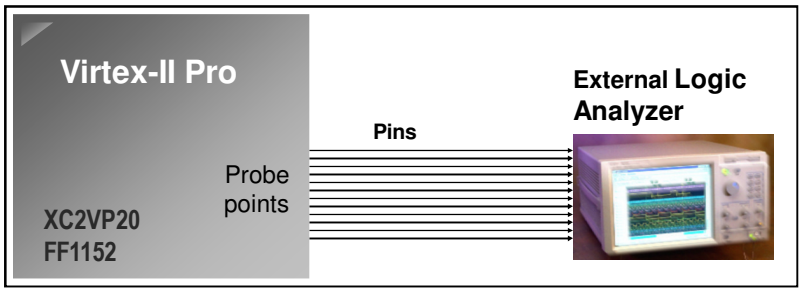
Challenges accessing internal signals



- Limited Internal Visibility
  - How do I access the embedded system bus…
- Hard IP Cores
  - Can't get internal access to…
- Full Scan Insertion
  - Increases overhead…
- It's Too Late Anyway!
  - Re-spins are ENORMOUSLY expensive
- Co-Verification
  - Tools are cumbersome and slow
  - Modeling issues

Traditional Debug

Dedicated pins connected to logic analyzer



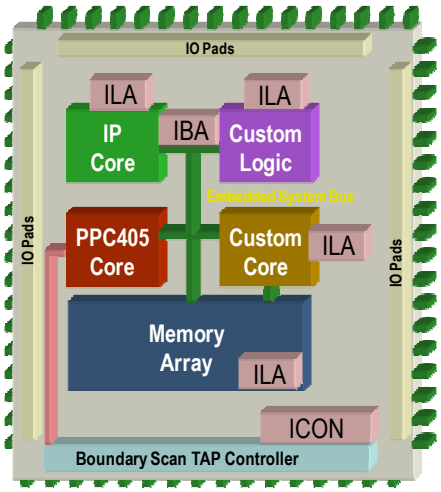
- Requires Extensive Dedicated I/O for Debug
  - Driving signals to external I/O introduces additional problems
- Inflexible solution
  - Difficult or impossible to add additional debug pins if needed
- Limited visibility to on-chip activity

Outline

- Introduction
- **Debugging Using ChipScope Pro**
- Inserting ChipScope Core
- Summary

Chipscope Debug

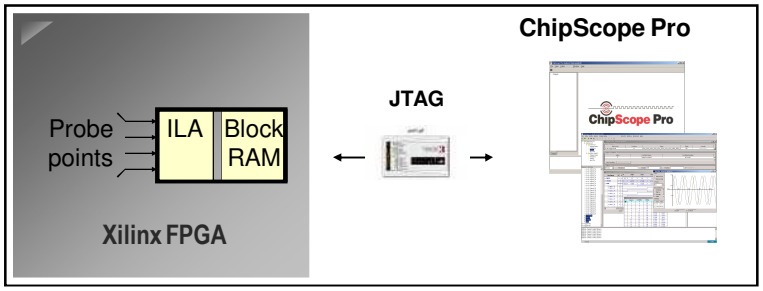
ChipScope Pro tools provide complete on-chip access to internal signals



- FPGA Enables Full Internal Visibility
  - ChipScope Pro Integrated Logic Analyzer
- Access Processor System Busses
  - ChipScope Pro Integrated Bus Analyzer
- Flexible On-Chip Debug
  - Small, efficient cores access any node or signal and can be removed at any time
- It's Never Too Late in an FPGA!
  - Fix problems during development AND after product deployment
- Enable Complete System Verification
  - Debug systems in real-time
  - No need to extrapolate behavior

ChipScope Pro On-Chip Debug

Integrated Logic Analyzer Core

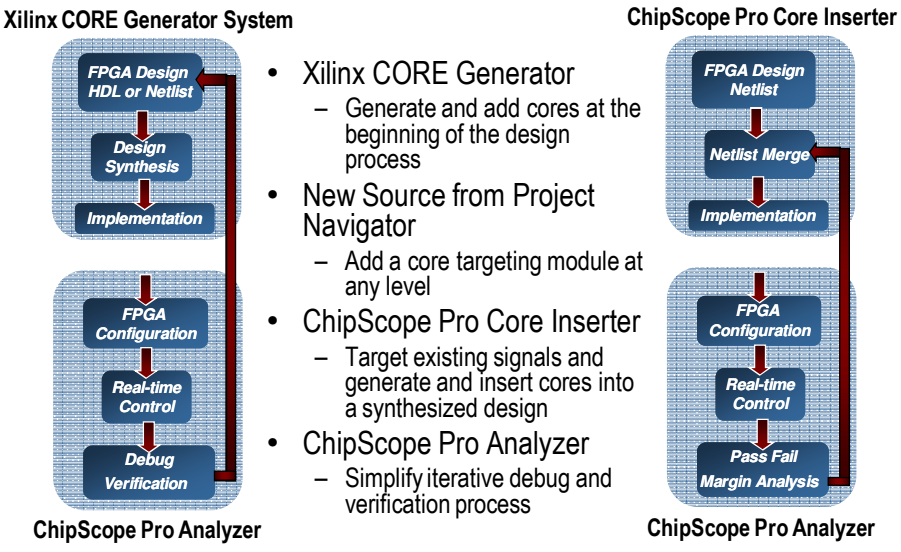


- No I/O pins required for debug
  - Access via the JTAG Port
- On-Chip access to every signal and node in the FPGA design
  - Driving signals to external I/O introduces additional problems
- Add and remove cores at any time in the design process

Outline

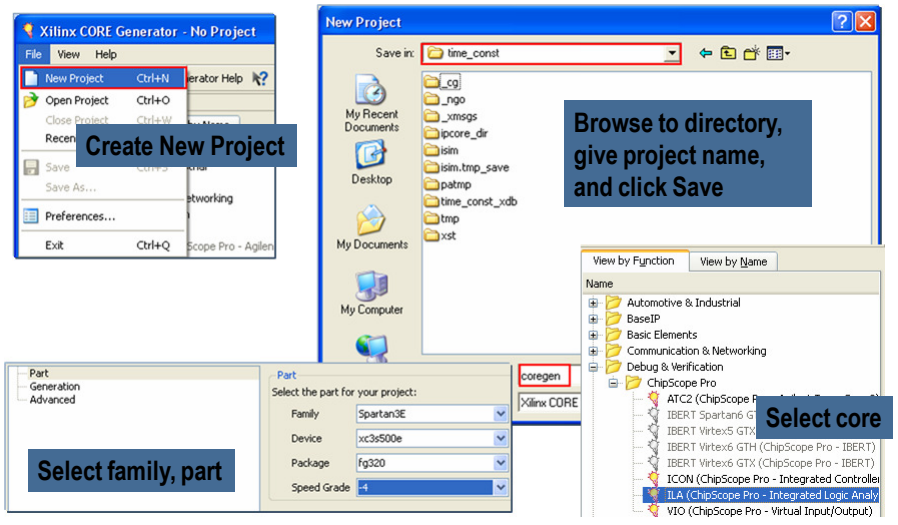
- Introduction
- Debugging Using ChipScope Pro
- **Inserting ChipScope Core**
- Summary

Add Cores at Any Time in the Design



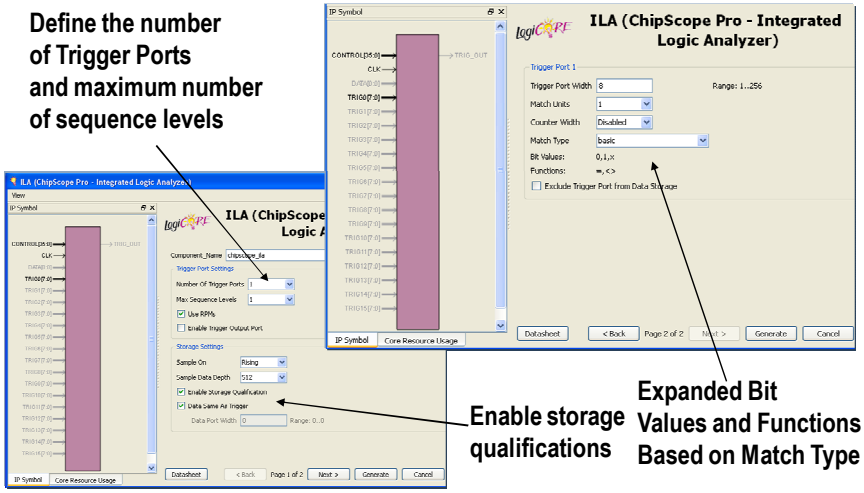
- Xilinx CORE Generator
  - Generate and add cores at the beginning of the design process
- New Source from Project Navigator
  - Add a core targeting module at any level
- ChipScope Pro Core Inserter
  - Target existing signals and generate and insert cores into a synthesized design
- ChipScope Pro Analyzer
  - Simplify iterative debug and verification process

Xilinx CORE Generator



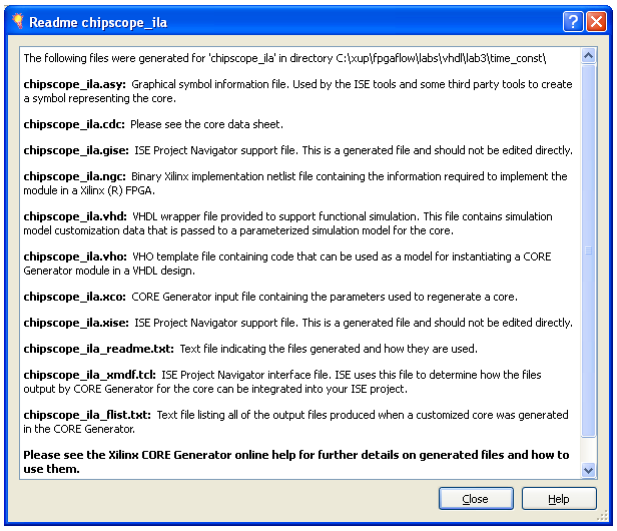
Xilinx CORE Generator

Define the number of Trigger Ports and maximum number of sequence levels



Xilinx CORE Generator

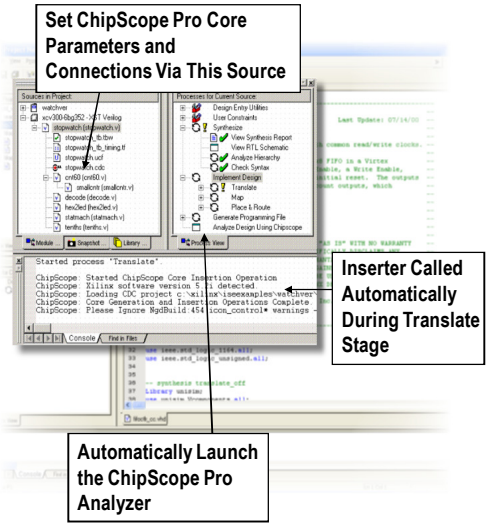
Readme file generated showing various generated file and their use



ChipScope Pro Core Inserter

Insert ChipScope Pro Cores in an existing design

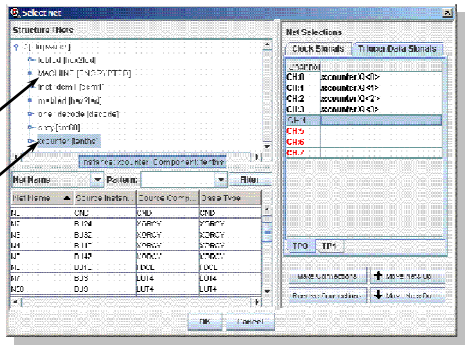
- ChipScope Pro Core Inserter
  - Add ILA and ATC2 Cores to an Existing Design Netlist
  - ChipScope Pro and ISE Integration
    - ChipScope File (.CDC) Added to Project as a Source, Associated With the Top Level Design Source
    - User Double-Clicks .CDC to Set Parameters and Connections
  - Versions of ISE and ChipScope Pro Must Match



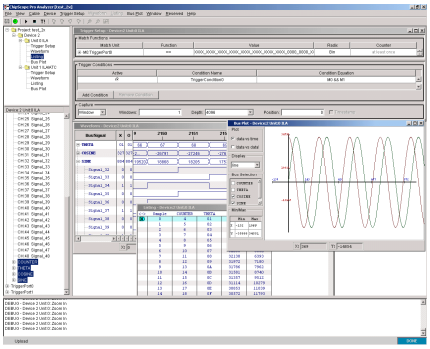
Chipscope Pro Core Inserter

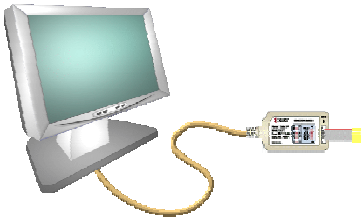
Easily connect Chipscope Pro cores in the design

- ChipScope Pro Core Inserter can now see entire design
  - NGCBUILD is used to merge top-level netlist with all underlying cores and sub-netlists
  - Encrypted files appear as black-boxes with “ENCRYPTED” tag
  - Cores appear as levels of hierarchy in Hierarchy Browser



PC Based Interface Makes FPGA Debug Easy





ChipScope Pro Analyzer functions as a logic analyzer, bus analyzer, and control console

- Access ChipScope cores via JTAG or user defined Trace port
- Configure FPGA, define trigger conditions, and view data via ChipScope Pro analyzer running on a PC

Outline

- Introduction
- Debugging Using ChipScope Pro
- Inserting ChipScope Core
- **Summary**

Summary

- ChipScope Pro core provides improved visibility and access
  - Point access from within design
  - Minimal impact to design
- Move to hardware faster
  - Accelerate evaluation, debug, and verification
  - Finish faster at the system rate
- Increased productivity
  - Integrated into FPGA design flow
  - Rapid iteration
  - Share resources