Seminar

FPGAXpert
Workbook

Lecturer

Nir Balulu

Designing FPGAs Using the Vivado Design Suite 3 2019.1

© Copyright 2019 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, UltraScale, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

DISCLAIMER

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps. All other trademarks are the property of their respective owners.

Table of Contents

Basic FPGA Architecture **1-1**

Objectives 1-1

Overview 1-2

Slice Resources 1-6

I/O Resources 1-14

Clocking Resources 1-19

RAM and DSP Slice Resources 1-26

XADC and AMS 1-30

Targeted Design Platforms 1-34

Summary 1-39

Introduction to Vivado Design Flows **2-1**

Objectives 2-1

Vivado Design Flows 2-2

ISE Design Suite Tool Flow (1) 2-4

ISE Design Suite Tool Flow (2) 2-5

Vivado Design Suite Tool Flow (1) 2-6

Vivado Design Suite Tool Flow (2) 2-7

ISE vs. Vivado Design Suite 2-9

Different Users – Different Needs 2-10

Vivado Design Suite Use Models 2-11

Project Mode vs. Non-project Mode 2-12

Project Mode vs. Non-project Mode Tcl Commands 2-13

Launching the Vivado IDE 2-14

Integrated Design Environment (IDE) 2-15

© Copyright 2019 Xilinx

Journal and Log Files 2-16
Summary 2-18

Vivado Design Suite Project-based Flow 3-1

Objectives 3-1
Project Mode 3-2
What is a Project? 3-2
Project Directory 3-4
Project-based Flow 3-5
New Project Creation Wizard 3-5
Creating a Project with HDL Sources 3-8
Creating a Project with a Synthesized Netlist 3-10
Project Filesets 3-12
Constraint File Management 3-14
Filesets and Files as Objects 3-15
Project Mode: GUI → Tcl 3-16
Project Manager 3-17
Vivado Design Suite GUI 3-18
Project Settings 3-20
Apply Your Knowledge 3-22
Summary 3-22

Vivado Design Suite I/O Pin Planning 4-1

Objectives 4-1
I/O Planning 4-1
I/O Planning before RTL 4-3
I/O Planning with RTL 4-5
I/O Planning on Netlist and Implemented Design 4-6
Pin Assignment 4-6
I/O Planning Layout View 4-7

© Copyright 2019 Xilinx

Package View 4-8
Using the Package Viewer 4-10
I/O Ports View 4-12
Interactively Placing I/O Ports 4-13
DRCs and SSN Analysis 4-14
Summary 4-15

Vivado IP Flow 5-1

Objectives 5-1
IP Operations 5-2
Vivado Design Suite: IP-Centric Design Tool 5-2
IP Synthesis Options 5-4
Output Products Generated for IP 5-6
IP Packager 5-8
IP Catalog 5-9
Adding IP to the Design: Clocking Wizard 5-10
Summary 5-12

Clocking Resources 6-1

Objectives 6-1
Overview 6-2
Clock Networks and Buffers 6-6
Clock Management Tile 6-34
Usage Models 6-46
Using Clock Resources 6-54
Summary 6-58

Basic Timing Constraints (XDC) 7-1

Objectives 7-1
Overview 7-2

© Copyright 2019 Xilinx

Basic Timing Constraints 7-8
Constraints Editor 7-18
Summary 7-23

Timing Constraints Wizard 8-1

Objectives 8-1
Timing Constraints Wizard 8-2
Creating Complete and Accurate Constraints 8-4
Missing Primary Clocks 8-6
Missing Generated Clocks in the Fabric 8-8
Forwarded Clocks 8-9
External Feedback Delays 8-9
Input Constraints 8-11
Output Constraints 8-12
Combinational Delays 8-13
Exclusive Clocks 8-15
Summary 8-15

Timing Constraints Editor 9-1

Objectives 9-1
Timing Constraint Entry 9-2
Launching the Timing Constraints Editor 9-3
Entering a create_clock Constraint 9-4
Reference Option 9-6
Other Clock Constraints 9-8
Entering set_ input_delay/set_output_delay Constraints 9-9

Timing Summary Report 10-1

Objectives 10-1
Timing Summary Report 10-1

© Copyright 2019 Xilinx

Timing Summary Report Sections 10-2
Worse Timing Path 10-4
Check Timing Report Sections 10-5
Timing Summary Report Sections 10-7
Report Timing Summary (Options) 10-9
Report Timing Summary (Advanced) 10-10
Report Timing Summary (Timer Settings) 10-13
Reviewing Timing Paths 10-14

Introduction to Timing Exceptions 11-1

Objectives 11-1
Timing Exceptions 11-1
Multicycle Paths 11-3
Creating a Multicycle Path Exception 11-4
Multicycle Path Example 11-6
Modifying the Setup and Hold Capture Edge 11-7
Enumerating Paths 11-9
Enumerating Paths Example 11-10
Creating a False Path Exception 11-12
Constants and Pseudo-Constants 11-12
Mutually Exclusive Sub-Paths 11-14
Setting Maximum and Minimum Delays 11-15
Constraining Metastability Flip-Flops 11-17
Paths Between Different Clock Domains 11-18
Combinatorial Paths Between Ports 11-18
Max/Min Delay on a Non-Startpoint 11-19
Path Segmentation on a Startpoint 11-21
Path Segmentation on an Endpoint 11-21
Path Segmentation – Breaking Other Paths 11-22

© Copyright 2019 Xilinx

Summary 11-24

Introduction to Vivado Reports 12-1

Objectives 12-1
Vivado Reports 12-2
Report Timing Summary and Report Timing 12-3
Check Timing 12-5
Report Clock Networks 12-8
Report Utilization 12-9
Report DRC 12-10
Report Methodology 12-11
Report Design Analysis 12-12
Report Clock Interaction 12-13
Report CDC 12-14
Report Exception 12-16
Report Bus Skew 12-17
Report Power 12-19
Report QoR Suggestions 12-20
Slack Histogram 12-21
Configurable Report Strategies 12-22
Summary 12-22

Introduction to FPGA Configuration 13-1

Objectives 13-1
Introduction: When Does Configuration Happen? 13-2
Configuration Pins – Basics 13-3
Configuration Pins – Key Pins To Know (1) 13-4
Configuration Pins – Key Pins To Know (2) 13-5
BIT File Generation 13-7
PROM File Generation 13-8

© Copyright 2019 Xilinx

Creating a Configuration Memory File 13-10
Summary 13-10

Introduction to the Vivado Logic Analyzer 14-1

Objectives 14-1
Logic Analyzer 14-2
Motivation for the Vivado Logic Analyzer Tool 14-3
Vivado Logic Analyzer 14-3
Benefits of Logic Debug 14-4
Vivado Logic Analyzer Probing Flows 14-6
HDL Instantiation or Insertion Flow? 14-9
Summary 14-10

Introduction to Triggering 15-1

Objectives 15-1
Triggering 15-2
Triggering is the Key 15-2
Setting Up Triggering 15-3
ILA Terminology 15-5
Displaying Captured Data 15-6
Summary 15-8

Debug Cores 16-1

Objectives 16-1
ILA Core 16-2
ILA Core Wizard 16-3
ILA Dashboard 16-5
System ILA Core 16-7
Virtual Input/Output (VIO) Core 16-8
VIO Core Wizard 16-10

© Copyright 2019 Xilinx

VIO Ports and Parameters 16-11
VIO Dashboard 16-12
Debug Core Hub 16-13
Debug Core Hub Insertion 16-14
Debug Core Hub Customization 16-16
Debug Bridge 16-16
Integrated Bit Error Ratio Tester (IBERT) Core 16-18
IBERT Sweep Function 16-19
JTAG to AXI Master 16-19

Netlist Insertion Debug Probing Flow 17-1

Objectives 17-1
Netlist Insertion Flow 17-2
Using the mark_debug Attribute 17-3
Set Up Debug Wizard to Insert Debug Cores 17-5
Debug Core Customization 17-6
Implementing Debug Cores in the Vivado IDE 17-8
Configuring the Number of Comparators and Input Pipe Stages 17-9
Apply Your Knowledge 17-9
Summary 17-10

Trigger Using the Trigger State Machine in the Vivado Logic Analyzer 18-1

Objectives 18-1
ILA Advanced Triggering 18-1
What is the Trigger State Machine? 18-3
States in the TSM 18-4
Goto Action in the TSM 18-4
Conditional Branching 18-5
Counters in the TSM 18-6
Flags in the TSM 18-6

© Copyright 2019 Xilinx

Counter Conditions 18-8
Conditional Statements 18-8
Combined Debug Probe and Counter Conditions 18-10
TSM Code in Language Templates 18-10
TSM Code Example 18-12
Summary 18-12

Sampling and Capturing Data in Multiple Clock Domains 19-1

Objectives 19-1
Multiple Clock Domains 19-2
Debugging in Multiple Clock Domains 19-3
Multiple ILAs 19-4

UltraScale Architecture DSP Resources 20-1

Objectives 20-1
UltraScale Architecture DSP48E2 Slice 20-2
X, Y, Z, and W Multiplexers 20-4
Apply Your Knowledge 20-5
Dual B Register 20-7
Dual A, D, Registers and Pre-Adder 20-8
Two-Input Logic Functions 20-9
Pattern Detect 20-10
SIMD 20-11
Cascade Paths 20-12
Wide XOR 20-13
IP Support and Inference 20-13
Math and DSP Operations in the IP Catalog 20-15
Inferring a 16 x 16 Multiplier 20-16
Inferring a 2-Input Adder 20-17
Inferring a Loadable MACC 20-18

© Copyright 2019 Xilinx

Apply Your Knowledge 20-18
Summary 20-20

Non-Integer Numbers 21-1

Objectives 21-1
Overview 21-2
Fixed Point Numbers 21-5
Floating Point Numbers 21-14
Summary 21-24

Synchronous Design Techniques 22-1

Objectives 22-1
Limitations of Asynchronous Circuits 22-1
Synchronous Circuits 22-2
Clock Signal Controls Events in Synchronous Circuits 22-4
Dedicated Clock Buffers to Reduce Clock Skew 22-5
Traditional Clock Divider 22-6
Recommended Clock Divider 22-8
Clock Glitches 22-9
Avoid Clock Glitches 22-10
Coding Clock Enables 22-10
Set/Reset Glitches 22-12
Avoid Set/Reset Glitches 22-13
Coding Synchronous Flip-Flops 22-14
Summary 22-15
Apply Your Knowledge 22-15

Designing with the IP Integrator 23-1

Objectives 23-1
What Is the Vivado IP Integrator? 23-2

© Copyright 2019 Xilinx

Vivado IP Integrator: Intelligent IP Integration 23-3
Vivado IP Integrator: Features 23-4
Design Canvas 23-5
Vivado IP Integrator: New Features 23-6
Module Referencing 23-8
DRCs 23-10
Using Tcl in IP Integrator 23-11
Board Tab 23-12
Using Board Flow to Create a Block Design 23-13
Debugging IPI Designs 23-14
Summary 23-15

Pipelining 24-1

Objectives 24-1
Pipelining Concept 24-2
Pipelining Considerations 24-3
Latency in Pipelines 24-5
Pipelining Example 24-7
Pipelining Example 24-8
Apply Your Knowledge 24-10
Answers 24-11

Synchronization Circuits 25-1

Objectives 25-1
Synchronization Circuits 25-2
Risk of Metastable Signal 25-3
Metastability Resolution 25-4
Metastability Resolution Circuit 25-6
Constraining the Single Bit Metastability Resolution Circuit 25-7
Constraints Between Different Clock Domains when Using the Single Bit Synchronizer 25-8

© Copyright 2019 Xilinx

Reset Bridge 25-10
Bus Coherency 25-11
Incorrect Bus Coherency 25-12
Capturing a Bus – Solutions 25-13
Capturing a Bus When Stable (1) 25-14
Capturing a Bus When Stable (2) 25-15
Clock-Crossing FIFO 25-16
Analyzing the CDC Report 25-17
Analyzing the MTBF Report 25-18
Summary 25-20

AXI: Transactions 26-1

Memory-Map AXI Channels 26-1
AXI Interface: Handshaking 26-3
AXI Interface: Memory-Map Read 26-4
AXI Interface: Memory-Map Write 26-7
AXI Interface: Streaming 26-9
AXI Interface: Streaming Write 26-12

JTAG to AXI Master Core 27-1

Objectives 27-1
JTAG to AXI Core 27-2
JTAG to AXI Core Features 27-4
Communication with the Core 27-4

© Copyright 2019 Xilinx

Basic FPGA Architecture

Slide 1-1:

2014.1

This module describes the basic slice resources available in 7 series FPGAs, identifies the basic I/O resources available in 7 series FPGAs, list some of the dedicated hardware features of 7 series FPGAs. 30 minutes.

Slide 1-2:

Objectives

After completing this module, you will be able to:

- Describe the basic slice resources available in 7 series FPGAs
- Identify the basic I/O resources available in 7 series FPGAs
- List some of the dedicated hardware features of 7 series FPGAs
- Differentiate different families of 7 series FPGAs
- Describe the contents of a Xilinx Targeted Design Platform kit

Note that due to time constraints, not every relevant feature of these devices will be discussed. The primary goal is to get you familiar with the most fundamental architecture features available.

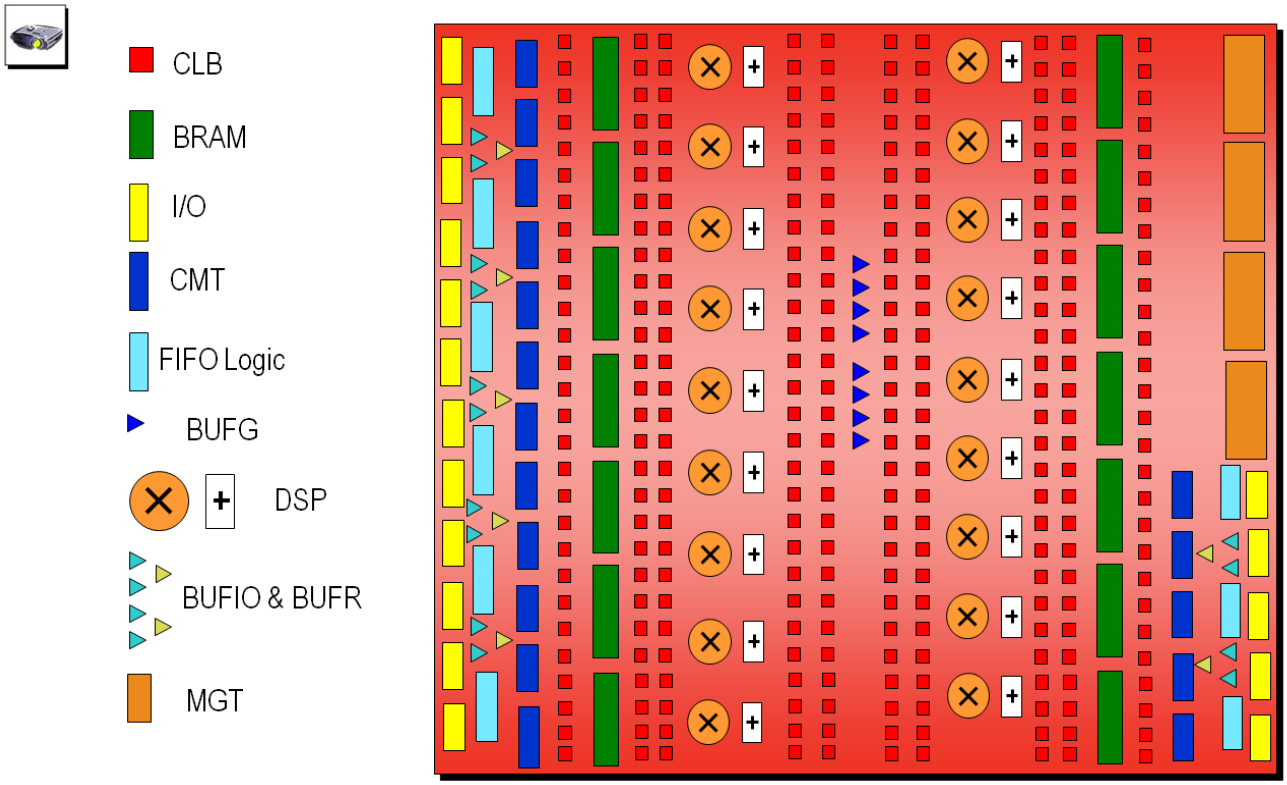
Slide 1-3:

Overview

- **Overview**
- Slice Resources
- I/O Resources
- Clocking Resources
- RAM and DSP Slice Resources
- XADC and AMS
- Targeted Design Platforms
- Summary

Slide 1-4:

Kintex-7 FPGA Feature Overview



All Xilinx FPGAs contain the same basic resources. Slices, which are grouped into Configurable Logic Blocks, or CLBs, contain combinatorial logic and register resources.

- As you can see from the figure, there are a lot of CLBs on the die. While each device family is offered with a different number of CLBs, determining which density is right for your application often requires you to at least synthesize your design before choosing an appropriate density.
- Only with some experience can you properly estimate how many slice resources will be necessary for your design, without at least synthesizing first. The Input/Output Blocks, or IOBs, interface between the FPGA and the outside world.
- The IOBs contain interface logic that is designed to translate the internal voltage domain of the FPGA to whatever I/O standard that you are employing outside of the FPGA. This saves you from having to add interface logic outside of the FPGA. The IOBs also contain registers and some specialized resources. While the slice and IOB resources are the most significant and commonly used resources, each device family has a set of dedicated resources, which are specialized.
- In the 7 series family this includes serial Multi-Gigabit Transceivers (MGTs), DSP slices, block RAM, Mixed Mode Clock Managers (MMCMs), and PLLs.
- Depending on the sub-family that you choose, you may receive all of these dedicated resources (more on this later). Finally, the programmable interconnect (which is not shown on this slide) is used to connect the signals between all of the architectural resources.
- The implementation tools select an appropriate routing solution that meets your timing needs. Therefore, it is important that you employ timing constraints to communicate your timing objectives to the implementation tools.

Slide 1-5:

Architecture Alignment

- Common elements enable easy IP reuse for quick design portability across all 7 series families
  - Design scalability from low cost to high performance
  - Expanded ecosystem support
  - Quickest time to market

Logic Fabric
LUT-6 CLB

Precise, Low Jitter Clocking
MMCMs

On-Chip Memory
36Kbit/18Kbit Block RAM

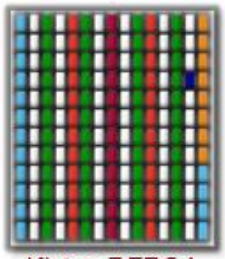
Enhanced Connectivity
PCIe® Interface Blocks

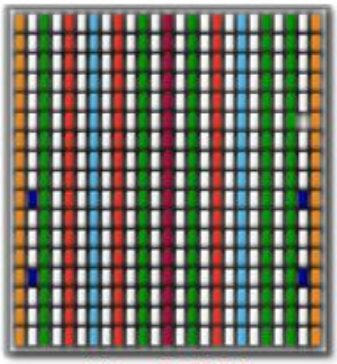
DSP Engines
DSP48E1 Slices

Hi-perf. Parallel I/O Connectivity
SelectIO™ Technology

Hi-performance Serial I/O Connectivity
Transceiver Technology

Artix-7 FPGA


Kintex-7 FPGA


Virtex-7 FPGA

All 7 series families share the same basic building blocks. The mixture and number of these resources varies across families.

Basic topology: Note the column-based architecture. The Advanced Silicon Modular Block (ASMBL) architecture allows Xilinx to assemble multiple programmable platforms with an optimal blend of features for target application domains—meaning that Xilinx has built subfamilies of the 7 Series FPGAs for particular markets.

There are multiple columns of block RAM and dedicated DSP resources dispersed across the device.

IOB columns are available via flip-chip technology.

The multi-gigabit serial transceivers (MGTs) are located on the right side of the device, along with other dedicated IP blocks. In Virtex®-7 MGTs are found on both left and right sides of the device.

על הטקסט שברצונך שיופיע כאן.1 Headingשגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 1-6:

FPGAs and SoCs

Maximum Capability	ARTIX-7 Lowest Power and Cost	KINTEX-7 Industry's Best Price/Performance	VIRTEX-7 Industry's Highest System Performance	ZYNQ Extensible Processing Platform
Logic Cells	20K – 355K	30K – 410K	285K – 2,000K	28K - 444K
Block Ram	12 Mb	29 Mb	65 Mb	240KB - 3020KB
DSP Slices	40 – 700	120 – 1,540	700 – 3,960	80 - 202
Peak DSP Perf.	504 GMACS	1,965 GMACs	5,053 GMACS	2622 GMACS
Transceivers	4	16	88	16
Transceiver Performance	3.75Gbps	6.6Gbps and 10.3Gbps	10.3Gbps, 13.1Gbps and 28Gbps	6.6Gbps - 12.5Gbps
Memory Performance	1,066Mbps	2,133Mbps	2,133Mbps	1333Mbps
I/O Pins	450	500	1,200	528
I/O Voltages	3.3V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below

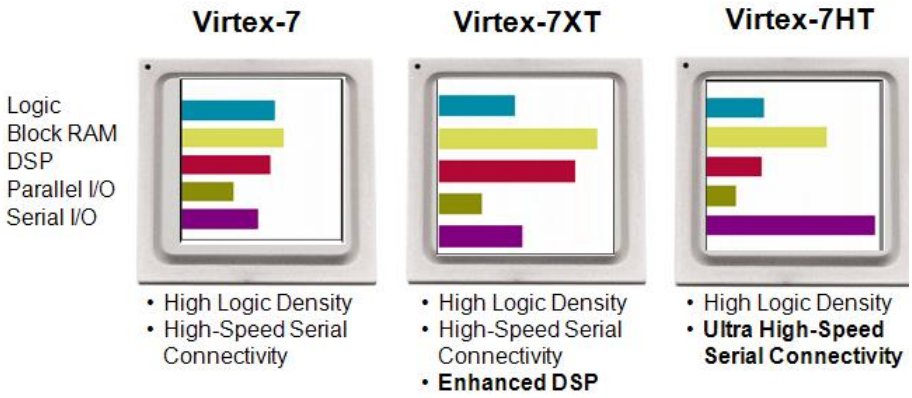
The Xilinx 7 series FPGAs are a unified set of FPGA families. These families share the same underlying technology, construction, and implementation tools. The families are optimized for different price / performance points to meet all market needs. However, the unified architecture allows unprecedented migration capability from one family to another.

The Zynq®-7000 All Programmable SoC family series architecture contains an ARM®-based processing system that is tightly integrated to the programmable logic. The programmable logic portion of Zynq-7000 devices leverages the Xilinx 7 series programmable logic. This allows designers currently developing on the Xilinx 7 series to seamlessly port their FPGA designs to Zynq-7000 devices.

Slide 1-7:

Virtex-7 FPGA Sub Families

- The Virtex-7 family has several sub-families
  - Virtex-7: General logic
  - Virtex-7XT: Rich DSP and block RAM
  - Virtex-7HT: Highest serial bandwidth



Slide 1-8:

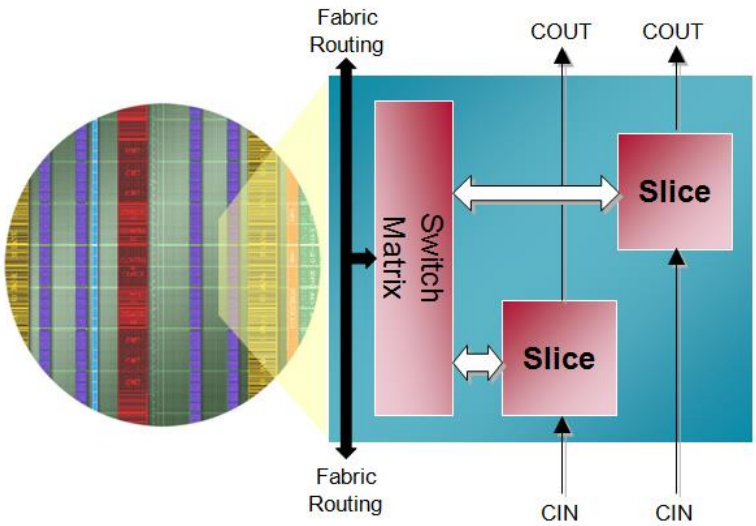
Slice Resources

- Overview
- **Slice Resources**
- I/O Resources
- Clocking Resources
- RAM and DSP Slice Resources
- XADC and AMS
- Targeted Design Platforms
- Summary

Slide 1-9:

CLB in the 7 Series FPGAs

- Primary resource for design
  - Combinatorial functions
  - Flip-flops

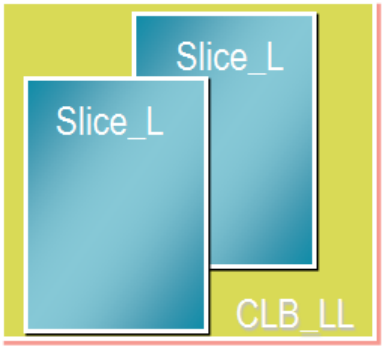


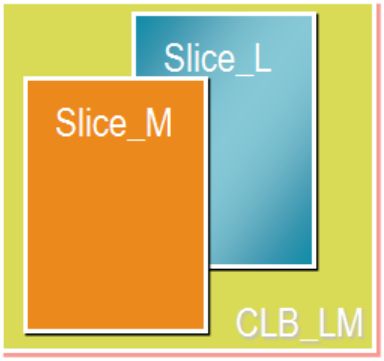
- CLB contains two slices
- Connected to switch matrix for routing to other FPGA resources
- Carry chain runs vertically in a column from one slice to the one above

Slide 1-10:

Two Types of Slices

- Two types of slices
  - 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



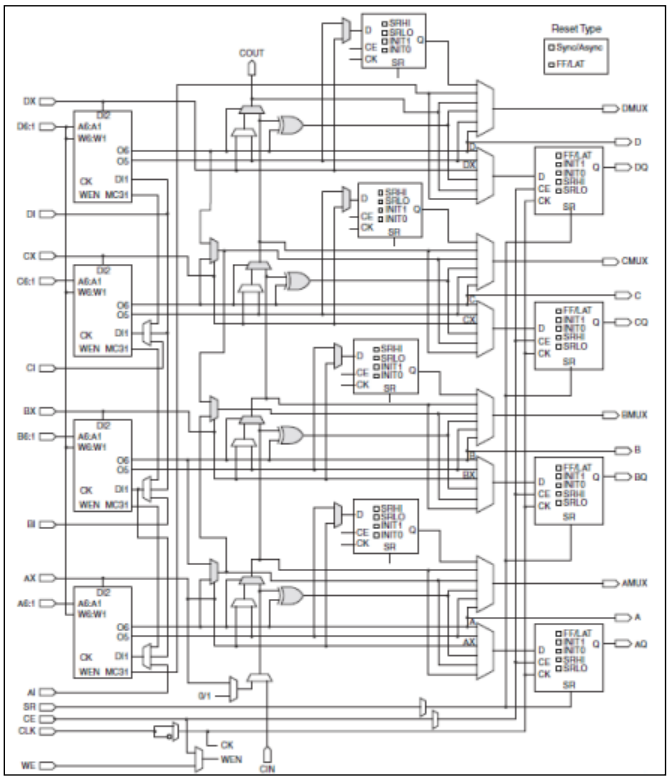


In the 7 series FPGAs, approximately ¼ of slices are SLICEM, the remainder are SLICEL. CLB columns on both sides of the block RAM columns have SLICEM / SLICEL CLBs, resulting in slightly more than ¼ of SLICEM.

Slide 1-11:

FPGA Slice Resource

- Four six-input look-up tables (LUT)
- Wide multiplexers
- Carry chain
- Four flip-flop / latches
- Four additional flip-flops
- The implementation tools (MAP) are responsible for packing slice resources into the slice



This is the diagram of SLICEM.

Each LUT also has some associated logic that includes carry logic. The carry chain enables the propagation of a carry signal between the corresponding bits when implementing arithmetic functions (such as accumulators, subtractors, or comparators, for example). This enables high performance and efficient device utilization.

The dedicated multiplexers, called the F7 and F8 multiplexers, allow for the implementation of wider logic. If two LUT6s and the associated F7MUX are used, any arbitrary 7-input combinatorial function can be implemented.

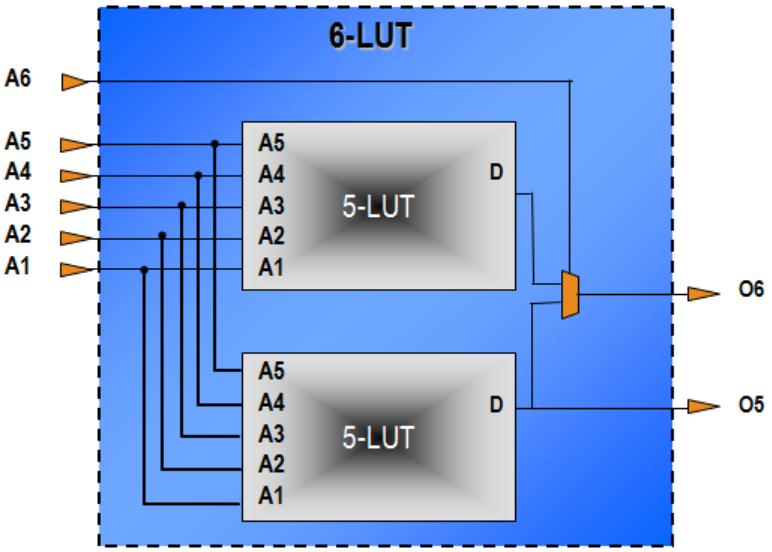
Similarly, if all four LUTs, the F7MUX resources, and the F8MUX are used, an arbitrary 8-input combinatorial function can be implemented. These multiplexers can also be used to build larger multiplexers.

Because a 4-input multiplexer can be implemented in one LUT6 (four data inputs and two control inputs), a 16-1 multiplexer can be implemented using all four LUTs and the F7MUX and F8MUX. Logic that uses these built-in multiplexers will be significantly faster than logic built using only LUTs.

Slide 1-12:

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



LUTs can perform any combinatorial function limited only by the number of inputs. It is your primary combinatorial logic resource and it is the industry standard.

The look-up table functionality is essentially a small memory containing the desired output value for each combination of input values.

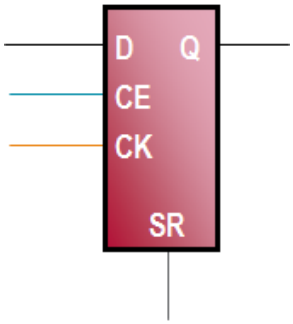
- The truth table for the desired function is stored in the memory.
- The inputs of the function act as the address to be read from the memory (essentially a multiplexer controlled by the inputs).
- The values for the storage elements are generated by the ISE® software tools, and downloaded to all LUTs at configuration time.

Each 6-input LUT can be configured as two 5-input LUTs. This gives the device a great deal of flexibility to build an efficient design.

- Thus, the LUT can be used to build any function of six variables or two independent functions of five variables.
- The synthesis and implementation tools use these resources to build combinatorial functions automatically.

Slide 1-13:

Slice Flip-Flop Capabilities

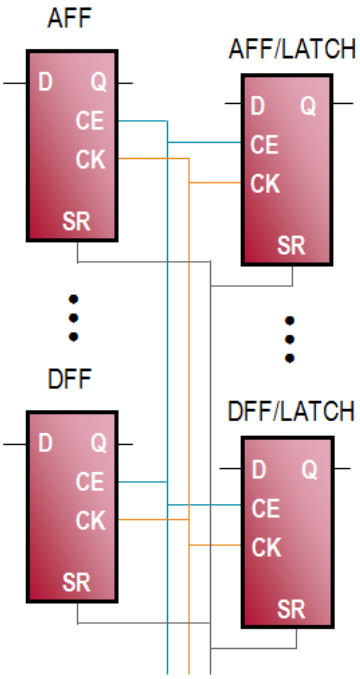


- All flip-flops are D type
- All flip-flops have a single clock input (CK)
  - Clock can be inverted at the slice boundary
- All flip-flops have an active high chip enable (CE)
- All flip-flops have an active high SR input
  - Input can be synchronous or asynchronous, as determined by the configuration bit stream
  - Sets the flip-flop value to a pre-determined state, as determined by the configuration bit stream

The management of the control signals (CLK, CE, and SR) are discussed in the *Slice Flip-Flops* module.

Slide 1-14:

Control Sets



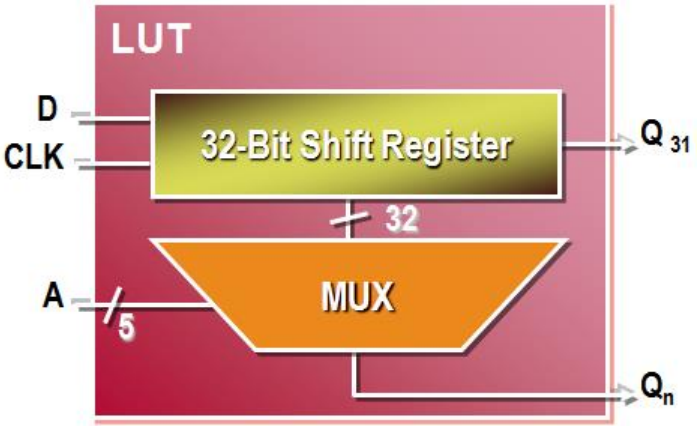
- All flip-flops and flip-flop / latches share the same CK, SR, and CE signals
  - This is referred to as the *control set* of the flip-flops
  - CE and SR are active high
  - CK can be inverted at the slice boundary
- If any one flip-flop uses a CE, all others must use the same CE
  - CE gates the clock at the slice boundary
  - Saves power
- If any one flip-flop uses the SR, all others must use the same SR
  - The reset value used for each flip-flop is individually set by the SRVAL attribute

All flip-flops within the slice will enter and leave the reset state at the same time, because they share the same SR signal. However, each flip-flop has its own SRVAL attribute, so the value forced on to the Q output when in the reset state is individually set.

Slide 1-15:

SLICEM Used as 32-bit Shift Register

- Versatile SRL-type shift registers
  - Variable-length shift register
  - Synchronous FIFOs
  - Content-Addressable Memory (CAM)
  - Pattern generator
  - Compensate for delay / latency
- Shift register length is determined by the address
  - Constant value giving fixed delay line
  - Dynamic addressing for elastic buffer
- Cascadable up to 128x1 shift register in one slice



SRL Configurations in One Slice (4 LUTs)
16x1, 16x2, 16x4, 16x6, 16x8
32x1, 32x2, 32x3, 32x4
64x1, 64x2
96x1
128x1

In the SLICEM slices, the LUT can also be configured as a dynamically addressable shift register. This component basically acts as a programmable pipeline delay element.

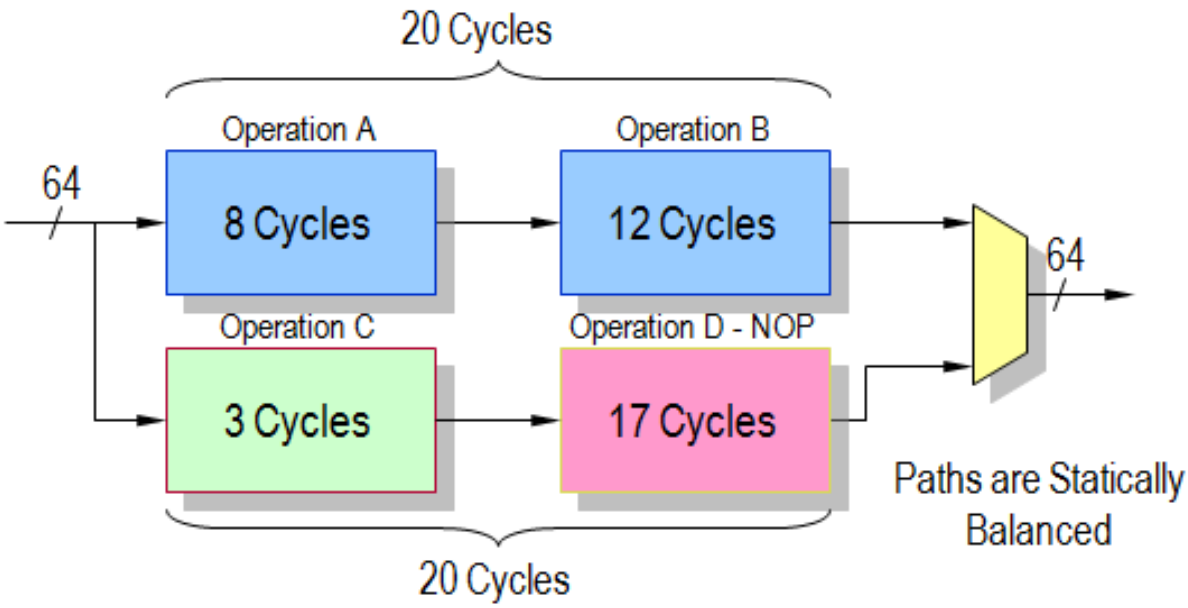
There are no set or reset capabilities; it is not loadable, and data can only be read serially. To ensure that software can map pipeline delays to the SRL, be sure to code them with these restrictions in mind.

Each LUT6 can implement a maximum delay of 32 clock cycles. The SRLs within a slice can be cascaded for longer shift registers (up to 128).

The shift register length can be changed asynchronously by changing the value applied to the address pins (A). This means that you can dynamically change the pipeline delay associated with an SRL.

Slide 1-16:

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)

Because there are so many registers in FPGAs, pipelining is an effective method of designing to increase design performance. Because pipelines can sometimes become unbalanced, it may be necessary to delay branches of the pipeline. SRLs are ideal for this purpose.

In this example, you see a 64-bit bus processed through operations A, B, and C. A has a delay of eight cycles, B has a delay of twelve cycles, and C has a delay of three cycles. Because the data processed is also grouped at its output with a multiplexer, these datapaths must be synchronized so that appropriate data is compared at the multiplexer. To do this, the SRL can be used to delay the C operation by 17 clock cycles; essentially, 17 *No Operation (NOP)* operations.

If you were to do this with registers, it would require 1,088 registers. If you use the SRL functionality instead, you only need 64 LUTs, each programmed for 17 clock cycles of delay.

Slide 1-17:

SLICEM Used as Distributed SelectRAM Memory

- 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

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

The look-up table functionality is essentially a small memory containing the desired output value for each combination of input values. These storage cells are programmed at configuration time, and the look-up itself is done by using the inputs as the control for a wide multiplexer.

By allowing these storage elements to be modified using FPGA fabric resources, the LUT can be used for the implementation of a small distributed memory.

Each LUT can be a single ported 64-bit RAM with synchronous write and asynchronous read. LUTs in slices can be combined to create small dual-port and multi-port RAMs.

Approximately one quarter of the slices in each 7 series device is a SLICEM, which has LUTs that can be used as distributed SelectRAM™ memory.

Simple dual-port configurations can be used to implement LUT FIFOs and MicroBlaze™ processor register files.

Slide 1-18:

Apply Your Knowledge

1) Can you describe the capabilities of a LUT?

2) Can you describe the capabilities of a LUT?
The LUT (or function generator) can be used to:
- Build combinatorial logic
- Implement a shift register (SRL) in SLICEM only
- Build a distributed memory in SLICEM only

Slide 1-19:

I/O Resources

- Overview
- Slice Resources
- **I/O Resources**
- Clocking Resources
- RAM and DSP Slice Resources
- XADC and AMS
- Targeted Design Platforms
- Summary

Slide 1-20:

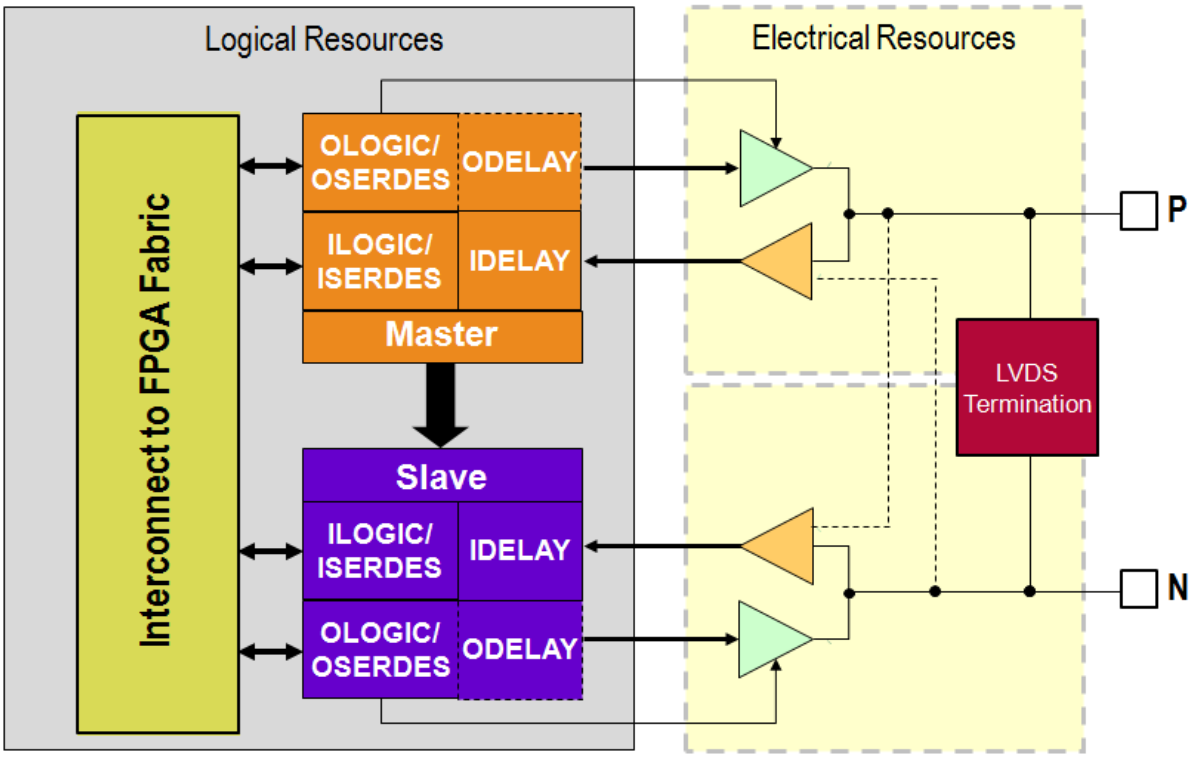
I/O Types

- Two different types of I/O in 7 series FPGAs
  - High Range (HR)
    - Supports I/O standards with Vcco voltages up to 3.3V
  - High Performance (HP)
    - Supports I/O standards with Vcco voltages up to 1.8V only
    - Designed for the highest performance
    - Has ODELAY and DCI capability

I/O Types	Artix-7 Family	Kintex-7 Family	Virtex-7 Family	Virtex-7 XT/HT Family
High Range	All	Most	Some	
High Performance		Some	Most	All

Slide 1-21:

I/O Block Diagram



Note: The ODLEAY is only available in High Performance (HP) banks.

Slide 1-22:

I/O Versatility

- Each pin can be input and output (including 3-state)
- Each pin can be individually configured
  - IDELAY, drive strength, input threshold, termination, weak pull-up or pull-down
  - Based on banking rules (some standards not compatible within the same bank)
- Each I/O supports 40+ voltage and protocol standards, including
  - LVCMOS
  - LVDS, Bus LVDS
  - LVPECL
  - SSTL
  - HSTL
  - RSDS_25 (point-to-point)
- Many standards offer programmable slew rate

Each pair of I/O pins can be configured as either one differential I/O or two single-end I/O.

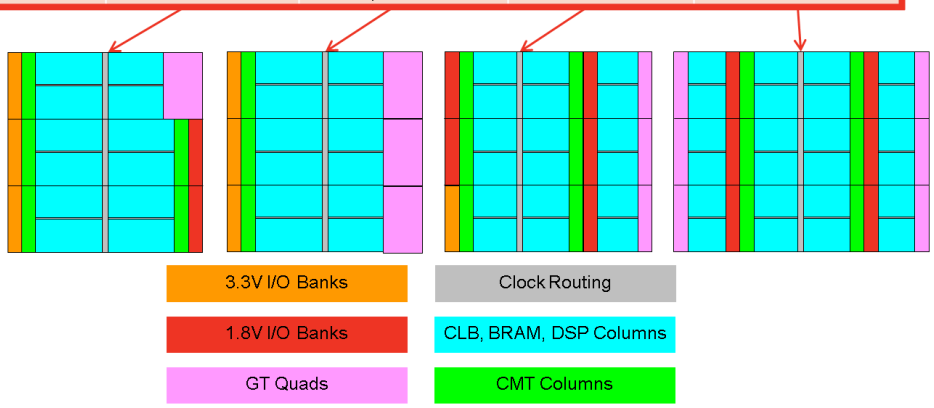
Supported I/O standards vary by device family. Refer to the device data sheet for specific I/O-standard information.

7 series devices offer programmable slew rate on more I/O standards than previous generations, including SSTL and HSTL. When migrating from an older technology, be sure to set the slew rate appropriately. The default slew rate is SLOW.

Slide 1-23:

I/O Columns and Types – Kintex and Virtex Families

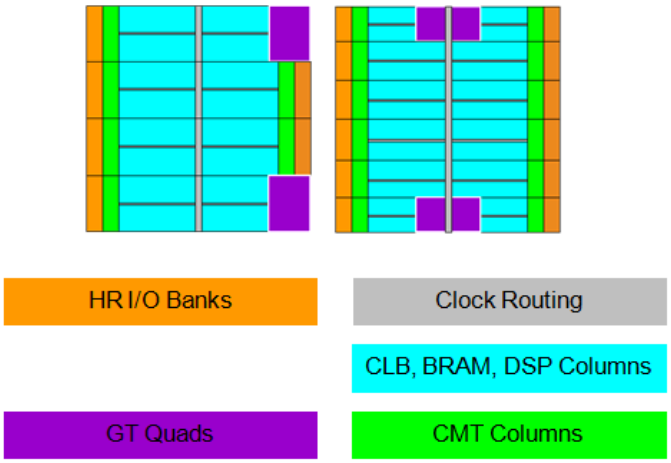
I/O Feature	Kintex-7 Family		Virtex-7 Family	Virtex-7 XT/HT Family
CMT Columns	1 + Partial	1	2	2
I/O Columns	1 + Partial	1	2	2
GT Columns	Partial Shared with I/O	Complete Replaces I/O	1	2



Slide 1-24:

I/O Columns and Types – Artix Family

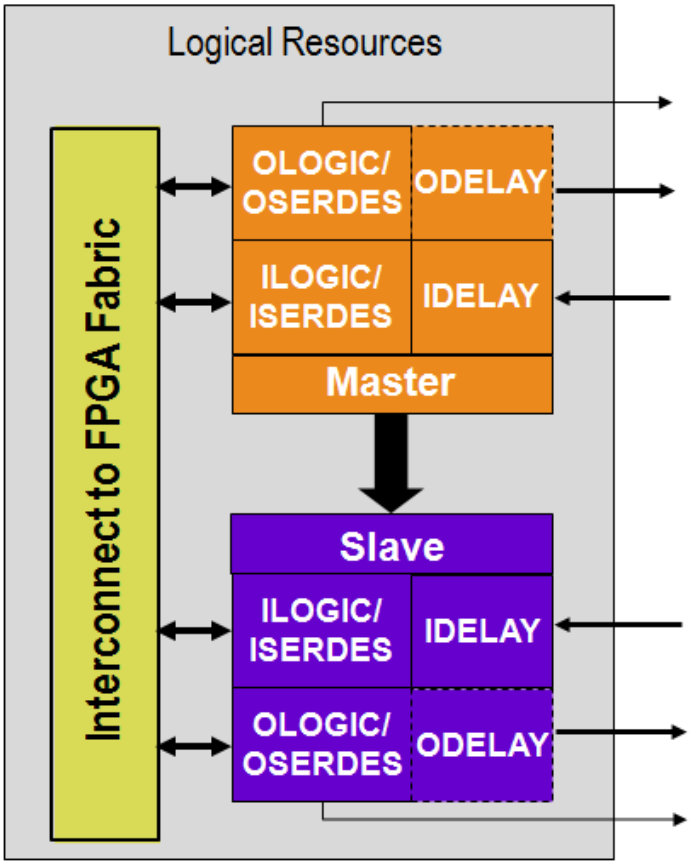
Feature	Mid-Range Devices	Larger Devices
CMT Columns	1 + Partial	2
I/O Columns	1 + Partial	2
GP Quads	Partial Shared with I/O	Embedded in Fabric



Slide 1-25:

I/O Logical Resources (7 Series)

- Two blocks of logic per I/O pair
  - Master and slave
  - Can operate independently or concatenated
- Each block contains
  - ILOGIC/ISERDES
    - SDR, DDR, or high-speed serial input logic
  - OLOGIC/OSERDES
    - SDR, DDR, or high-speed serial output logic
  - IDELAY
    - Selectable fine-grained input delay
  - ODELAY
    - Selectable fine-grained output delay
    - Only available on High Performance I/O



The SERDES resource can do 1-8 serialization / deserialization.

Master and Slave can be paired for 1-14 serialization / deserialization (could be added to master if required).

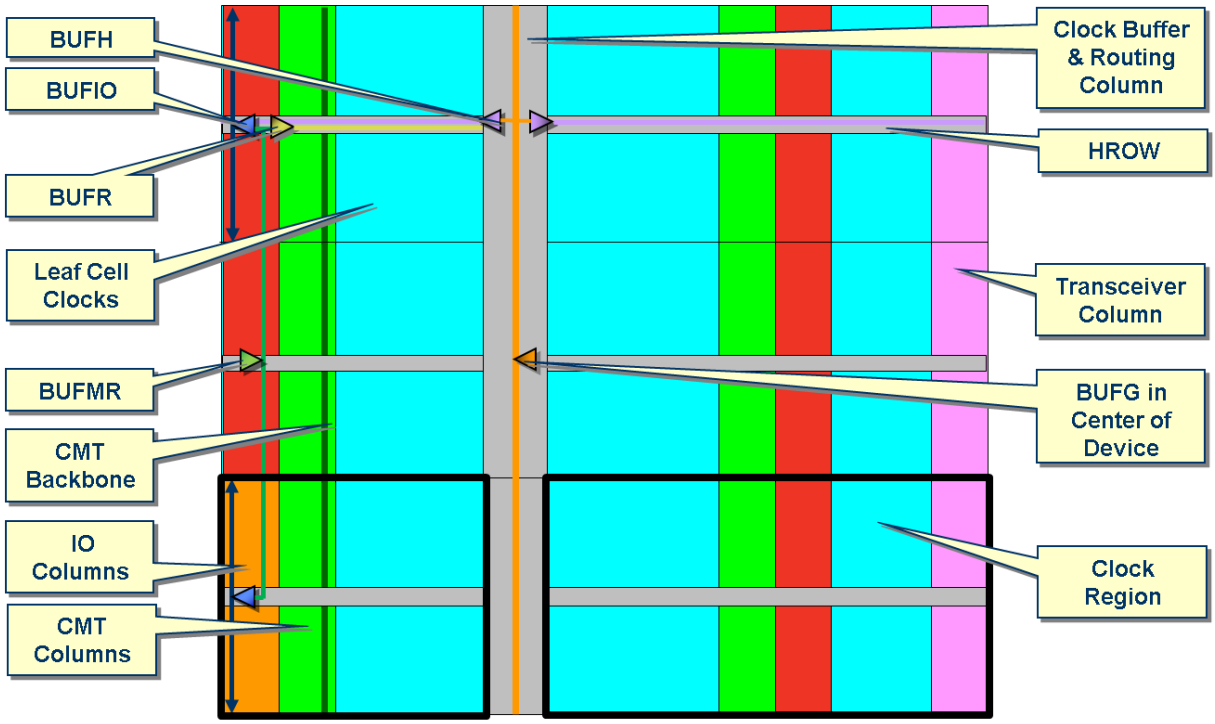
Slide 1-26:

Clocking Resources

- Overview
- Slice Resources
- I/O Resources
- **Clocking Resources**
- RAM and DSP Slice Resources
- XADC and AMS
- Targeted Design Platforms
- Summary

Slide 1-27:

7 Series Clock Networks



The Clock Buffer and Routing Column contains only the global clock buffers and routing.

The MMCMs and PLLs are located in the CMT columns that are immediately adjacent to the IO columns.

The global clock buffers (BUFG) are in the middle of the chip. These drive the vertical spines of the global clock network.

The horizontal spines of the global clock network run through the center of each clock region. These horizontal spines are driven by BUFH buffers.

There are also regional clock routing resources driven by BUFRs.

Clocks are driven up and down from the center horizontal row (HROW) of each clock region.

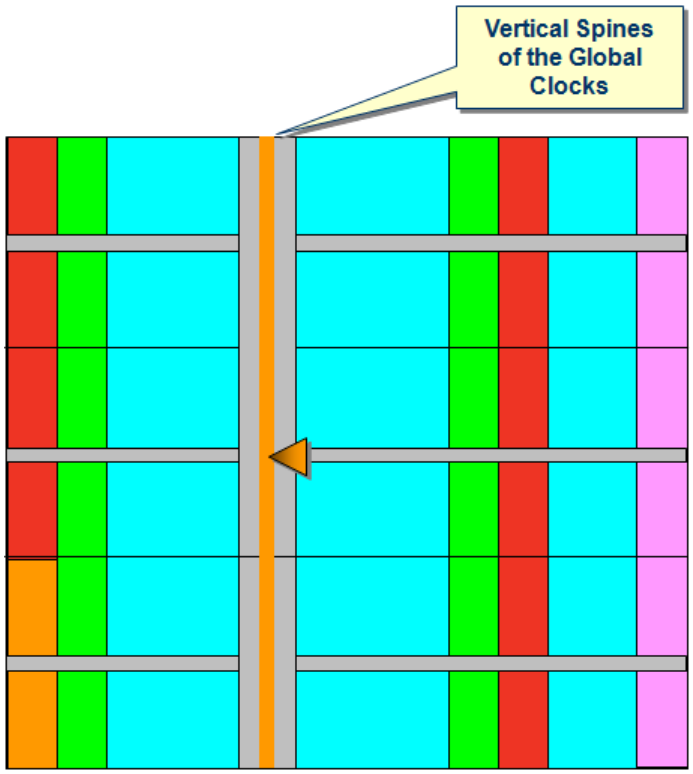
The BUFIO are placed within the I/O column, and drive the I/O clock network in that bank.

The BUFMR are dedicated buffers that allow clock inputs to drive the BUFIOs and BUFRs of adjacent regions.

Slide 1-28:

Global Clocking – Vertical Spines

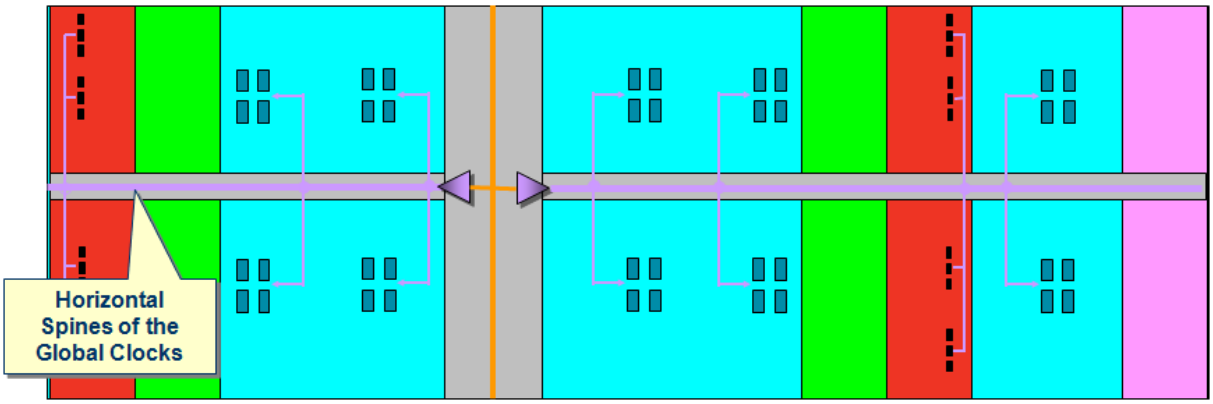
- Global clock networks allow clocks to be distributed to potentially every clocked element on the die
- Global clocks are driven by BUFGCTRL located in the middle of the die
- Each BUFGCTRL drives a vertical spine in the center of the die that runs from the bottom to the top of the die
- All 7 series devices have 32 vertical spines



Slide 1-29:

Global Clocking – Horizontal Spines

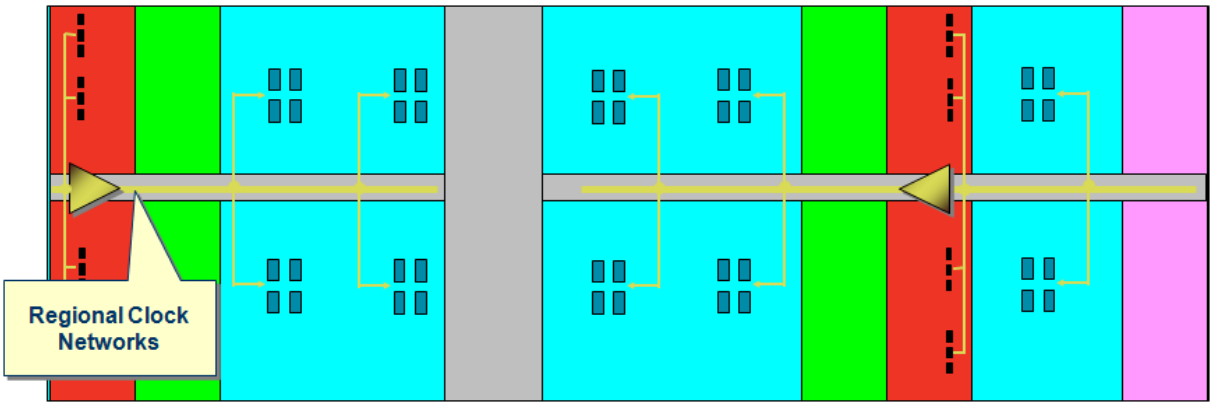
- Each clock region has 12 horizontal spines for carrying global clocks
  - Clock regions on left and right have 12 independent horizontal spines
- These horizontal spines can drive all clocked resources within the region
  - Slice flip-flops, distributed RAMs, SRLs, block RAMs, DSP slices, IOB flip-flops, CLKDIV of ISERDES and OSERDES ...
- Each horizontal spine is driven by a BUFH



Slide 1-30:

Regional Clock Networks

- Each clock region has four regional clock networks
  - Regional clocks in the left and right half are separate
- These networks are in addition to the 12 horizontal spines of the global clock
- These regional networks can drive all clocked resources within the region
- Each horizontal spine is driven by a BUFR

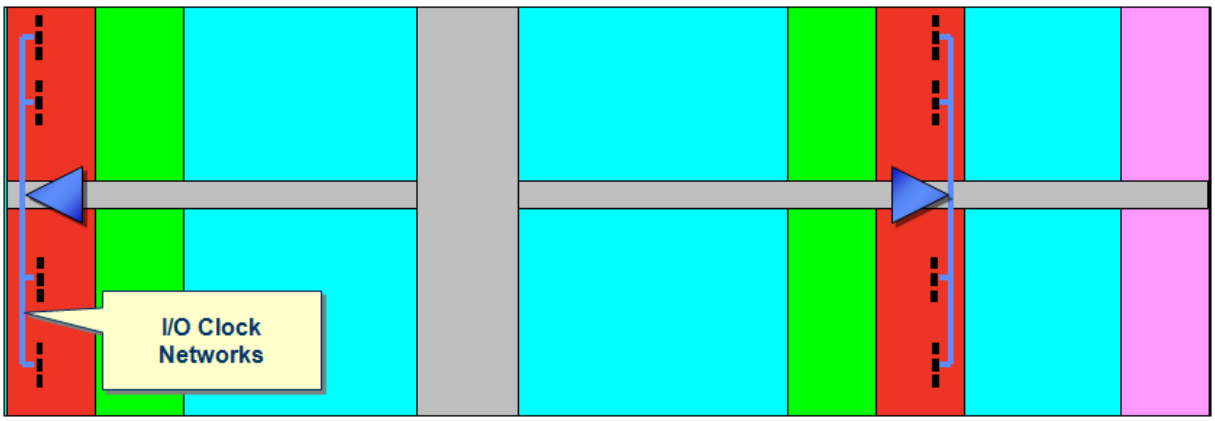


על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 1-31:

I/O Clock Networks

- Each clock region has four I/O clock networks
- These networks can drive only the clock ports of the ILOGIC / OLOGIC resources and the high-speed clock ports (CLK) of the ISERDES / OSERDES
- Each I/O clock network is driven by a BUFIO
- Highest quality clock available



The I/O clock networks can drive the clock pins of the IOB resources *only*:

- DDR and SDR flip-flops in the ILOGIC / OLOGIC block.
- High-speed clock port (CLK) of the ISERDES and OSERDES resources.

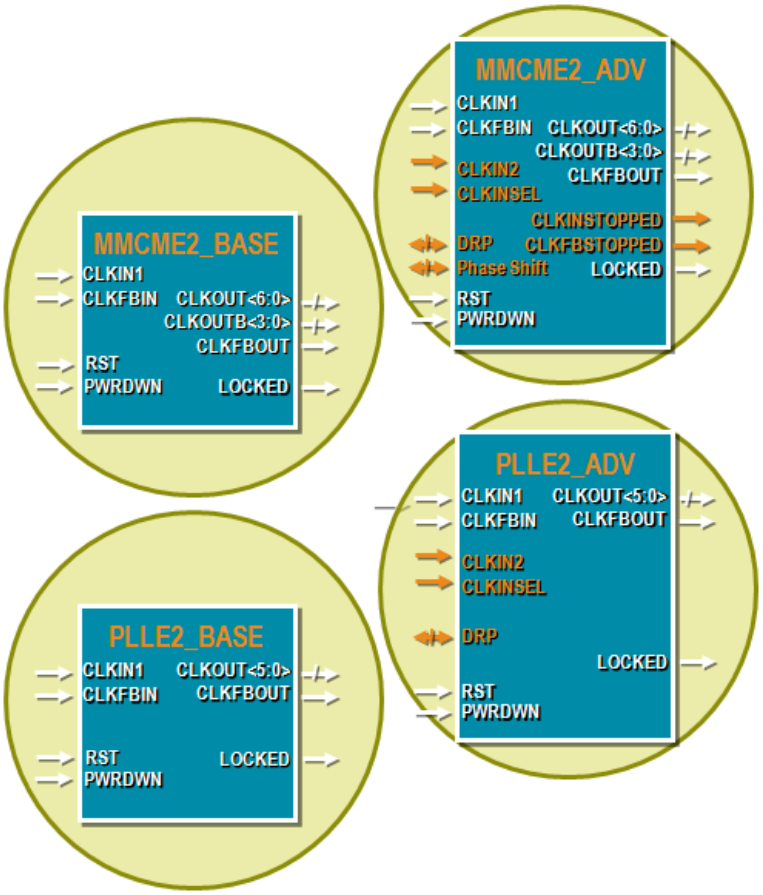
Because of the low fanout, they have extremely low skew and a very short insertion delay (ideal for source-synchronous interface applications).

These networks add the least jitter of all clock networks, and have run at the highest speeds (up to 1066 MHz).

Slide 1-32:

MMCM and PLL Features

- Up to 24 CMTs per device
  - One MMCM and one PLL per CMT
- Two software primitives
  - *_BASE has only the basic ports
  - *_ADV provides access to all ports
- PLL is primarily intended for use with the I/O phaser for high-speed memory controllers
- The MMCM is the primary clock resource for user clocks



Each CMT has one MMCM and one PLL. The two resources share some of the same basic functionality, but the MMCM has some additional features for manipulating user clocks.

While the PLL can also be used for user clocks, it is primarily intended to be used in conjunction with the I/O phaser and I/O FIFOs for high speed memory controllers.

While there is only one MMCM per CMT (as opposed to two in the Virtex-6 FPGA), there are generally twice as many CMTs in a device, because there are two columns of CMTs (as opposed to one in the Virtex-6 FPGA). This allows the PLLs to be reserved for high-speed memory controllers.

על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 1-33:

PLL Features

- Frequency lock between F_{PFD} and CLKFB
- Phase lock between CLKIN and CLKFB
- Frequency synthesis for up to six different outputs
  - Fout = Fin * M / (D*O)
    - One M and one D value per PLL
    - Each PLL output can have its own O value
    - M: 1-64; D: 1-56; O: 1-128
- Static phase shifting increments of 1/8 of the VCO period
  - Each output has independent selection of the phase
- Duty cycle generation in increments of 1/8 of the VCO period
- Reference clock switching
- Lock detection
- PWRDWN signal to disable regulated power supply

There is one M and D divider per PLL. These two dividers determine the frequency of operation of the VCO.

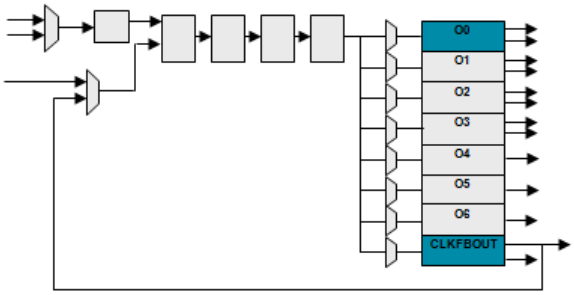
$F_{VCO} = F_{CLKIN} * M / D$

Each output has its own divider, O. The output frequency is F_{VCO} / O.

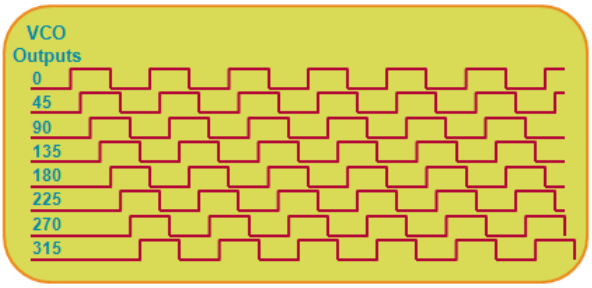
Slide 1-34:

Extra MMCM Features

- Fractional counters
  - Ability to configure O0 and M as a counter with 1/8th granularity (for example, 2.125, 2.250, 2.375, etc.)
  - Enables many more VCO frequencies and output frequencies to be synthesized



- Dynamic phase shifting



- Fine phase shifting
- Divider cascading for wider divider range
  - O4 divider uses resources from O6

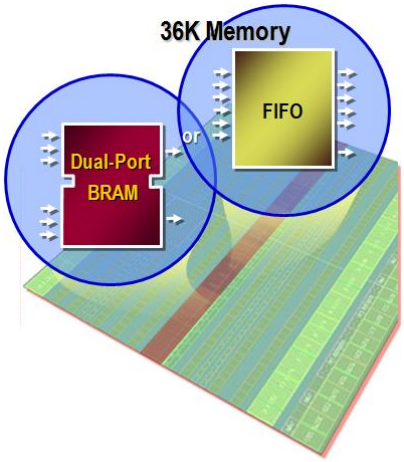
Slide 1-35:

RAM and DSP Slice Resources

- Overview
- Slice Resources
- I/O Resources
- Clocking Resources
- **RAM and DSP Slice Resources**
- XADC and AMS
- Targeted Design Platforms
- Summary

Slide 1-36:

7 Series Block RAM and FIFO (1)



- All members of the 7 series families have the same Block RAM / FIFO
- Fully synchronous operation
  - All operations are synchronous; all outputs are latched
- Optional internal pipeline register for higher frequency operation
- Two independent ports access common data
  - Individual address, clock, write enable, clock enable
  - Independent data widths for each port
- Multiple configuration options
  - True dual-port, simple dual-port, single-port

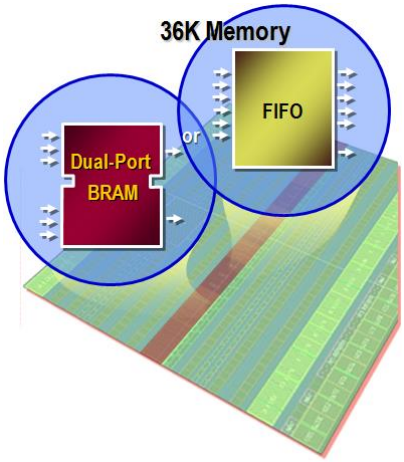
The 7 series FPGAs have a separate Vccbram power supply. This allows the voltage of the core to be lowered in the -1L part without also lowering the voltage of the block RAMs.

Unused block RAMs are automatically powered down by the configuration bitstream. This reduces overall power of the FPGA.

Slide 1-37:

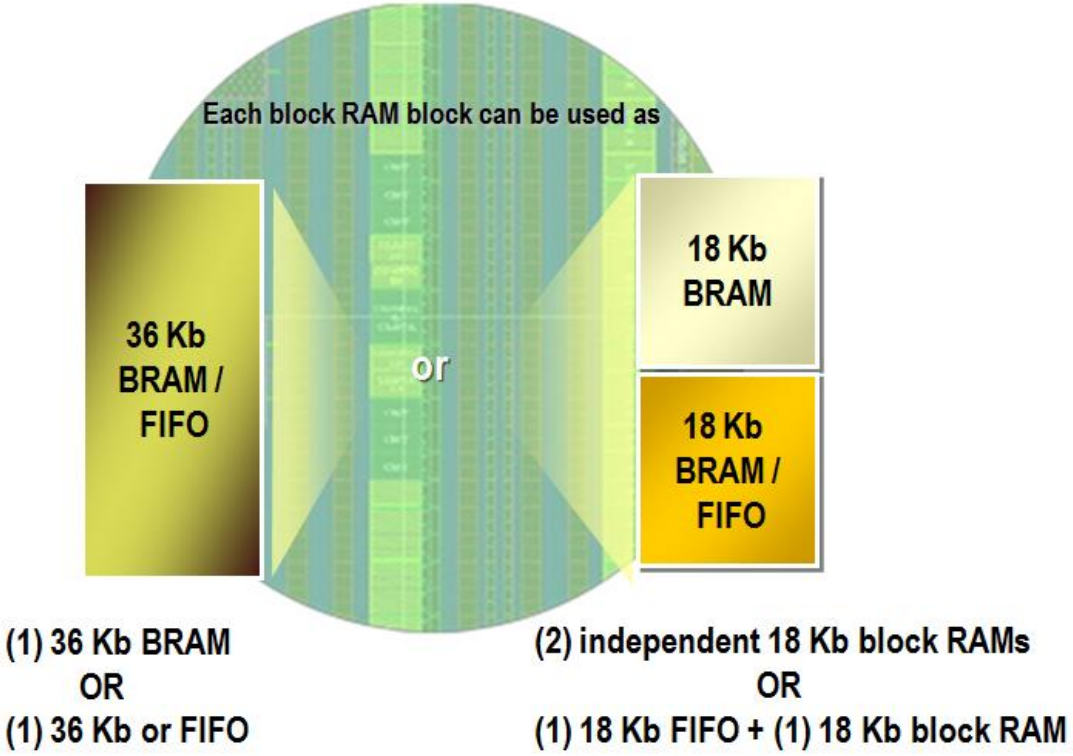
7 Series Block RAM and FIFO (2)

- Integrated cascade logic
- Byte-write enable in wider configurations
- Integrated control for fast and efficient FIFOs
- Integrated 64 / 72-bit Hamming error correction
- Separate Vbram supply to ensure block memory functionality in -1L



Slide 1-38:

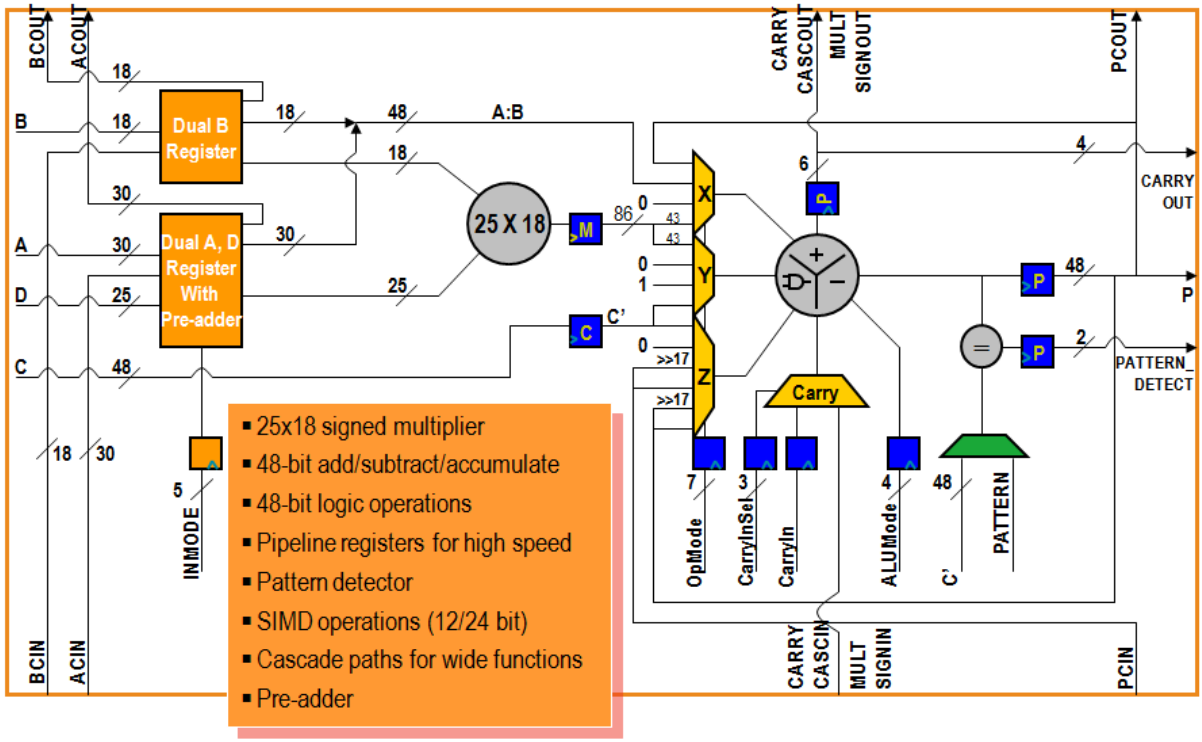
7 Series Block RAM and FIFO Block



על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 1-39:

7 Series DSP48E1 Slice



All 7 series FPGAs contain the same DSP48E1 slice. Only the number of slices and maximum frequency vary from family to family.

C′ denotes the output of the optional C pipeline register. This bus is one input of the pattern detect multiplexer; the other input is the 48-bit PATTERN attribute.

Slide 1-40:

XADC and AMS

- Overview
- Slice Resources
- I/O Resources
- Clocking Resources
- RAM and DSP Slice Resources
- **XADC and AMS**
- Targeted Design Platforms
- Summary

Slide 1-41:

XADC and Analog Mixed Signal

- XADC is a high quality and flexible analog interface new to the 7 series
  - Dual 12-bit 1Msps ADCs, on-chip sensors, 17 flexible analog inputs, and track & holds with programmable signal conditioning
- Analog Mixed Signal (AMS)
  - Using the FPGA programmable logic to customize the XADC and replace other external analog functions; for example, linearization, calibration, filtering, and DC balancing to improve data conversion resolution

The integration of the XADC block now allows FPGAs to support a general-purpose analog interface that can address a wide range of applications.

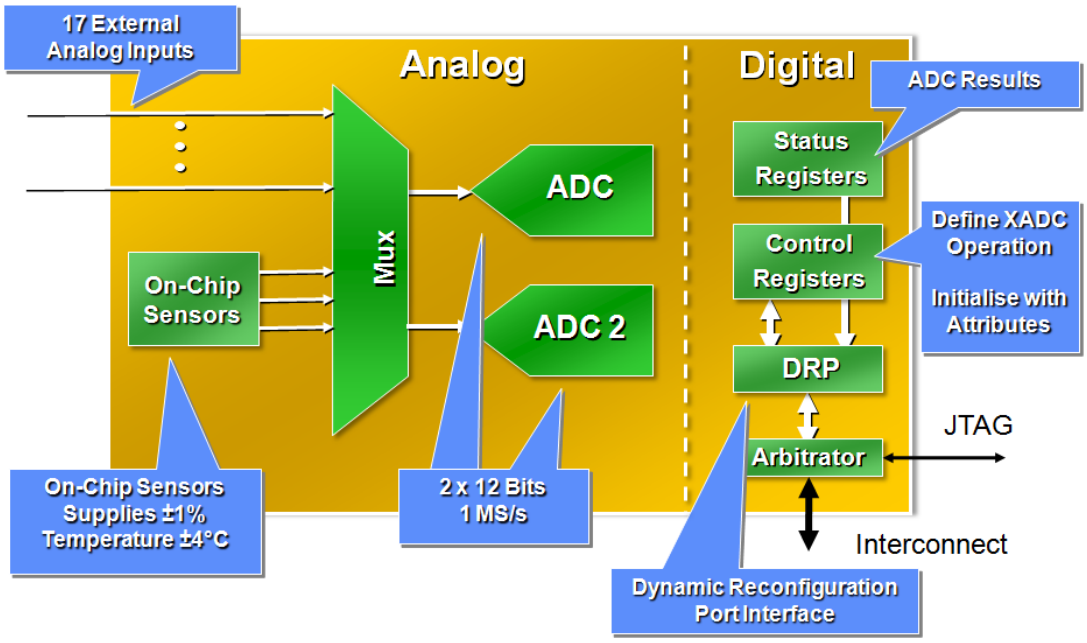
More importantly with the combination of the XADC with the programmable logic and resources in the FPGA, a highly customized analog interface can be implemented.

AMS is the combination of analog and programmable logic (WP398: *Agile Mixed Signal Addresses Analog Design Challenges*).

על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 1-42:

XADC Block Diagram



Slide 1-43:

High Quality ADCs

- High resolution analog-to-digital converters (ADCs)
  - Factory tested and specified 12-bit accuracy with 1V input range
    - 16-bit resolution conversion
    - Built in digital gain and offset correction / calibration
- Fast sampling
  - Conversion time of 1 us with support for simultaneous sampling
  - Flexible timing modes (self and externally triggered sampling modes)
  - Separate track/hold amplifier for each ADC ensures maximum throughput using multiplexed analog input channels
- Flexible analog inputs
  - Differential analog inputs with high common mode noise rejection
  - Support for unipolar, bipolar, and true differential input signal types
- Available in all 7 series devices

The ADCs in the XADC block are fast and precise. These can replace external ADCs in many applications, reducing overall system cost. This can be especially true in systems built using smaller Artix®-7 devices.

Slide 1-44:

Other Features

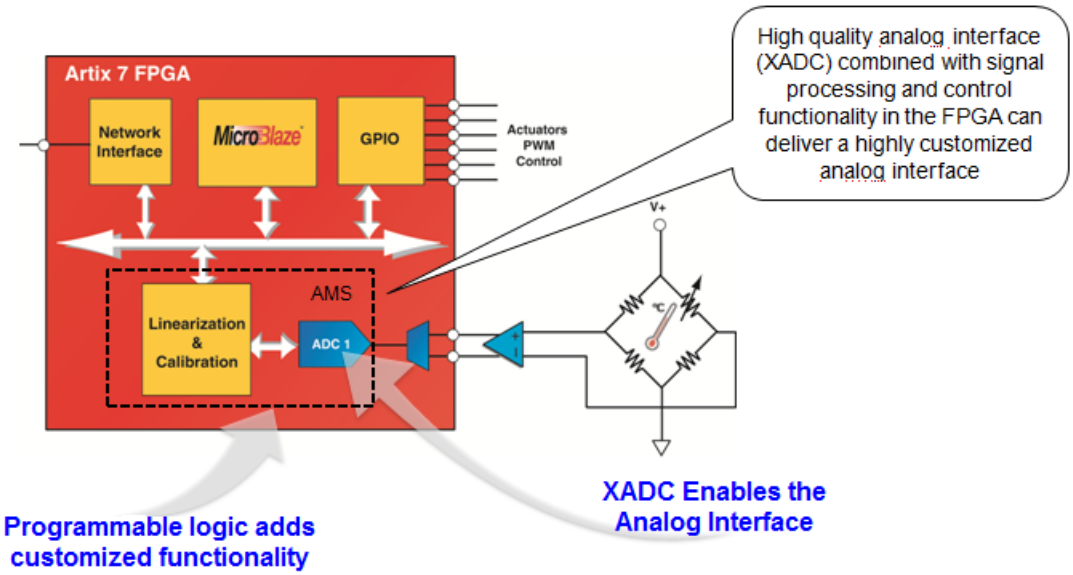
- Internal and external multiplexing and sampling
  - Can sample internal power supplies and temperature
  - Multiplexes internal sources and 17 external analog inputs
  - Can control an external analog multiplexer to reduce pin count
- Flexible triggering
  - Conversion data is stored in internal status registers
  - Internal control registers control source selection, sampling, and alarms
  - Registers can be accessed internally via the dynamic reconfiguration port (DRP)
  - Register can be accessed via JTAG
    - Available on power up, before configuration
    - Accessible through ChipScope™ Pro tool GUI
- Operates over a wide temperature range (–40°C to +125°C)

The XADC can generate alarm outputs whenever one of the internal measurements goes outside bounds set in the control registers.

Slide 1-45:

Analog Mixed Signal Benefits

- Unique customization
- Cost
- Co-verification of analog and digital hardware
- Enhanced reliability, safety, and security



- Unique customization:
  - Programmable logic customizes the XADC to address diverse needs
  - For example: application-specific calibration, linearization, and signal processing
- Cost:
  - Elimination of discrete analog components and associated board area
  - Also eliminates risk of obsolescence and reduces long lead times for proprietary analog components
- Co-verification of analog and digital hardware:
  - Xilinx tools support better verification of the analog interface in the HDL verification process
- Enhanced reliability, safety, and security:
  - On-chip sensors for improved reliability, safety, and security

- Monitoring of the physical environment required in a wide range of applications

These are the main benefits from integrating a quality analog interface block into an FPGA.

The primary value lies in the ability to customize the interface. Integration also reduces the FPGA solution costs for many applications.

The ability to use HDL simulation models for the ADCs and support for adding analog stimulus supports better co-verification of the analog and digital positions of the design. The on-chip sensors provide comprehensive on-chip monitoring for improved reliability.

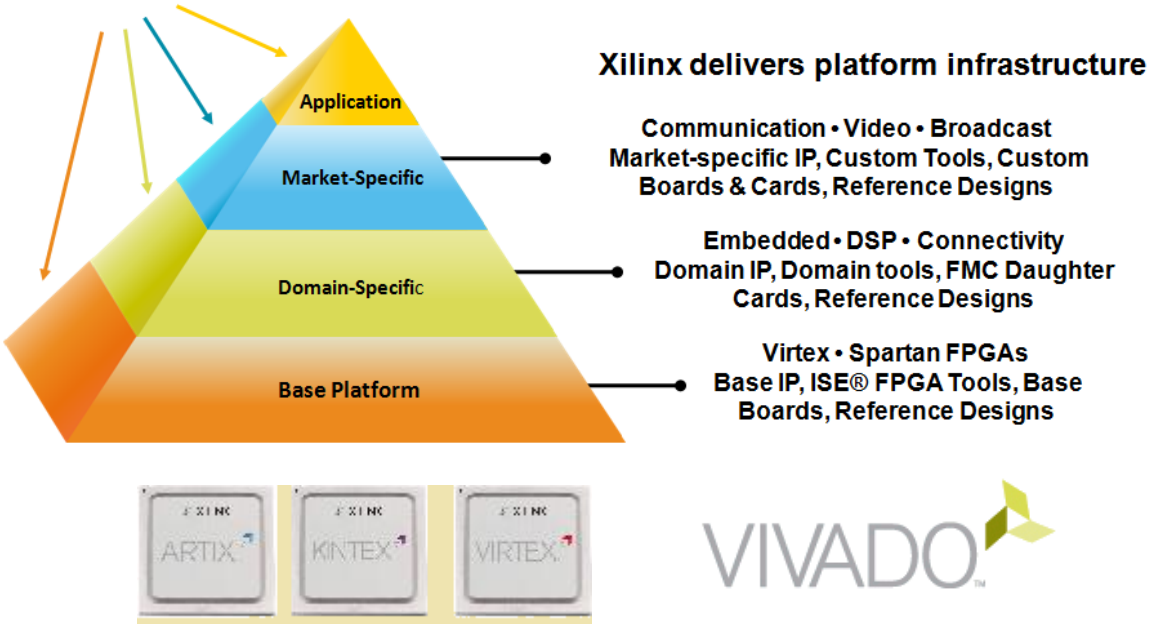
Slide 1-46:

Targeted Design Platforms

- Overview
- Slice Resources
- I/O Resources
- Clocking Resources
- RAM and DSP Slice Resources
- XADC and AMS
- **Targeted Design Platforms**
- Summary

Slide 1-47:

Xilinx Targeted Design Platform Strategy



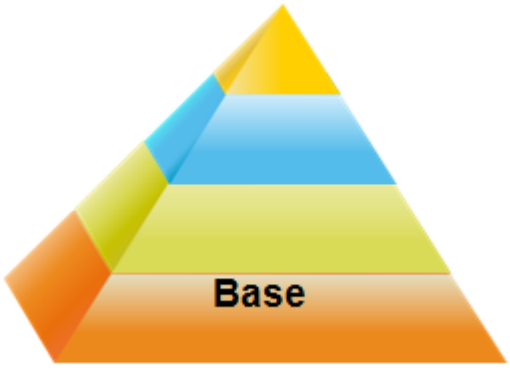
The Xilinx Targeted Design Platform strategy provides custom IP, reference designs, and evaluation boards. This allows designers to focus on differentiating their end product.

Slide 1-48:

Base Kits

- Enable general FPGA-based system design
  - Kintex®-7 FPGA KC705 Evaluation Kit

- Each kit contains
  - An FPGA evaluation board
  - AMS evaluation board
  - Vivado Design Suite with node-locked license
  - Getting Started demo design
  - Reference designs
  - Interactive reference interface
  - Documentation
  - Cables and power supply
  - USB flash drive
    - Contains all reference designs, demos, and documents



Each kit ships with everything you need to start designing.

Interactive reference interfaces work with reference designs, allowing you to easily adjust system parameters and view the results.

על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

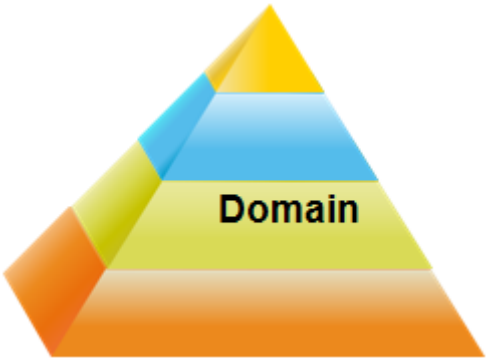
Slide 1-49:

Domain-Specific Kits

- Built upon the base evaluation kits
  - Kintex-7 FPGA KC705 board

- Target key design domains
  - Embedded
  - DSP
  - Connectivity

- Each kit contains
  - Hardware (base board, power supply, cables)
  - Vivado Design Suite
  - Domain-specific IP and reference designs
  - Documentation



Both the base and domain-specific kits can be purchased via **www.xilinx.com** > **Products** > **Boards and Kits**.

Video demonstrations of some kits can be seen via www.xilinx.com/events/webcasts.htm.

Slide 1-50:

Interactive Reference Interfaces

- Enable users to quickly and easily experiment with reference design modules
- Interactive user interface builds on reference design demonstration
- More powerful than a simple design example
- Encourages what-if evaluations to rapidly analyze design options, realize FPGA benefits
- Simplify the silicon and system feature evaluation process
- Quickly adjust system features and measure the results
- Analyze the technical trade-offs
- Are optimized specifically to the environment (domain) of each kit

על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 1-51:

Summary

- Overview
- Slice Resources
- I/O Resources
- Clocking Resources
- RAM and DSP Slice Resources
- XADC and AMS
- Targeted Design Platforms
- **Summary**

Slide 1-52:

Apply Your Knowledge

3) Can you identify the features of a slice?

4) Can you identify the features of a slice?

- Four 6-input LUTs
- Eight registers (four flip-flop / latches and four flip-flops)
- Carry logic
- MUXF7 and F8 multiplexers for combining outputs of multiple LUTs and slices
- Some LUTs can implement a shift register or distributed memories

Slide 1-53:

Summary (1)

- The 7 series 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 7 series FPGA IOBs contain DDR registers as well as SERDES resources
- The SelectIO™ interfaces enable direct connection to multiple I/O standards
- The 7 series FPGA includes dedicated block RAM and DSP slice resources

Slide 1-54:

Summary (2)

- The 7 series FPGAs includes dedicated MMCMs, PLLs, and routing resources to improve your system clock performance and generation capability
- The 7 series FPGAs include other dedicated hardware that is not described in this module
- The Xilinx Targeted Design Platform strategy provides kits that allow you to focus on product differentiation

על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 1-55:

Where Can I Learn More?

- **www.xilinx.com**
  - Check out the 7 series FPGA user guides and data sheets
  - In the tools, select **Help** > **Xilinx on the Web** > **Support and Services**

- **www.xilinx.com/training**
  - *Vivado Design Suite Static Timing Analysis and Xilinx Design Constraints* and *Designing with the 7 Series Families* courses
  - Learn about free recorded e-learning modules and other course offerings
  - In the tools, select **Help** > **Xilinx on the Web** > **Support and Services** > **Training**

Capture Your Notes Here

© Copyright 2019 Xilinx

Introduction to Vivado Design Flows

Slide 2-1:

```
108167**slide

2019.1
108167!*note
```

This module provides an overview of design flows within the Vivado Design Suite. 15 minutes.

Slide 2-2:

Objectives

```
108168**slide
```

After completing this module, you will be able to

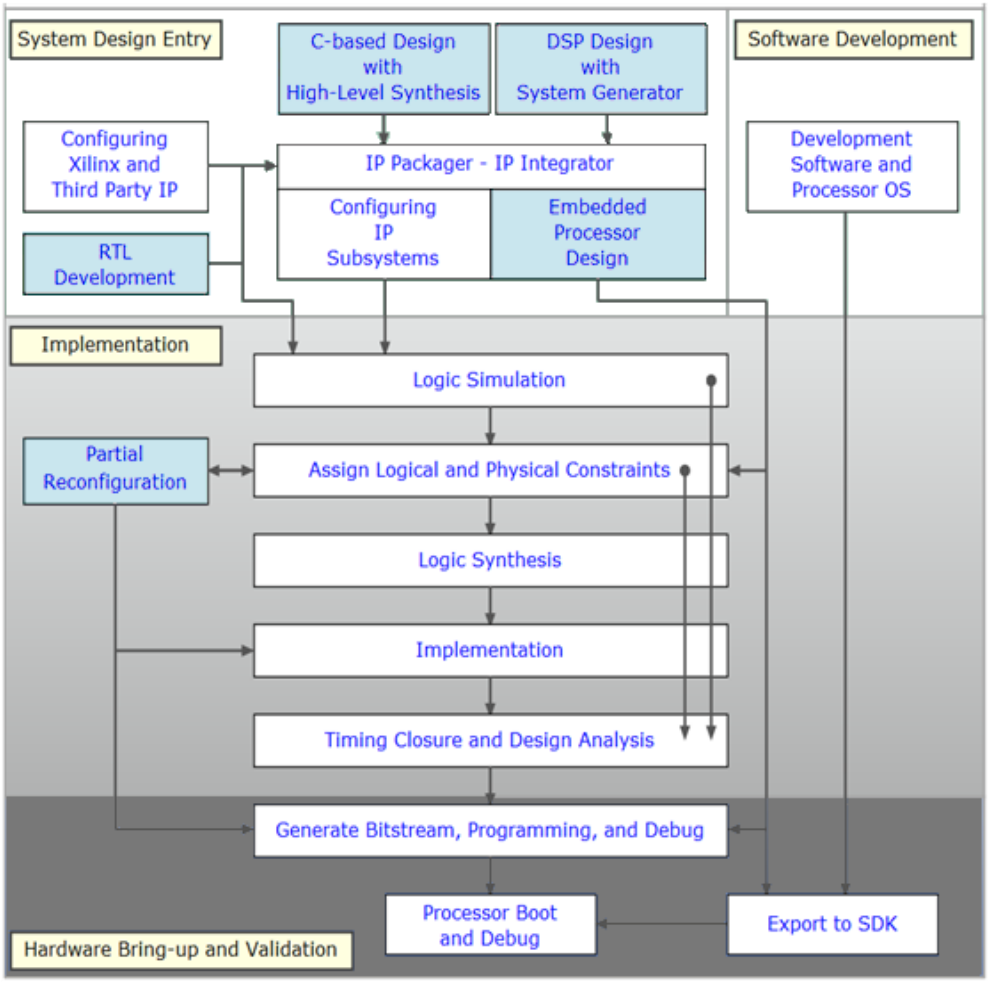
- Describe various design flows in the Vivado Design Suite
- Explain how the Vivado tool flow is different from ISE tool flow
- Identify and describe the supported use models in the Vivado Design Suite

© Copyright 2019 Xilinx

Slide 2-3:

Vivado Design Flows

128336**slide



128336!*note

© Copyright 2019 Xilinx

The above figure shows the high-level design flow in the Vivado Design Suite.

Different design flows supported in the Vivado Design Suite:

- RTL and IP: using HDL sources - Verilog, VHDL, System Verilog
- High-Level Synthesis (HLS) C-based Design
- DSP design using System Generator
- Embedded Processor Design
- Partial Reconfiguration Design

The implementation and analysis techniques shown in the figure apply to all the design flows.

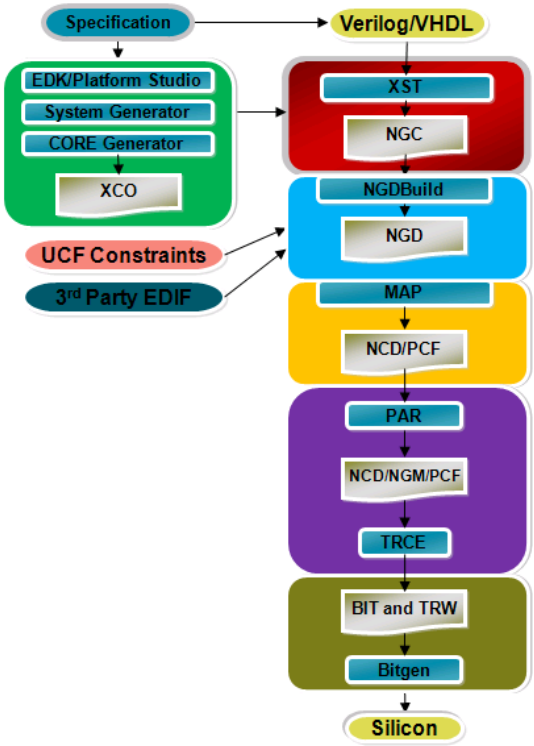
© Copyright 2019 Xilinx

Slide 2-4:

ISE Design Suite Tool Flow (1)

108173**slide

- RTL through bit file control
- Standalone point tools used at various stages
  - System design: Platform Studio, System Generator
  - RTL: CORE Generator™ tool, ISim, PlanAhead™ tool
  - NCD: FPGA Editor, Power Analyzer, ISim, PlanAhead tool
  - Bit file: ChipScope™ analyzer, iMPACT



108173!*note

All of these point tools and technologies are integrated into the ISE Design Suite.

The Vivado Design Suite replaces ISE point tools like ChipScope, iMPACT, and CORE Generator. These point tools are built directly into the Vivado Design Suite.

© Copyright 2019 Xilinx

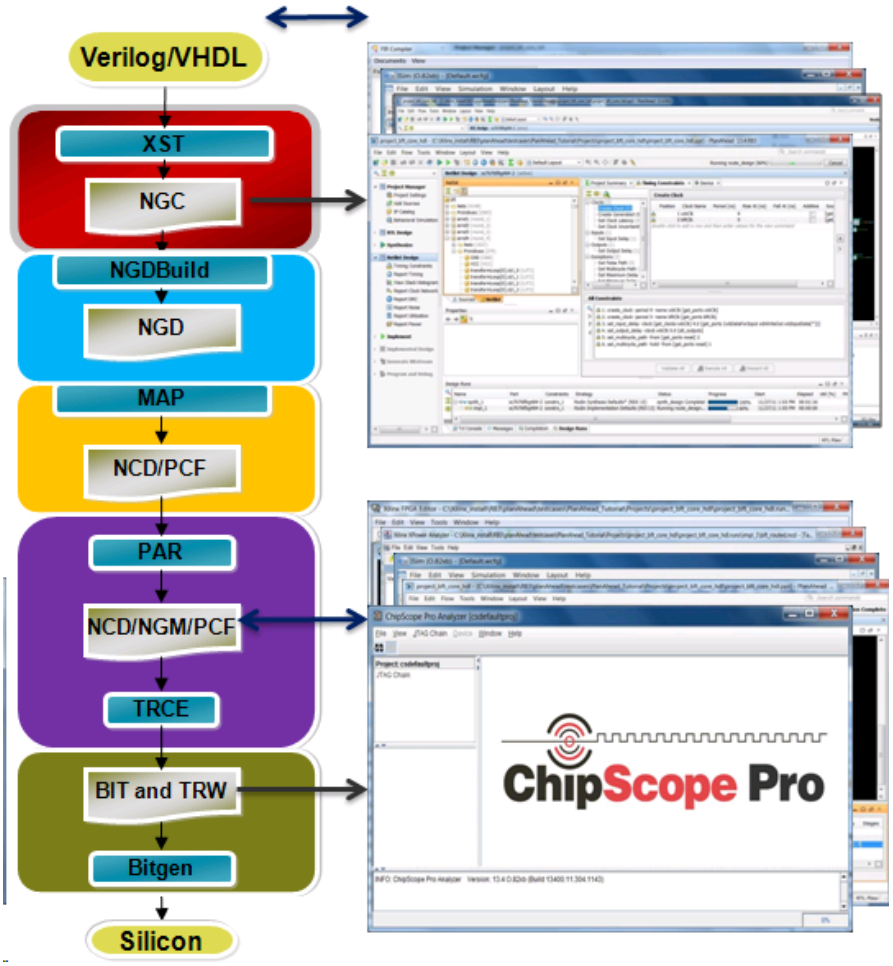
על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 2-5:

ISE Design Suite Tool Flow (2)

108174**slide

- Each step requires a different data model (NGC, NGD, NCD, NGM)
  - Constant constraint and data translation
  - Run-time and memory penalties
  - Ease-of-use impact
- No post-implementation timing changes
  - No backward compatibility for cross-probing



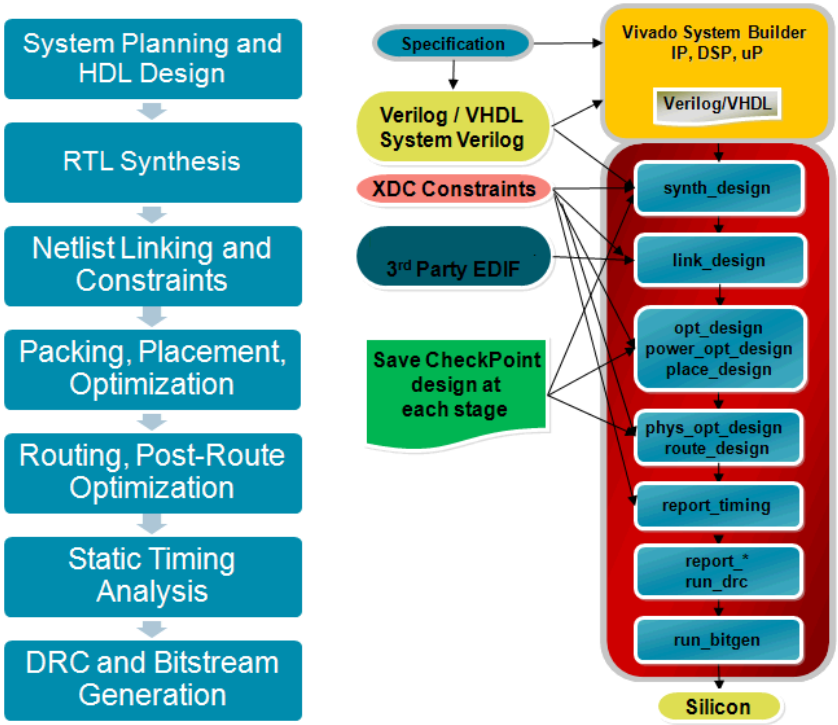
© Copyright 2019 Xilinx

Slide 2-6:

Vivado Design Suite Tool Flow (1)

108175**slide

- Interactive IP plug-n-play environment
  - AXI4, IP_XACT
- Common constraint file (XDC) throughout flow
  - Apply constraints at any stage
  - Real-time debug
- Design analysis and reporting
  - Cross-probing
  - Report generation at any stage
  - Robust Tcl API



108175!*note

The Vivado Design Suite is an IP and system-centric design environment that allows you to specify source files and add IP modules to your design. It enables you to take your design from RTL creation to bitstream generation.

The entire design process can be executed in-memory without any need of having intermediate file formats. All the process are written into tool command language (Tcl) interface.

The steps or options available in the integrated design environment of the Vivado Design Suite are accessible either from the GUI or through Tcl commands.

© Copyright 2019 Xilinx

The Vivado Design Suite includes a timing-driven synthesis tool that uses a common constraint language throughout the flow from synthesis all the way through place and route.

Synthesis stage (`synth_design`) is same except that it can now read XDC (Xilinx Design Constraints) into synthesis. Netlist linking is done in `link_design`. Here any third-party netlist files EDIF can be read into the Vivado Design Suite.

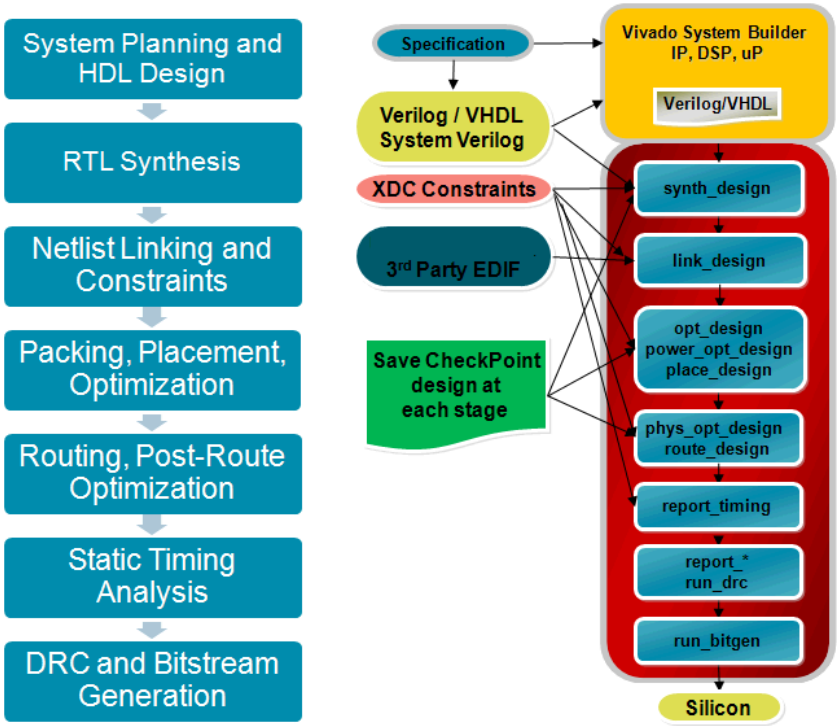
`opt_design`, `power_opt_design`, and `place_design` are the packing and placement functions. `power_opt_design` is an optional process.

Slide 2-7:

Vivado Design Suite Tool Flow (2)

108176**slide

- Common data model throughout the flow
  - Cross-probing support
- Save checkpoint designs at any stage
  - Netlist, constraints, place and route results



108176!*note

© Copyright 2019 Xilinx

`phys_opt_design` is an optional stage. At this stage, timing-driven optimization is performed on the failing paths of the design.

All the placed cells need to be connected. `route_design` performs this routing between the cells.

The single data model in memory allows you to do a lot of things, such as cross-probing and fixing your timing at the end of the implementation without having to reimplement again.

The Vivado IDE allows you to save design checkpoints and generate reports at all the stages of the design flow. `report_timing` enables you to generate timing reports of the design.

After the successful implementation of the design, the next step involves generating a bitstream and programming the FPGA device with your design.

© Copyright 2019 Xilinx

Slide 2-8:

ISE vs. Vivado Design Suite

108172**slide

ISE Tool Implementation Flow: Executables	Vivado Tool Implementation Flow: Tcl Commands	
ngdbuild	link_design*	Translate design, apply constraints
map	opt_design*	Logic Optimization
	power_opt_design*	Power Optimization
	place_design	Placement
	phys_opt_design*	Physical Optimization
par	route_design	Routing
trce	report_timing_summary	Timing Analysis
bitgen	write_bitstream	Bitstream Generation

* Optional Step

108172!*note

If you are familiar with the ISE tools implementation flow, this graphic provides more detail on the pieces that have changed. In general, the ISE tools flow consists of a sequence of programs that run and communicate using files. The Vivado tool flow is a series of Tcl commands that operate on a single in-memory database. This allows more flexibility and interaction.

© Copyright 2019 Xilinx

Slide 2-9:

Different Users – Different Needs

108178**slide

- FPGAs are used in different applications
  - Small designs for *glue*
  - Small designs for low to mid-volume consumer applications
  - Medium designs for specialty consumer applications
  - Large designs for industrial applications
  - Very large designs for high-end infrastructure
  - Extremely large designs for ASIC prototyping

- Each of these applications has differing needs
  - Different size teams
  - Different levels of FPGA architectural skill
  - Different levels of project infrastructure management

- Need different flows to support all users
  - Simpler pushbutton flow
  - Low overhead, completely automated, and supporting large teams / projects

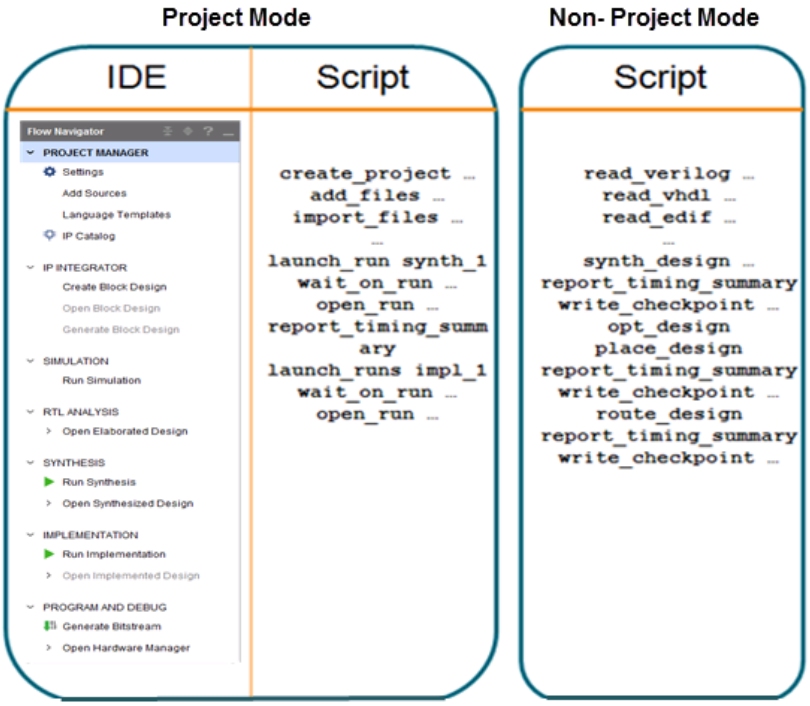
© Copyright 2019 Xilinx

Slide 2-10:

Vivado Design Suite Use Models

108202**slide

- The Vivado tool supports two use models
  - Project mode
  - Non-project mode
- The Vivado IDE can be used both in project mode and non-project mode
- Project mode
  - Creates project structure on disk
  - The Vivado tool manages design runs, status, and data
  - Can use GUI and/or Tcl scripts
  - Reports are generated automatically
  - Robust interactive design environment
- Non-project mode
  - Project is compiled in memory
  - Tcl-based, batch processing
  - Manual design source file management
  - Can use GUI for visualization via start_gui command
  - Must manually create reports and checkpoints



© Copyright 2019 Xilinx

Slide 2-11:

Project Mode vs. Non-project Mode

108203**slide

Project	Non-Project
High degree of automation • Flow: a single Tcl command – multiple flow steps • Source and result management • Multiple strategies: easy run	Fine-granularity commands • More efficient scripts. Enable focusing on specific challenges • Easy access to all features (ex: re-entrant routing) File structure: fully customizable • Sources and result files • Easy adoption to revision control systems Runtime: potentially faster
Flow • Reduced flexibility to work with intermediate implementation results • Not all features are easily accessible (ex: re-entrant routing) File structure: fixed file organization • Revision control – many project files	Requires more in-depth knowledge You are responsible for • Dependency file management • Saving key result files (ex: checkpoints)

108203!*note

The project mode allows the GUI to handle all of your project management and file construction (easier).

The non-project mode provides the fastest possible results but is executed entirely with script. Even though you can still use the GUI for design analysis, it will have to be opened manually.

Note that the Tcl commands for each mode are very different and cannot be shared between project mode and non-project mode. It is important that the commands are not mixed between these use models. Refer to the *Vivado Design Suite Tcl Command Reference Guide (UG835)* for further details.

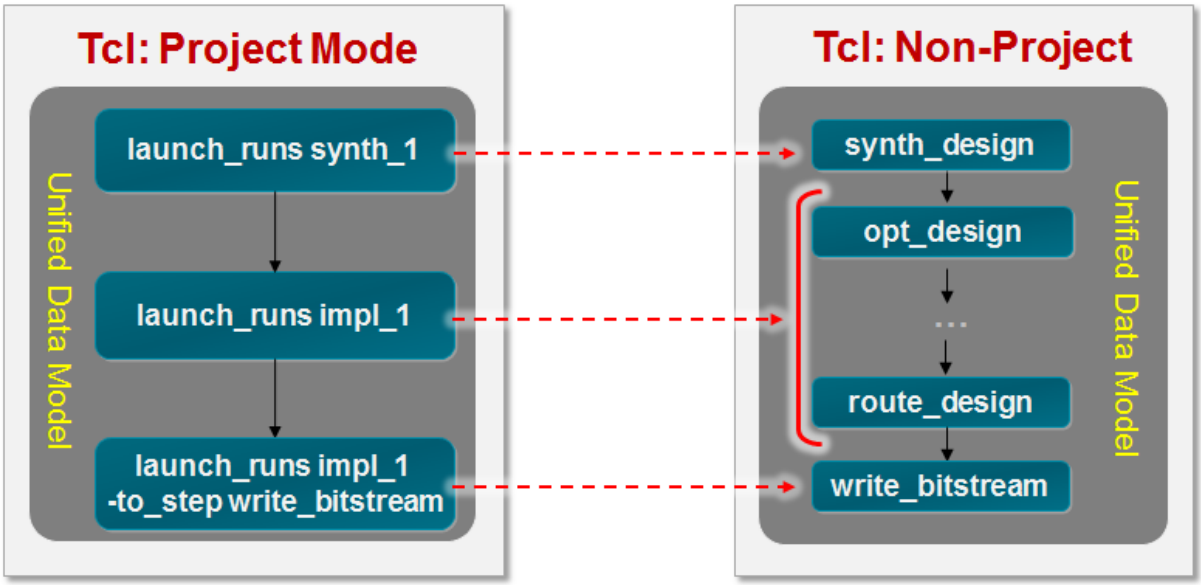
© Copyright 2019 Xilinx

Slide 2-12:

Project Mode vs. Non-project Mode Tcl Commands

108204**slide

- Tcl commands correspond to project mode and non-project mode



Higher-level abstraction
Automation = fewer steps

Lower-level commands
Finer control

© Copyright 2019 Xilinx

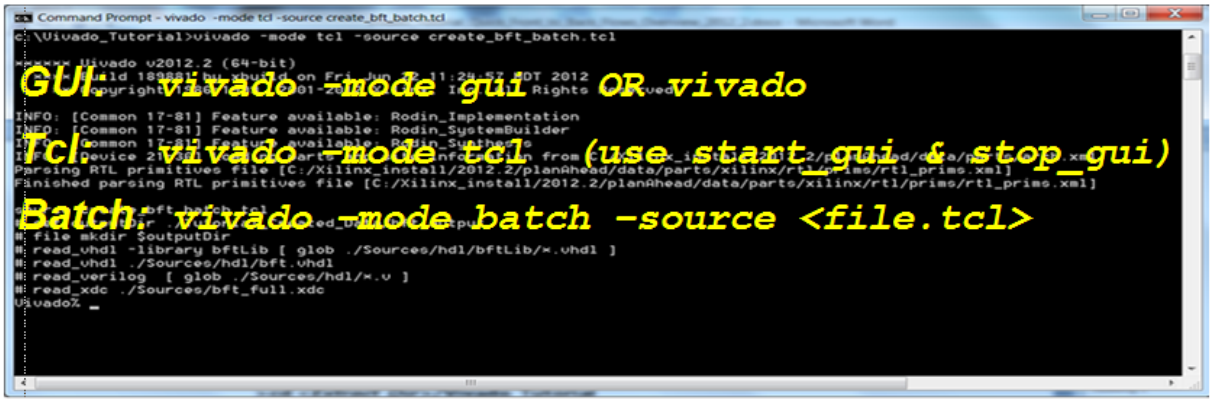
Slide 2-13:

Launching the Vivado IDE

108205**slide

- The Vivado IDE can be launched two ways
  - From the desktop or Start menu
  - From the command line

- When launching from the command line, there are three modes
  - GUI: launches the IDE (LOG / JOU file stored in launching directory)
  - Tcl: launches into interactive Tcl console
  - Batch: launches and executes commands in sourced Tcl script



108205!*note

GUI

Usage: `vivado -mode gui` OR `vivado`

Tcl

Usage: `vivado -mode tcl`

Can use `start_gui` and `stop_gui` to enter and exit IDE for visualization

Batch

Usage: `vivado -mode batch -source <file.tcl>`

Options include:

© Copyright 2019 Xilinx

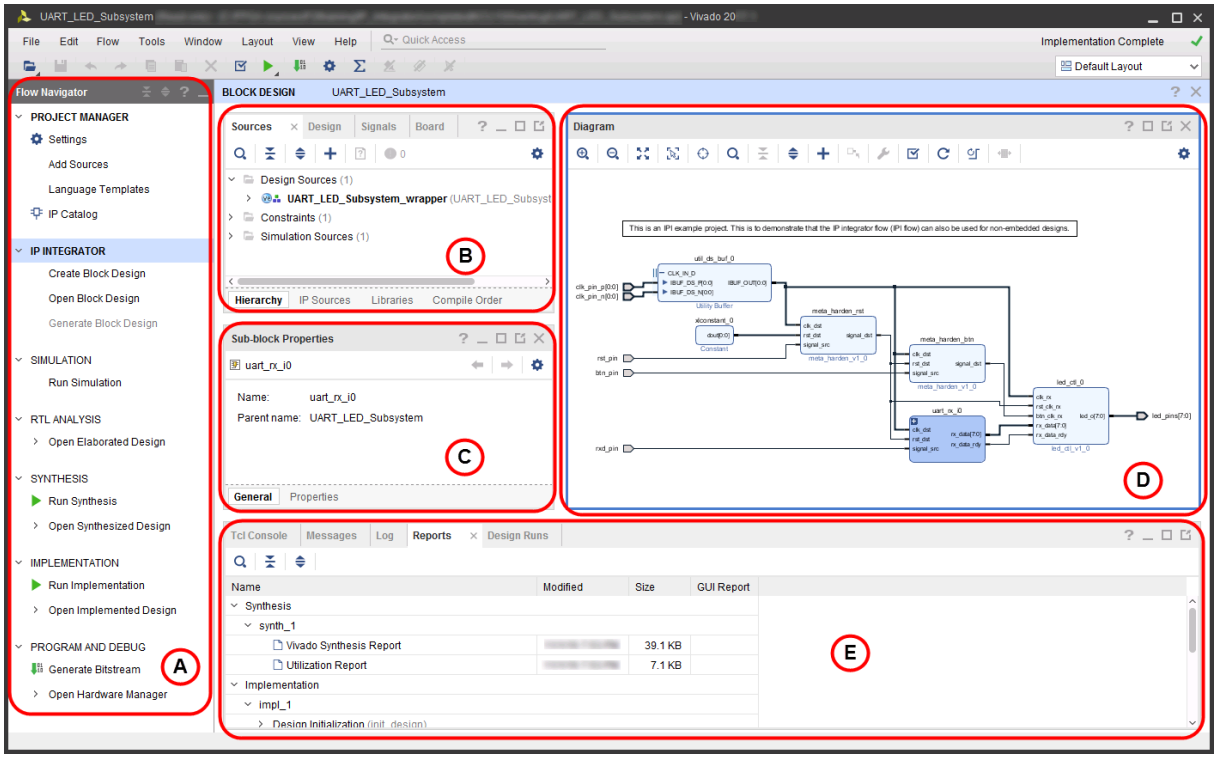
- `-mode`: invocation mode: `gui`, `tcl`, and `batch`. Default: `gui`
- `-source`: Source the specified Tcl file. Required for batch mode
- `-journal`: Journal file name. Default: `vivado.jou`
- `-appLog`: Open log file in append mode
- `-log`: Log file name. Default: `vivado.log`
- `-m32`: Run 32-bit binary version
- `-m64`: Run 64-bit binary version
- `-version`: Output version information and exit
- `-tclargs`: Arguments passed on to Tcl `argc argv`
- `-tempDir`: Temporary directory name
- `-verbose`: Suspend message limits during command execution

Slide 2-14:

Integrated Design Environment (IDE)

111058**slide

A: Flow Navigator
B: Sources
C: Block Properties / Attributes
D: Workspace (IP integrator shown here)
E: Console



© Copyright 2019 Xilinx

111058!*note

The Flow Navigator (A) provides access to all the high-level operations that can be performed on the design.

The Sources window (B) is divided into tabs. The Sources tab enumerates all the sources present in the design and organizes them in a variety of fashions including hierarchical, as IP sources, libraries, and by compile order. The Design Hierarchy tab appears after synthesis and lists out the netlists generated in a hierarchical fashion.

The Block Properties window (C) enables users to observe, and if possible, change the attributes and properties of the selected IP block.

The main work window (D) contains content relevant to the task selected. Generally it contains source files either in text form (in an editor window) or graphically as seen here in the IP integrator.

The Console window (E) acts as a Tcl console, message display, log file, and report access point.

Slide 2-15:

Journal and Log Files

108574**slide

- Journal file (*vivado.jou*)
  - Contains just the Tcl commands executed by the Vivado IDE
- Log file (*vivado.log*)
  - Contains all messages produced by the Vivado tool, including Tcl commands and results, info/warning/error messages, etc.
- Location
  - Linux: directory where the Vivado IDE is invoked
  - Windows via icon: *%APPDATA%\Xilinx\Vivado* or *C:\Users\<user_name>\AppData\Roaming\Xilinx\Vivado*
  - Windows via command line: directory where the Vivado IDE is invoked
  - From the GUI you can access via

© Copyright 2019 Xilinx

- Select **File** > **Open Log File**
- Select **File** > **Open Journal File**

© Copyright 2019 Xilinx

Slide 2-16:

Summary

111059**slide

- The Vivado tool replaces all the ISE point tools like CORE Generator, Chipscope, Project Navigator
  - All these capabilities of ISE are integrated in the Vivado Design Suite
- The Vivado tool uses a common data model throughout the FPGA design process, unlike the ISE tools
  - This yields runtime and memory resource benefits to the user
- The Vivado Design Suite supports two use model - Project mode and Non-project mode
  - The Vivado tool supports the use of Tcl commands for all processes

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Vivado Design Suite Project-based Flow

Slide 3-1:

```
108180**slide

2019.1
108180!*note
```

This module introduces the project mode within the Vivado® Design Suite. 20 minutes.

© Copyright 2019 Xilinx

Slide 3-2:

Objectives

120200**slide

After completing this module, you will be able to:

- Describe the project mode use model in the Vivado Design Suite
- Describe the structure and files of a project
- Create a simple Vivado Design Suite project in project mode

© Copyright 2019 Xilinx

Slide 3-3:

Project Mode

128517**slide

- In project mode, the Vivado IDE offers a Graphical User Interface (GUI)
- Provides predefined design flow steps in the GUI
- Creates a project directory structure on disk
- The design process and design data are managed automatically by the Vivado tool
- Can use the Vivado IDE or Tcl commands and scripts to run any process

128517!*note

From the Flow Navigator, you can launch predefined design flow steps like Synthesis, Implementation, or Generate Bitstream using push buttons.

The source files, process run results, reports, and project status are managed automatically by the Vivado Design Suite.

Slide 3-4:

What is a Project?

108206**slide

- A project is an encapsulation of the design process
  - Repository of all files that are part of the design process
  - Repository of intermediate and final design netlists
  - Project data to link all the elements together

© Copyright 2019 Xilinx

- State information that stores the current state of the project
  - What processes have been run

108206!*note

As implied by the name, non-project mode has no concept of project.

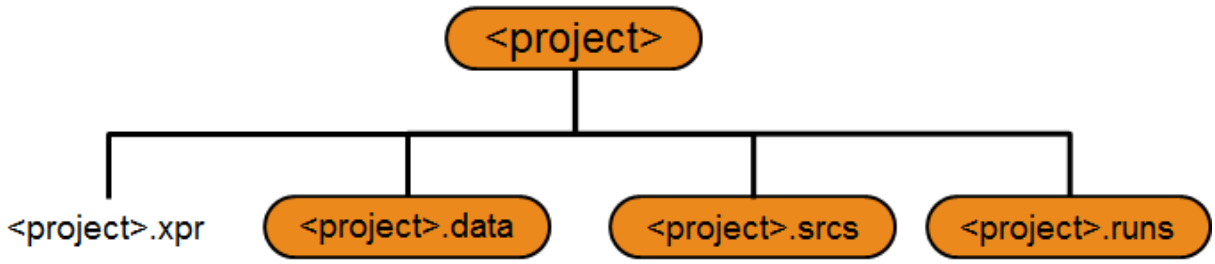
© Copyright 2019 Xilinx

Slide 3-5:

Project Directory

120199**slide

- The project is created by using the **Create Project** option in the IDE or with the `create_project` command
  - Creates the project directory and associated sub-directories
- The project directory holds all (or most) of the files associated with the project



120199!*note

The subdirectories of the project directory are created automatically as needed by the tool. These directories hold specific content and will be described in subsequent slides.

All project data is stored in a project_name directory containing the following

- project_name.xpr file: Object that is selected to open a project (the Vivado IDE project file)
- project_name.runs directory: Contains all run data
- project_name.srcs directory: Contains all imported local HDL source files, netlists, and XDC files
- project_name.data directory: Stores floorplan and netlist data

© Copyright 2019 Xilinx

Slide 3-6:

Project-based Flow

108183**slide

- Robust interactive design environment
  - Single, consistent interface – no point tools
  - Source and run management
  - IP configuration, implementation, packaging
  - Analysis and verification with cross probing
  - I/O, clock, timing, and floorplanning constraint assignment
  - Debug core insertion and analysis
  - Routing analysis
  - Bit file generation and programming
- Runs manager
  - Multiple runs configuration and management

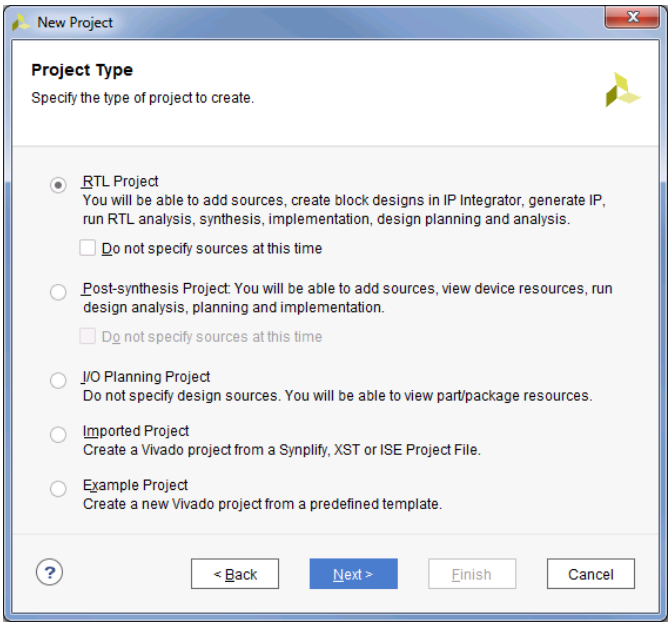
Slide 3-7:

New Project Creation Wizard

108187**slide

© Copyright 2019 Xilinx

- Five different types of project can be created
  - RTL
    - Using HDL source files
  - Post-synthesis
    - EDIF or NGC
  - I/O planning
    - For early pin testing
    - No design sources
  - Imported project from Project Navigator, XST, and Synplify projects
  - Example project
    - Using available design templates



- Next, add source files, constraints files, and select the target device

108187!*note

© Copyright 2019 Xilinx

Project Manager:

- The Project Manager environment is used to manage sources, customize IP, and view project details in the Project Summary.
- The Project Manager does not perform design compilation, and no data is loaded into memory. A design must be open to have access to the analysis commands.

RTL Project:

- The RTL Project environment enables analysis of the compiled RTL design including logic exploration, resource and power estimation, DRCs, and I/O pin planning.

Post-Synthesis Project:

- This is where pre-implementation design analysis, debug core insertion, and constraint definition is performed. You can explore various constraints sets, devices, and implementation options.

I/O Planning Project (without design sources):

- The Vivado IDE allows you to create an empty project that enables early device exploration and I/O port configuration.
- This allows for early pinout definition to eliminate pinout-related changes that typically happen late in the design cycle.
- When creating the project, you can create I/O ports manually or import them from comma separated values (CSV).

Import project:

- Imports RTL project data from ISE or Synplify to migrate to the Vivado Design Suite project
- The Implemented Design environment loads the netlist and implementation results directly from the run directory to ensure that the viewed data represents the launched run.

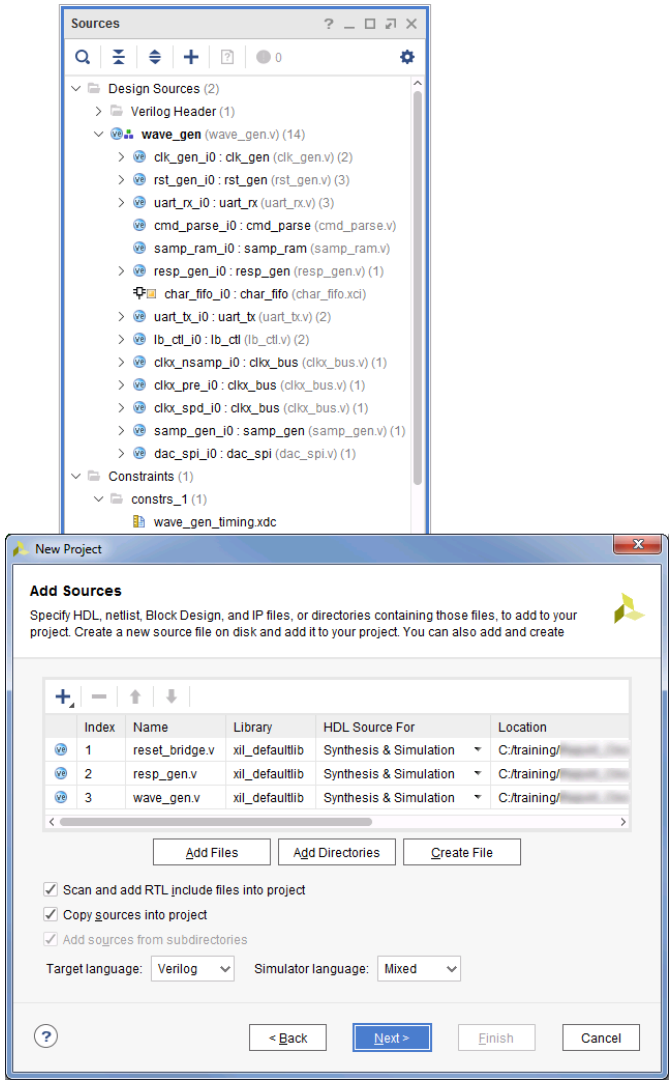
© Copyright 2019 Xilinx

Slide 3-8:

Creating a Project with HDL Sources

108188**slide

- Define the project name and project location
- Select source files, implementation results, project directories, etc.
  - All recognized source files, including Verilog, VHDL, SystemVerilog and constraint files, IP files in the directories, and subdirectories are added
  - Reference original existing files or import and copy them into the project
- Select the family and target device
- Sources are added to the project, not read as in non-project batch mode



108188!*note

© Copyright 2019 Xilinx

Vivado IDE projects can be created using Verilog or VHDL or SystemVerilog sources. The HDLs can be edited, modified, elaborated for further analysis, and run through synthesis.

You can also choose to write RTL code rather than adding the HDL sources. HDL templates are available in the text editor under Tools > Language Templates.

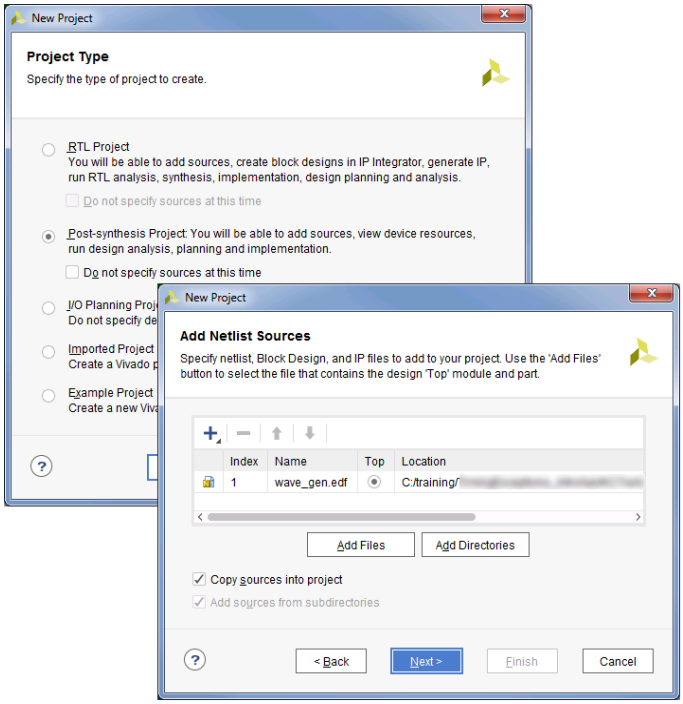
This is also the entry point if you are creating a project around an implementation result. In that case, there might not be any HDL, but an implementation directory for design analysis.

© Copyright 2019 Xilinx

Slide 3-9:

Creating a Project with a Synthesized Netlist

120908**slide



- Define the project name and project location
- Select the top-level netlist
  - Define search path directories for lower level netlists
  - Can import files and libraries from an XST or Synplify project
  - Can create a project without a netlist for I/O planning
- Automatically converts NGC and NGO files to EDIF with NGC2EDIF
  - For cores, the original NGC and NGO files are used during implementation
  - For a top-level NGC file generated from XST, the NGC2EDIF output is used during implementation

120908!*note

© Copyright 2019 Xilinx

Note that even though you can import a Synplify project, you can only synthesize with a Xilinx synthesis tool.

Treatment of NGC core netlists has changed from the PlanAhead™ software. Previously, the PlanAhead software passed the original NGC core netlists to the ISE software. With the Vivado IDE, the ngc2edif utility is used to convert these to EDIF netlists that are passed to the database during the link phase. This allows the Vivado logic analyzer to probe into EDIF cores and enables benefits.

NGC files are not supported for UltraScale device designs.

© Copyright 2019 Xilinx

Slide 3-10:

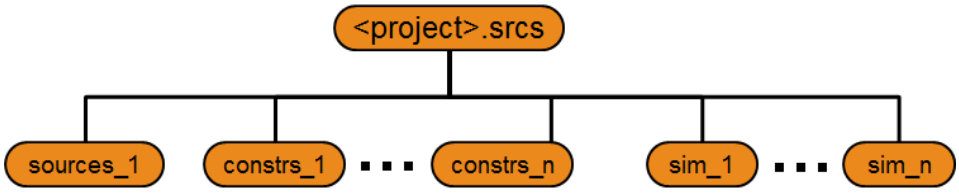
Project Filesets

111879**slide

- A project contains a large number of files

- Files are used for specific purposes and are kept in different filesets
  - There are different types of filesets
    - Sources fileset (*sources_1*)
      - Contains sources used for both simulation and synthesis
    - Constraints filesets (i.e., *constrs_1*)
      - Contains constraint files; used for synthesis and/or implementation only
    - Simulation filesets (i.e. *sim_1*)
      - Contains HDL files used for simulation only (test bench files)
  - You can have multiple filesets of some types (simulation and constraint)

- Filesets are Vivado Design Suite objects and are associated with information on disk



111879!*note

© Copyright 2019 Xilinx

When a project is created, the filesets *sources_1* and *constr_1* are created automatically.

Other constraint filesets can be created using the `create_fileset -constrset` command. This allows you to use different constraints during different runs.

Simulation filesets can be created using the `create_fileset -simset` command. You can create multiple simulation filesets.

Manually created filesets can be called by any name (does not need to be *constrs_2*, *sim_2*).

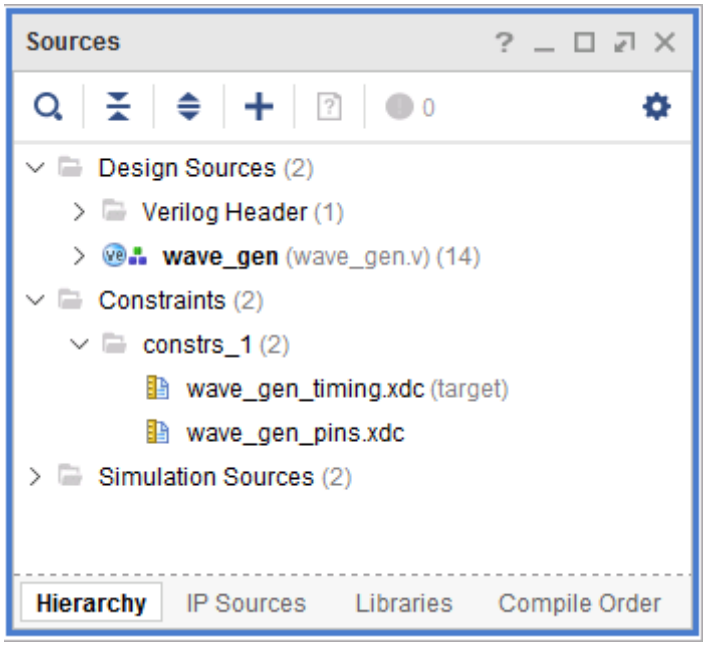
© Copyright 2019 Xilinx

Slide 3-11:

Constraint File Management

120270**slide

- Constraint sets are a collection of XDC files
  - A project can contain multiple constraint sets but all must be of the same type
  - For a constraints set to be applied, it must be set to *active*
  - Any constraint set can be made active by right-clicking and selecting *Make Active*



- Target XDC
  - The XDC file in a constraint set to which NEW constraints are written
  - Modified existing constraints are written back to the original constraint file

120270!*note

The Vivado Design Suite does not support the use of UCF or NCF constraints.

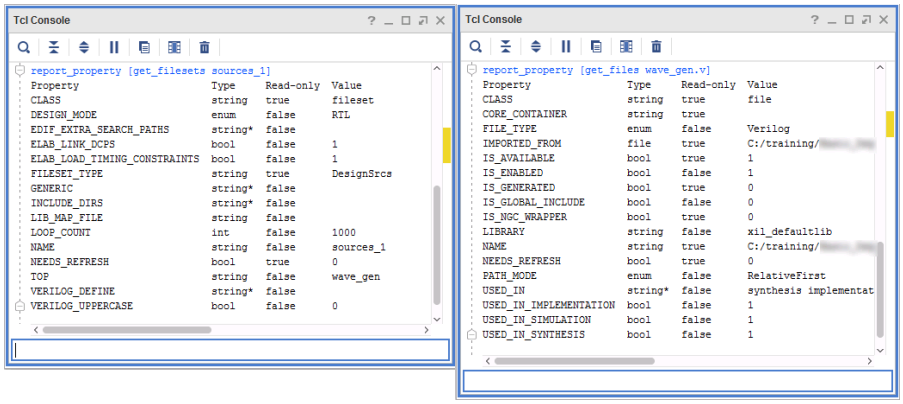
© Copyright 2019 Xilinx

Slide 3-12:

Filesets and Files as Objects

111880**slide

- Both filesets and files are Vivado Design Suite objects
  - Filesets are found using the get_filesets command
  - Files are found using the get_files command
- Most fileset and file management operations can be done through the GUI
- Like all objects, files and filesets have properties
  - Properties can be reported, queried, and in some cases even modified
- Files belonging to a fileset can be found using the get_files -of_objects <fileset> Tcl command



111880!*note

Files and filesets are manipulated in the same way as other objects in the Vivado Design Suite.

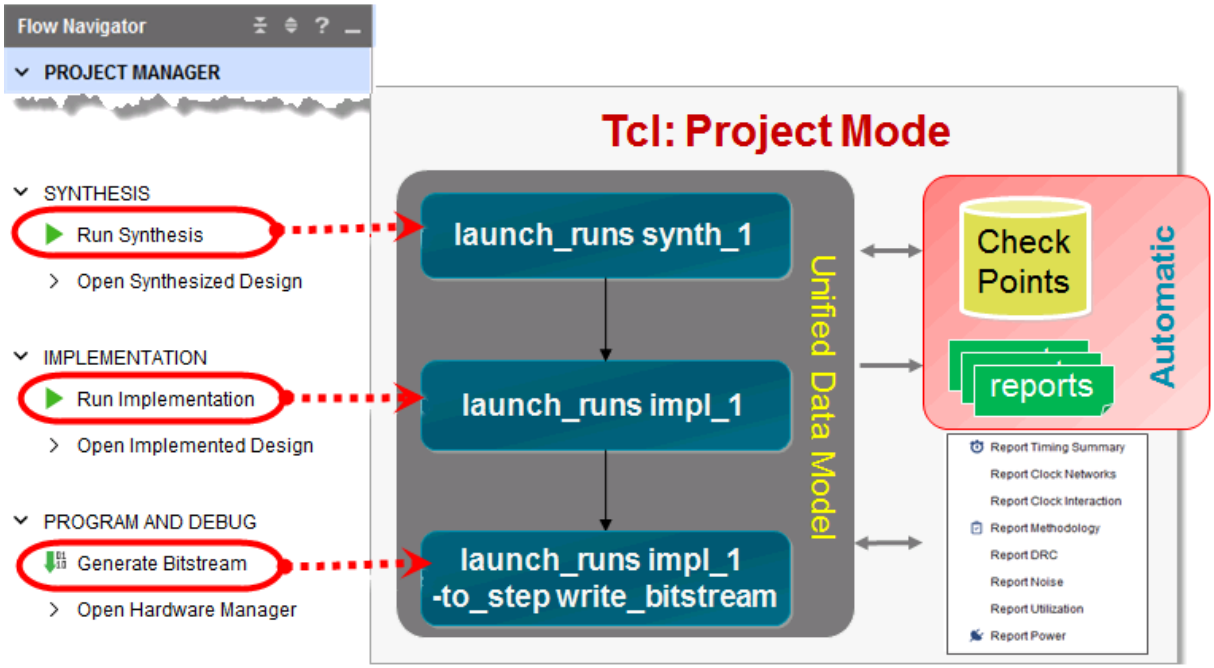
© Copyright 2019 Xilinx

Slide 3-13:

Project Mode: GUI → Tcl

108185**slide

- Key flow steps



108185!*note

In the GUI, the flow steps are nicely ordered in the Flow Navigator. Each step corresponds to a high-level Tcl command that can be used for scripting.

The results of the `launch_runs` commands are checkpoints and reports. `launch_runs` Tcl commands are used to manage status.

Runs are created with strategies and constraint sets.

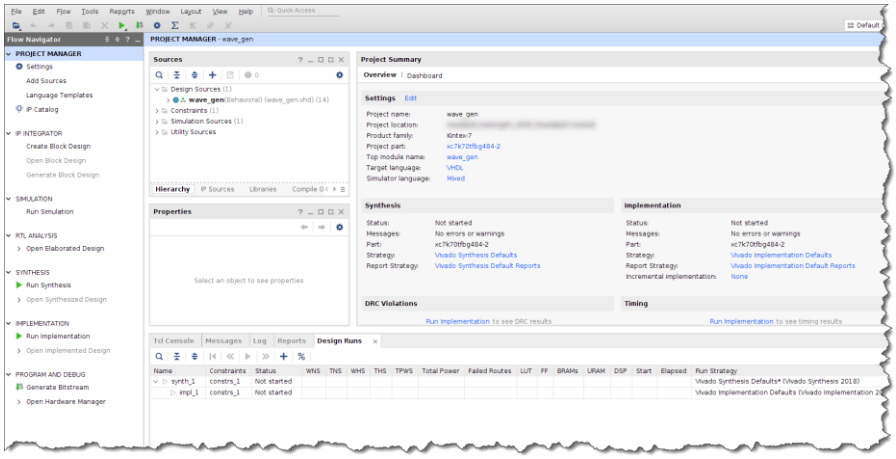
Running simulation from the Flow Navigator (i.e., **Simulation** > **Run Simulation**) is equivalent to using the `launch_simulation` Tcl command.

© Copyright 2019 Xilinx

Slide 3-14:

Project Manager

108191**slide



- Used to manage sources, customize IP, and view project details in the Project Summary
- Flow Navigator
- Sources view
  - Hierarchical display of sources
  - IP Sources and Libraries view
    - EDIF, NGC, or HDL, including references to library and location
  - Gives access to constraints file
- Tcl Console, Messages, Log, Reports, and Design Runs
- Project Summary
  - Gives access to device utilization (resources), timing summary, and strategy information
- Introducing project summary dashboards in the Vivado IDE
  - Enable you to view and analyze all of this information in one place

108191!*note

The Project Manager is used to manage sources, customize IP, and view project details in the Project Summary. In this case, an RTL design was created.

RTL Design:

© Copyright 2019 Xilinx

- The RTL Design environment enables analysis of the compiled RTL design, including logic exploration, resource and power estimation, DRCs, and I/O pin planning.

Post-Synthesis Design:

- This is where pre-implementation design analysis, debug core insertion, and constraint definition is performed. You can explore various constraints sets, devices, and implementation options.

Implemented Design:

- The Implemented Design environment now loads the netlist and implementation results directly from the run directory to ensure that the viewed data represents the launched run.

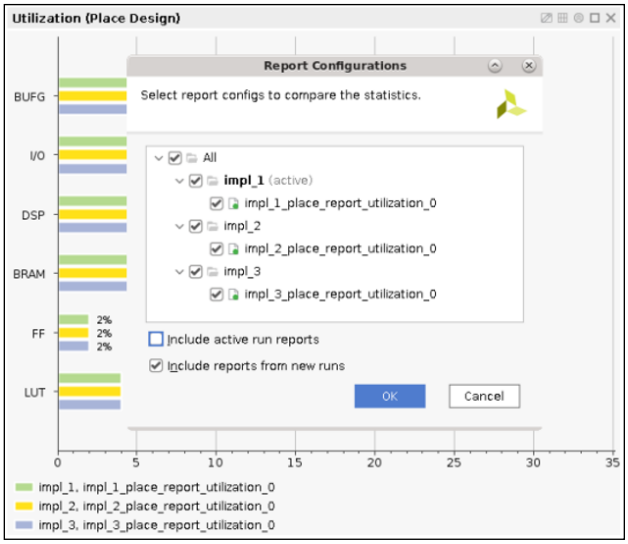
Slide 3-15:

Vivado Design Suite GUI

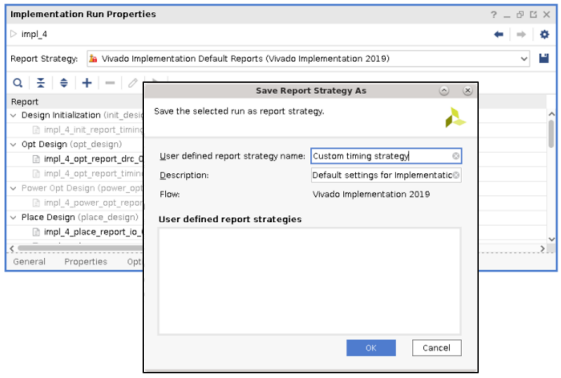
155198**slide

© Copyright 2019 Xilinx

- Dashboards
  - Project summary dashboards enable you to view and analyze all of this information in one place
  - Project summary dashboards are fully configurable using Tcl based on report_configuration objects
  - With configurable reporting you can define exactly which reports are created for each run and each individual step of each run
  - Reports from new runs added automatically



- Report strategies
  - Ability to save to new strategies



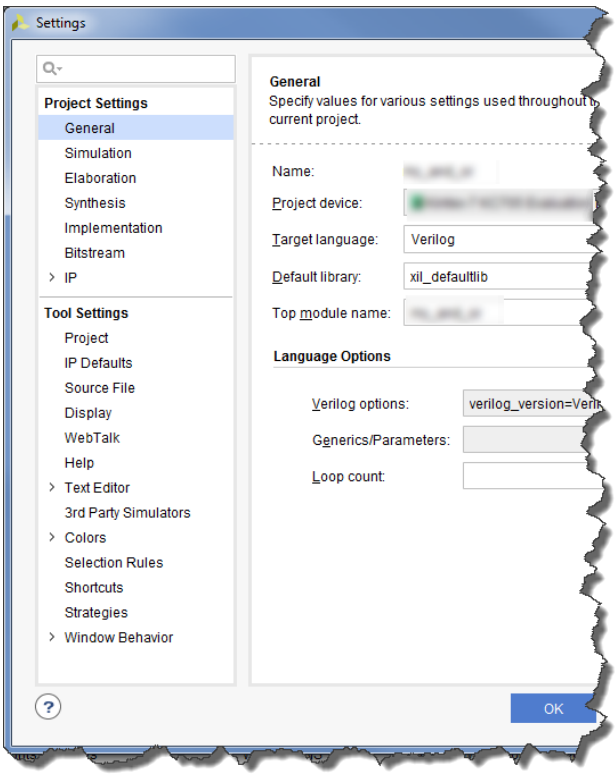
© Copyright 2019 Xilinx

Slide 3-16:

Project Settings

120269**slide

- Configure settings to meet design needs
- Categories to configure
  - General
  - Simulation
  - Elaboration
  - Synthesis
  - Implementation
  - Bitstream
  - IP



120269!*note

© Copyright 2019 Xilinx

General: Shows the project name, allows you to change the part, specify the top module name, and set the language options.

Simulation: Enables you to specify the target simulator (including third-party simulators). Displays the simulation set and simulation top module name.

Elaboration: Enables you to specify whether to load the Netlist model or Blackbox model.

Synthesis: Displays the default constraint set and provides options for choosing the synthesis strategy. The description of each strategy will be displayed at the bottom of the dialog box.

Implementation: Displays the default constraint set and provides options for choosing the implementation strategy. It also enables you to specify a placed and routed checkpoint to use as a reference for the next run.

Bitstream: Specifies the bitstream options.

IP: Allows you to specify settings for IP packager, the Core Container feature, etc. It also displays user-specified repositories.

© Copyright 2019 Xilinx

Slide 3-17:

Apply Your Knowledge

108493**slide

5) Which of the following is NOT true about the project mode in the Vivado Design Suite?

a) The Vivado tool manages the design data and status
b) The Vivado tool creates a project directory structure that the non-project batch flow does not
c) Source and results management is left to the designer
d) The Vivado IDE provides a single consistent scripting interface for the user

108493!*note

6) Which of the following is NOT true about the project mode in the Vivado Design Suite?

a) The Vivado tool manages the design data and status
b) The Vivado tool creates a project directory structure that the non-project batch flow does not
c) Source and results management is left to the designer
d) The Vivado IDE provides a single consistent scripting interface for the user

Slide 3-18:

Summary

111149**slide

© Copyright 2019 Xilinx

- In project mode, the Vivado Design Suite creates the directory structure for the project
- In project mode, the project status, sources, constraint files, and IP cores are managed automatically
- The Runs Manager handles the reading and writing of most files
  - Many files exist in pre-determined locations in the project hierarchy
- Multiple runs configuration is supported in project mode

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Vivado Design Suite I/O Pin Planning

Slide 4-1:

```
108107**slide
2019.1
108107!*note
```

This module describes the I/O Pin Planning feature within the Vivado® Design Suite.
15 minutes.

Slide 4-2:

Objectives

```
111573**slide
```

After completing this module, you will be able to:

- Create an I/O planning project in the Vivado IDE
- Use the Package Pins and I/O Ports windows in the Vivado IDE
- Place I/O ports interactively
- Describe the relationship between I/O banks and the logic in your design

Slide 4-3:

I/O Planning

```
107945**slide
```

- An FPGA design does not exist in a standalone environment
  - It must physically interface with other system components

© Copyright 2019 Xilinx

- The Vivado Design Suite provides simple methods for defining the FPGA-to-system interface
  - Use the I/O Planner to assign I/O pins in your design

- I/O planning can be done at different stages of the design flow
  - Elaborated, Synthesized, and Implemented design

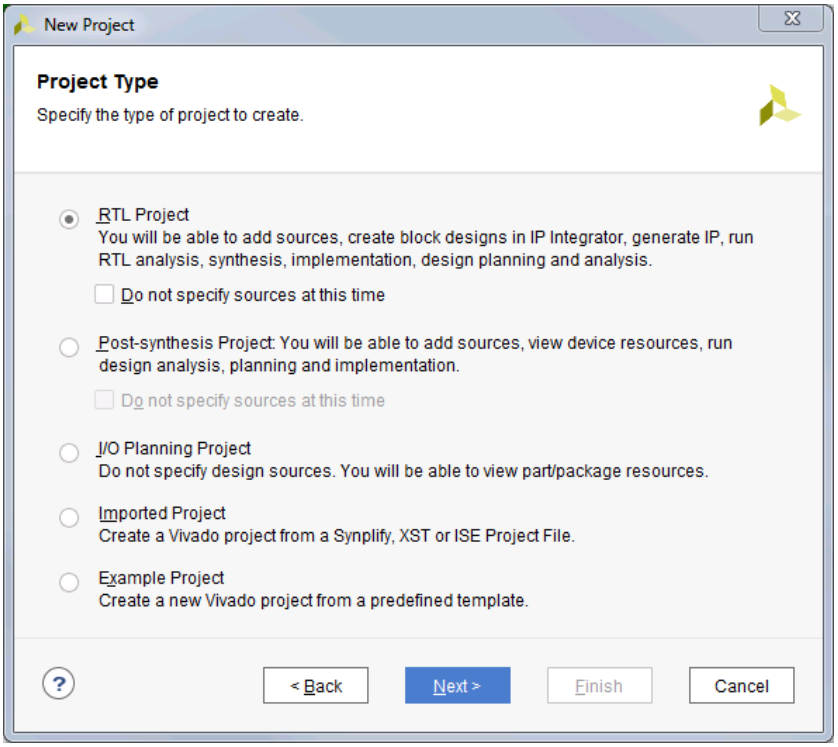
© Copyright 2019 Xilinx

Slide 4-4:

I/O Planning before RTL

107946**slide

- When creating a new project from the Getting Started page you have the option to create an I/O planning project without RTL
  - This allows you to test pin assignments
    - I/O banking rules
    - Avoid ground bounce
    - The I/O Planner performs error checking for your pinout
  - However, it is recommended that you have RTL associated
    - Error checking is better



107946!*note

© Copyright 2019 Xilinx

The New Project Wizard presents five choices of project types. Choosing an I/O planning project when there is not a netlist or RTL allows you to build a prototype pinout and test it. However, it is preferred that you have RTL associated with your project when you start to make your pin assignments. This is because the error checking is more thorough when you have RTL.

I/O pin planning projects always open in the I/O Planning view.

An XDC file is a user constraints file and is designed to store your pin assignments and pin attributes. It can also be used to store your design's timing constraints. The Vivado IDE software also allows you to modify your design's timing constraints.

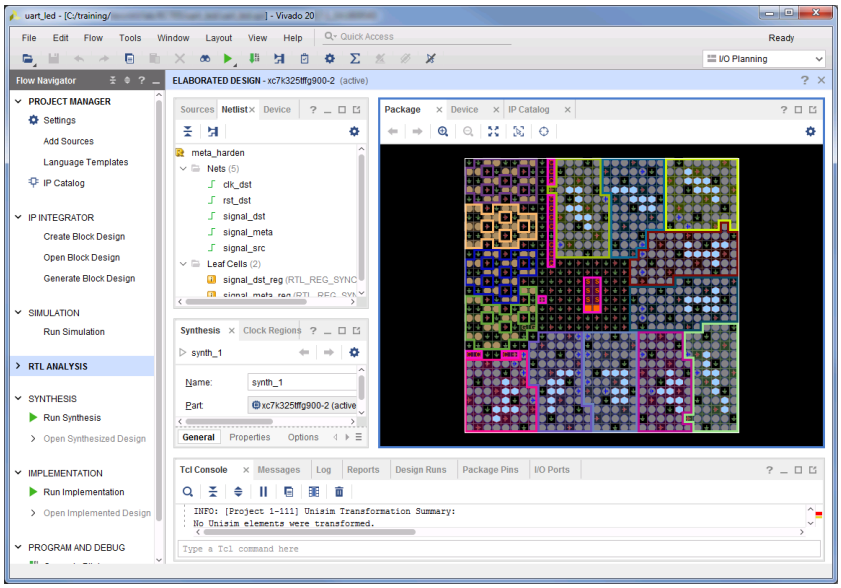
© Copyright 2019 Xilinx

Slide 4-5:

I/O Planning with RTL

107947**slide

- This flow is used if you already have RTL
  - Open the elaborated design by clicking **Open Elaborated Design**, which allows you to enter the I/O and Design Planners
  - Open the I/O Planner by selecting the **I/O Planning** view from the drop-down box from the Layout
  - The Vivado Design Suite allows you to check port assignments, I/O standards and other I/O planning details
    - Running the Design Rule Checks (DRCs) is critical



107947!*note

If you already have logic associated with your project, you will not be using the Create an I/O Planning Project option from the New Project Wizard. Instead, you will simply open an existing project and open the I/O Planner this way.

© Copyright 2019 Xilinx

Slide 4-6:

I/O Planning on Netlist and Implemented Design

129157**slide

- Netlist I/O planning flow is used if you have synthesized project or netlist
- At this stage, more details about the design will be available
  - Automatic I/O placement or interactive placement can be done
- With the implemented design, the final I/O pin planning needs to be validated

Slide 4-7:

Pin Assignment

107948**slide

- The I/O Planner performs error checking on the pin layout in the design
  - This requires rules-based I/O assignments
    - DRC provides guidance for pin assignments connecting to dedicated FPGA logic (microprocessor, MGT, or differential pairs, for example)
    - Noise analysis (to avoid ground bounce)
    - Verify I/O banking rules
  - Semi or fully automatic pin assignment capabilities
    - Xilinx recommends that you place timing-critical ports before allowing automatic pin assignment of the remaining pins
  - Supports grouping-related pins to simplify I/O interface management

© Copyright 2019 Xilinx

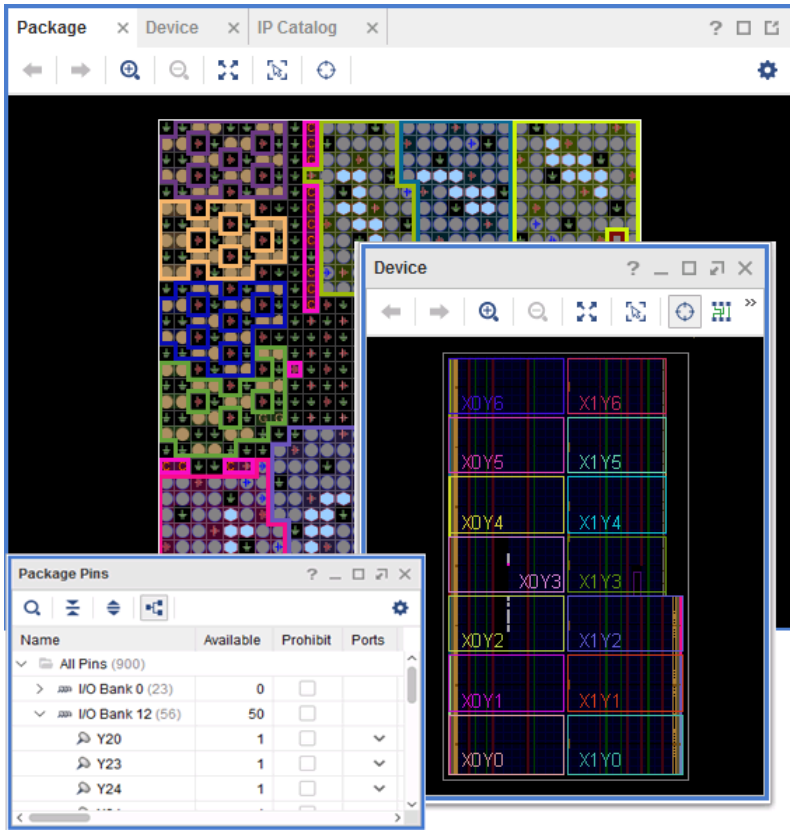
על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 4-8:

I/O Planning Layout View

107950**slide

- The I/O Planner layout lets you explore various device resources
- The I/O Planner allows you to view both the Device and Package views so that you can understand the I/O bank relationship with your logic
- Package and Device views
  - Graphically display package pins, die pads, and I/O banks
  - I/O ports can be assigned within either of these two views



- Package Pins view
  - Displays I/O package specifications and assignment status
    - Allows you to see the Trace delays, pin type, voltage standard, and differential pair partners
  - Can display pins as a group, in an I/O bank, or as a list
  - Shows pin and I/O bank placement status as you drag across the display icons

107950!*note

© Copyright 2019 Xilinx

As pins or banks are selected, the corresponding pins or banks become highlighted in the other views. This makes it easy to see that the pins assigned in each bank meet the I/O banking rules and are grouped appropriately. It also makes it easier to assign pins based on your logic placement, dedicated hardware placement, or clock signal distribution.

These views also allow you to manually assign objects, such as BUFRs, BUFGs, IODELAYCTRLs, or MGTs.

When pins are displayed as a list, they can be easily sorted. This allows you to see them organized by attributes, such as prohibited pins.

As you drag across the package view, yellow icons indicate assigned pins, gray icons indicate unassigned pins, and both displayed indicates partially assigned I/O banks.

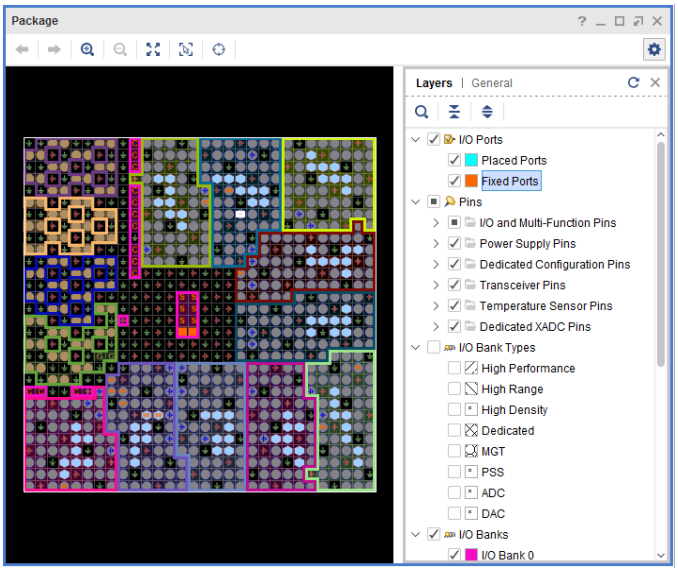
Slide 4-9:

Package View

107951**slide

© Copyright 2019 Xilinx

- Clock-capable pins are display as hexagon pins
- VCC and GND pins show as red and green square pins
- The colored areas between the pins display the I/O banks
- Clock-capable pins ( ), VCC ( ), GND ( ), no connection ( )



107951!*note

The Package View Layers option allows you to turn pins of a certain type on or off.

Moving and holding the cursor over the display will show the pin information.

Ports and I/O buffers can be dragged into the Package view or Device view for assignment.

Differential pairs can also be highlighted by clicking their icons in the vertical toolbar.

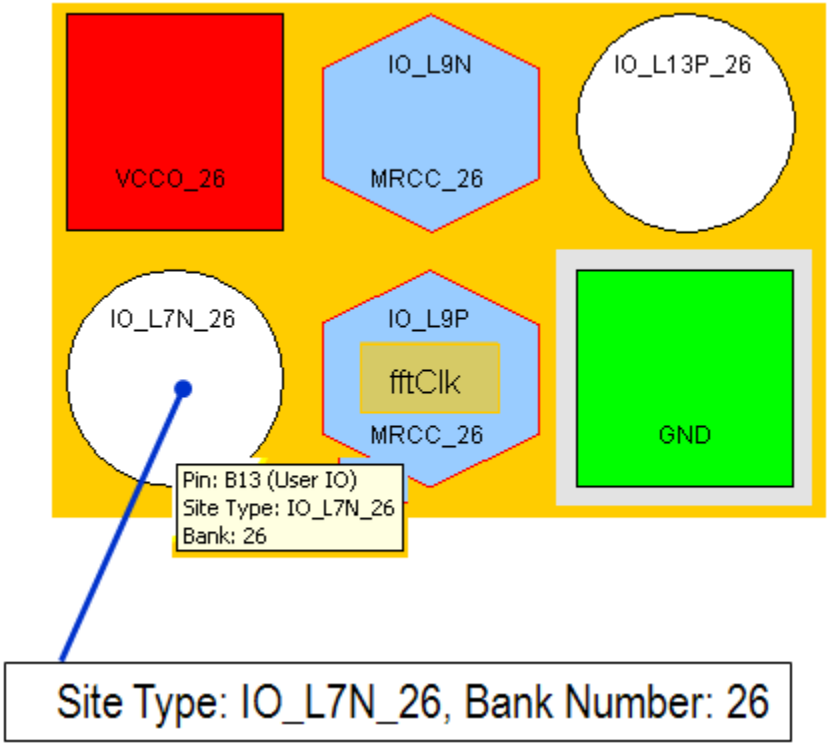
© Copyright 2019 Xilinx

Slide 4-10:

Using the Package Viewer

107952**slide

- The Package Pin viewer contains a very detailed list of the design package pins
  - Uses both colors and symbols
- Display includes
  - Pin name (signal name, if assigned)
  - Pin number (found by dragging over)
  - Site type (I/O)
  - Differential pair type (P or N)
  - Bank number (26)



107952!*note

© Copyright 2019 Xilinx

In this example, the rectangle inside of the dedicated clock pin (blue hexagon) represents a pin that is already assigned. Its type is defined as MRCC, which is a dedicated clock pin.

In this example, the pin B13 has not been assigned. Its site type is IO_L7N_26.

If a pin is assigned, the signal assigned shows up in a little yellow square (as seen in the image).

When the mouse rests over a site, the pin number and its information shows up in a little white square (as seen in the image).

The available differential pairs appear when the **Show Diff Pair** button is pressed on the vertical toolbar.

In this example, you can see that pin L9N is a differential pair with pin L9P.

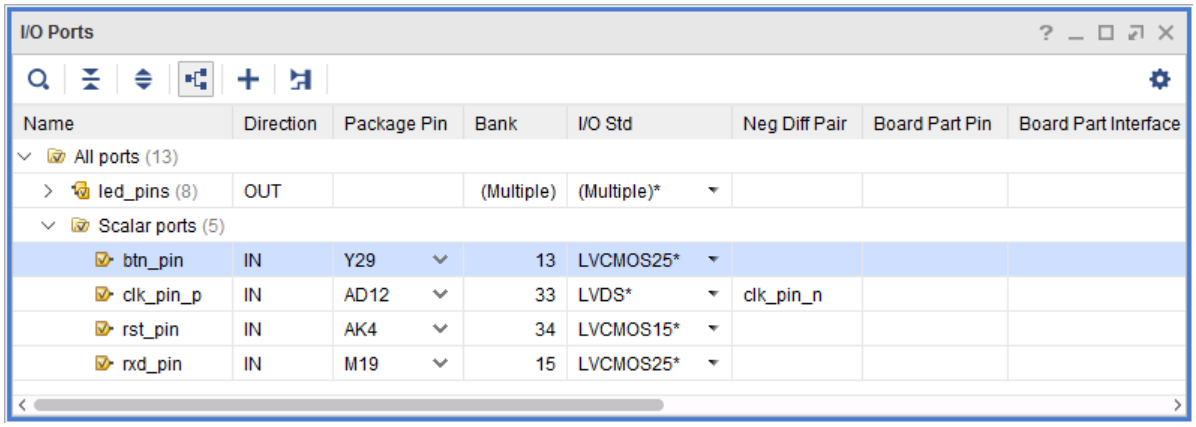
© Copyright 2019 Xilinx

Slide 4-11:

I/O Ports View

107953**slide

- Displays all I/O ports defined in the project
- Groups buses into expandable folders
- Can be displayed as groups of buses and interfaces or as a list
- Most I/O port assignment is initiated from this view
- Icons indicate I/O port direction and status



107953!*note

Yellow icons indicate placed ports and gray icons indicate unplaced ports. If both are displayed, this indicates partially placed buses or interfaces.

© Copyright 2019 Xilinx

Slide 4-12:

Interactively Placing I/O Ports

107954**slide

- Place by selecting the I/O ports, buses, or interfaces from **I/O Ports** window and right-clicking
  - **Place I/O Ports in an I/O Bank** – Places I/O ports in selected I/O banks
  - **Place I/O Ports in Area** – Draws a rectangle around the desire pins or pads
  - **Place I/O Ports Sequentially** – Selects a pin or pad for each I/O port individually
- The placement mode remains active until all selected I/O ports are placed or until the **ESC** key is pressed
- Assignment direction is dictated by the first selected I/O pin or I/O pad

107954!*note

Ports are displayed on the cursor. You can place an object by clicking the specific desired location. You cannot drag-and-drop pins. Dragging-and-dropping I/O ports will default to placing I/O ports in an available I/O bank.

The assignment order of the placement is determined by the order displayed in the I/O Ports view.

You can also sort to change bus direction.

© Copyright 2019 Xilinx

Slide 4-13:

DRCs and SSN Analysis

107955**slide

- Define I/Os early in the design process
  - Import RTL, CSV, XDC
- Interactively analyze and assign I/Os
  - DRC legal group drag and drop; assign clock logic
- Define alternate compatible parts
  - Ensure pin assignment works across multiple parts with a common package
- Robust DRCs
  - Find problems prior to implementation
- SSN analysis
  - Estimates switching noise levels caused by switching output ports

107955!*note

The I/O Planning view allows you to analyze device resources, including clock regions. You can create or import I/O ports for your design through a variety of formats. They can be grouped together into interfaces for easy placement, selection, or PCB symbol creation.

Spartan®-6, Virtex®-6, and 7 series FPGAs allow you to specify alternate parts in the same package. You can therefore ensure your design has a set of compatible pin assignments across multiple densities.

There are a variety of interactive I/O drag and drop placement capabilities. The interactive and batch DRCs check for common I/O planning mistakes. However, only the ISE software can ensure legal pin placement.

© Copyright 2019 Xilinx

6-series devices onwards support the use of SSN (Simultaneous Switching Noise) analysis to help you identify potential signal integrity issues and avoid building a ground bounce problem into your design. SSN analysis estimates the disruption that simultaneously switching outputs can cause on other output ports in an I/O bank. Virtex-5 FPGAs and older device families use a Weighted Average Simultaneously Switching Output (WASSO) analysis, which is slightly different.

The interactive and batch DRCs can only check the design based on the available information. Without a synthesized netlist, the tool cannot run all the checks. It does not know how the design is wired, so it is better to do pin planning with RTL or your design netlists, if they are available.

Slide 4-14:

Summary

107957**slide

- I/O planning can be done at different stages of the design process
- The I/O Planner view in the Vivado IDE provides an easy-to-use interface for assigning pin locations
- The Package and Device views provide a graphical display of package pins, die pads, and I/O banks
- The I/O Ports view displays all I/O ports defined in the project and their properties
- I/O pin assignments can be verified by running a DRC report

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Vivado IP Flow

Slide 5-1:

```
108926**slide

2019.1
108926!*note
```

This module introduces IP flows in the Vivado® Design Suite. 10 minutes.

Slide 5-2:

Objectives

```
108927**slide
```

After completing this module, you will be able to:

- Use the Managed IP flow in the Vivado IDE
- Access the IP catalog from the project to customize and add IP to the design
- Describe the usage of IP output files
- Describe synthesis options for IP

© Copyright 2019 Xilinx

Slide 5-3:

IP Operations

108878**slide

- The Vivado Design Suite has a unified catalog with all IP in one place
- Search for the desired IP in the IP catalog
- Customize the IP by double-clicking the IP to open the Customization page
- Generate output products
- Synthesize the IP
- Instantiate the IP in the design

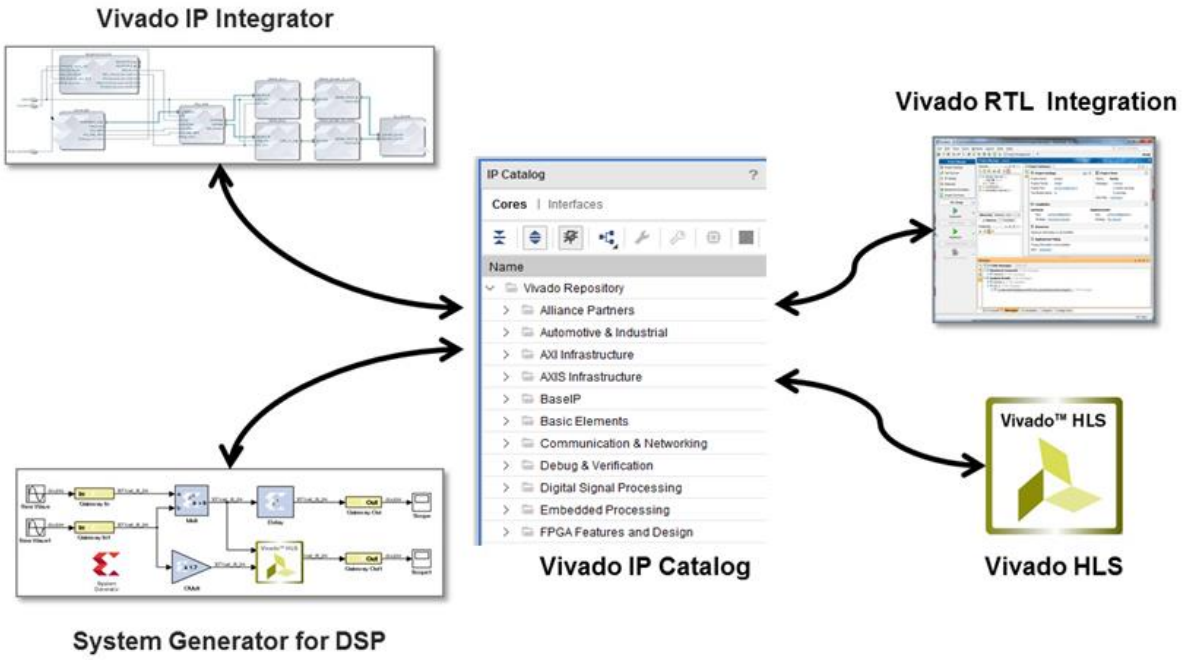
Slide 5-4:

Vivado Design Suite: IP-Centric Design Tool

108877**slide

© Copyright 2019 Xilinx

- The Vivado Design Suite provides direct flows from all four design environments into and out of the Vivado IP catalog



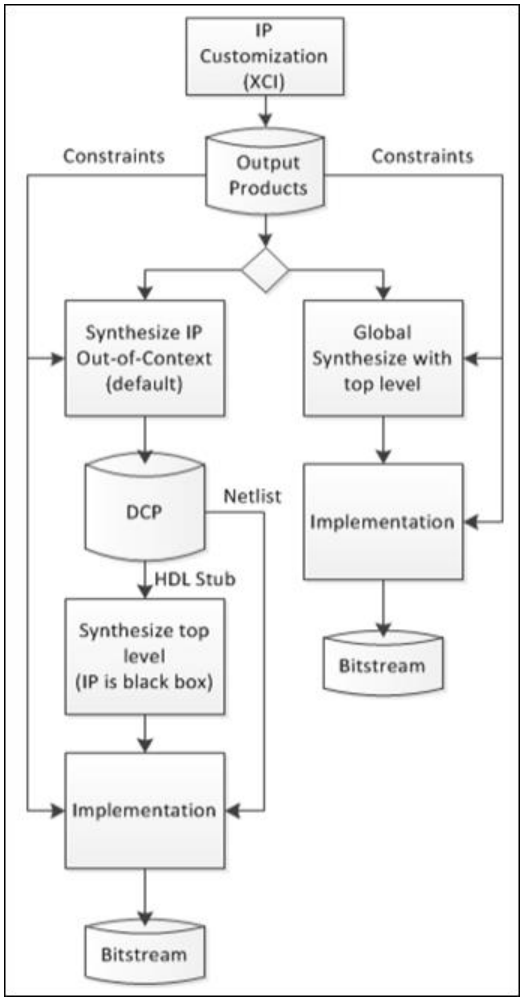
© Copyright 2019 Xilinx

Slide 5-5:

IP Synthesis Options

108879**slide

- Two ways to reference IP in a design
  - Out of context (OOC)-based (default flow)
    - By default, the OOC flow is used for all Vivado IP delivered via the IP catalog
      - User can override the default settings
    - DCP file read for IP as a netlist, avoiding re-synthesis
    - Can validate timing for IP at the synthesis level
  - Global synthesis
    - IP RTL synthesized with other RTL during project synthesis
    - Unlike the OOC flow, RTL changes (other than IP) lead to resynthesized IP
    - If IP is instantiated multiple times, synthesis will run for all instantiations separately



108879!*note

© Copyright 2019 Xilinx

Vivado IP generation creates encrypted Verilog that is used for synthesis and simulation. Only Vivado synthesis is supported for generating the netlist for an IP.

When using third-party synthesis tools for project synthesis, generate the IP in the Vivado Design Suite with DCP output. Instantiate the stub RTL in the project RTL and synthesize the design (the IP will be a black box in the synthesis netlist). Create a netlist project in the Vivado Design Suite, include the third-party synthesis netlist and DCP files for the IP, and then run the implementation.

© Copyright 2019 Xilinx

Slide 5-6:

Output Products Generated for IP

108880**slide

- By default Xilinx IP will generate
  - A design checkpoint (.dcp)
  - An instantiation template
  - Synthesis files
  - Simulation files

- An IP can deliver a number of XDC files (prepended with IP name)

108880!*note

- *<ip_name>.xci*, *.xml*: These files contain the IP configuration. The XCI file is a Xilinx Core Instance file that contains the IP customization information. You can generate the output products from this file. If an upgrade path exists for the IP in the catalog, you can upgrade from this file to the latest version.
- *synth*: RTL files that are synthesized to generate the IP. The files here contain IP elements present in the *<ip_element 1>*.. *<ip_element n>* directories.
- *simulation*: Simulation-specific files, such as testbenches. Usually all files in *synth* are also used for simulation. If any of the *synth* files needs a variant for simulation, it will be in this directory.
- *<ip_element 1>*.. *<ip_element n>*: These sub-directories correspond to sub-blocks that make the IP. The name and number of these directories varies between IPs. Under each directory there will be encrypted HDL files. These can be synthesized by Vivado synthesis.
- *example_design*: When IP contains an example design (such as TEMAC or GT), the RTL files that make up the example design are here. This example design instantiates the IP.
- *<ip_name>_example*: A Vivado Design Suite project is created in this sub-directory with the example design and testbench for simulating the example design. The *<ip_name>_example.xpr* file under this directory can be opened in the Vivado Design Suite to open this example design in project mode.

© Copyright 2019 Xilinx

- *<ip_name>_ex.tcl*: This Tcl script builds the *<ip_name>_example* project.
- *<ip_name>.dcp*: This is the DCP file for the IP.
- *<ip_name>.vho*, *.veo*: Instantiation template.
- *<ip_name>_stub.v*: Blank Verilog module for black-box synthesis with third-party synthesis tools.
- *<ip_name>_funcsim.vhdl*, *.v*: Functional simulation files generated from the IP netlist. (Output by the `write_vhdl` / `write_verilog` Tcl commands.)

File Name	Processing Order	Used During	Description
.xdc	EARLY	OOC/impl	Clock creation commands as well as all commands without outside clock dependencies.
_clocks.xdc	LATE	OOC/impl	All commands with dependencies to outside clock definitions.
_ooc.xdc	EARLY/OOC	OOC*	Default top-level clock definitions. May require user editing.
_board.xdc	EARLY	OOC/impl	For IP that support development board physical constraint generation.

- **_ooc.xdc*: Used only for IP OOC synthesis/implementation runs.

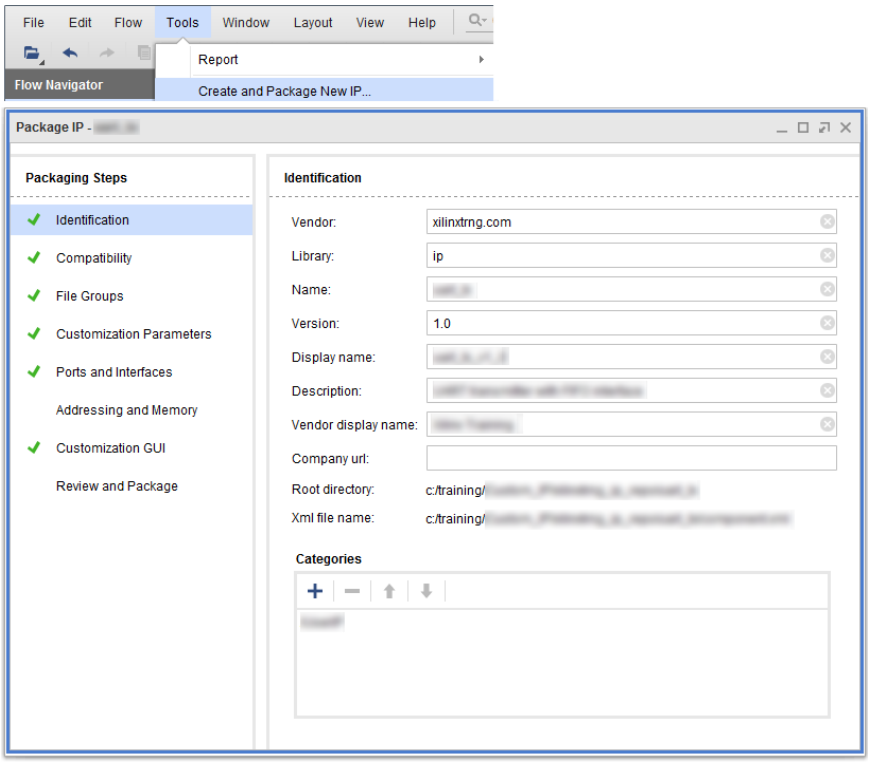
© Copyright 2019 Xilinx

Slide 5-7:

IP Packager

111567**slide

- The IP Packager is based on the IP_XACT (IEEE-1685) standard
- Converts your design into reusable IP
- Create a Vivado IDE user design that will be turned into an IP
- Add this IP to the Vivado IP catalog so others can use it for design work



- Adding packaged IP to the IP catalog
  - Unzip the IP to a local directory
  - Add the directory to the IP repository of the IP catalog

111567!*note

The Vivado Design Suite provides an IP Integrator tool that lets you create complex subsystem designs by instantiating and interconnecting IP cores from the IP catalog.

© Copyright 2019 Xilinx

Slide 5-8:

IP Catalog

144674**slide

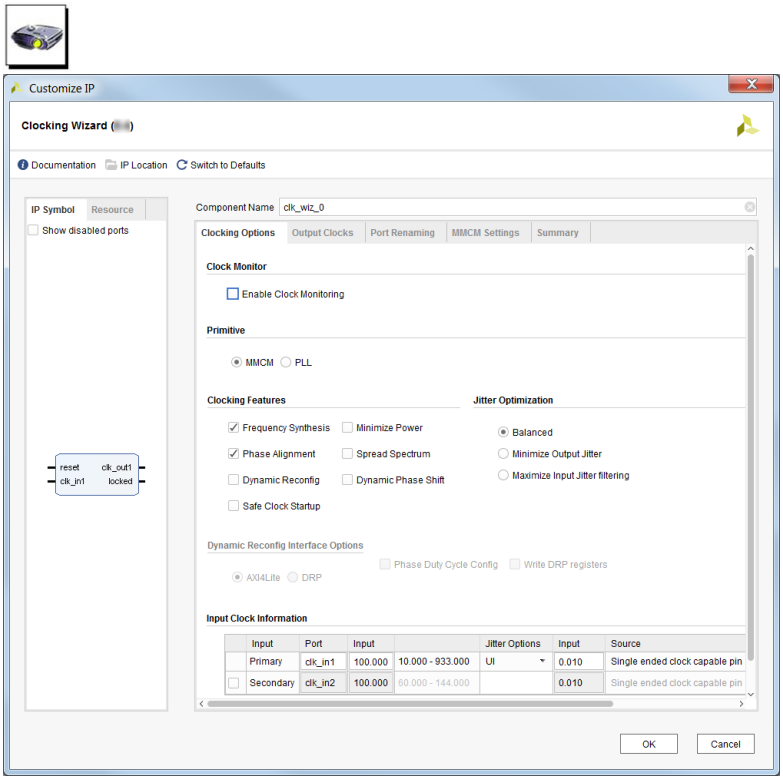
- Xilinx maintains a rich library of intellectual property (IP) that helps to get to market faster
- Consistent, easy access to Xilinx IP, including building blocks, wizards, connectivity, DSP, embedded, AXI infrastructure, and video IP from a single common repository regardless of the end application being developed
- On-demand delivery of IP output products, such as instantiation templates and simulation models
- Access to IP customization and generation using the Vivado IDE
- Access to version history details as recorded in the Change Log

© Copyright 2019 Xilinx

Slide 5-9:

Adding IP to the Design: Clocking Wizard

108873**slide



- You can choose the features or specific primitive to include

- The UltraScale™ device default is Auto
  - The software makes a decision based on required feature set
  - If possible, will use PLL

108873!*note

© Copyright 2019 Xilinx

Xilinx maintains a rich library of intellectual property (IP).

The Clocking Wizard is an interactive GUI very similar to the clocking wizards from previous generations of the Vivado Design Suite. The Clocking Wizard allows you to choose the specific primitive you require or define which features and capabilities you want and allows the software to make the choice.

Because memory interface planning will be done at the outset of the design flow, the availability of PLLs will already be known when the clock planning is done. For this reason, if the PLL is available, the Clocking Wizard will use it if the feature set allows.

© Copyright 2019 Xilinx

Slide 5-10:

Summary

108882**slide

- IP flow
  - The Vivado Design Suite developed from scratch to support an IP-centric design flow
  - The IP catalog consists of various categories of IP
  - The IP auto update is recommended but not mandatory

- The Vivado IP integrator
  - Is platform aware with real-time DRCs
  - Can build and reuse IP subsystems
  - Supports integrated debugging
  - Supports GUI and Tcl interfaces
  - Raises the level of design abstraction and increases design productivity

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Clocking Resources

Slide 6-1:

13627**slide

2014.1
13627!*note

This module describes the clock networks and the Clock Management Tile (CMT) capabilities in the 7 series FPGAs. 90 minutes.

Slide 6-2:

Objectives

13628**slide

After completing this module, you will be able to:

- Detail the clocking resources available in the 7 series FPGAs
- Specify the resources available in the Clock Management Tile (CMT)
- Use the above resources in your design

Slide 6-3:

Overview

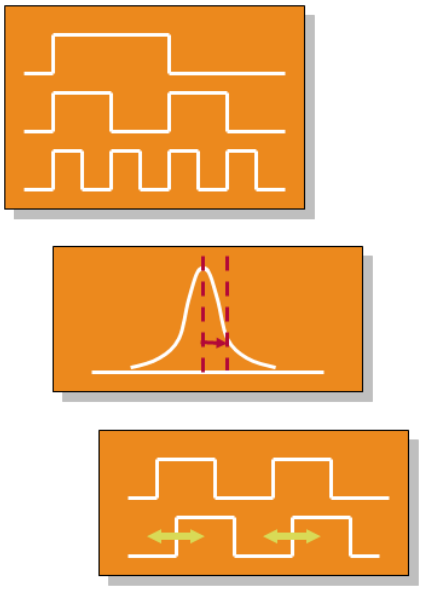
13629**slide

- **Overview**
- Clock Networks and Buffers
- Clock Management Tile
- Usage Models
- Using Clock Resources
- Summary

Slide 6-4:

High-Performance Clocking

13630**slide



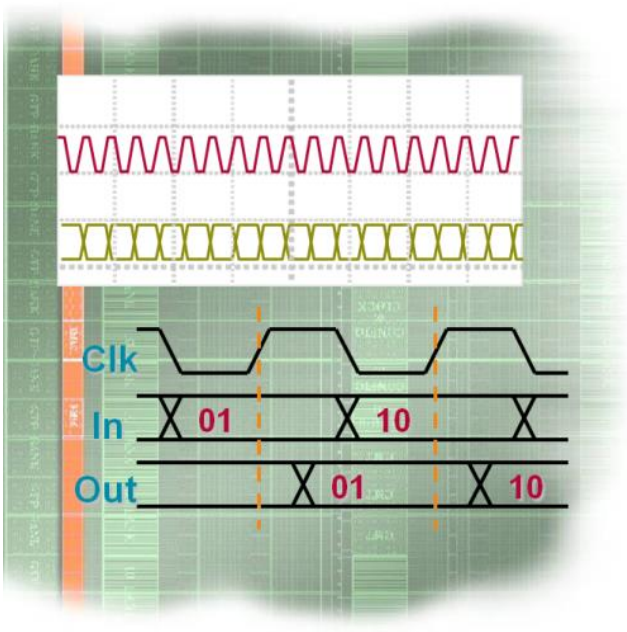
- Modern applications have complex clocking requirements
  - Extremely high-performance clock signals
  - Support for multiple frequency domains across a wide frequency range
  - De-skewing of clocks relative to one another
  - Low jitter and precise duty cycle to maintain the widest possible data valid window
  - Lowest possible system power
- Xilinx FPGAs have a rich mixture of clocking resources to accommodate these requirements
  - The perfect balance of resources at the right cost

Slide 6-5:

Clock Management

13631**slide

- Systems usually require multiple clock frequencies from the same source
  - Minimizing the number of oscillators lowers system cost
- External clock sources can often be noisy
  - Filtering jitter cleans up clocks widening the data valid window
- Many circuits need to be clocked at the same time to ensure correct operation
  - De-skewing and aligning clocks eliminates hold-time issues and race conditions

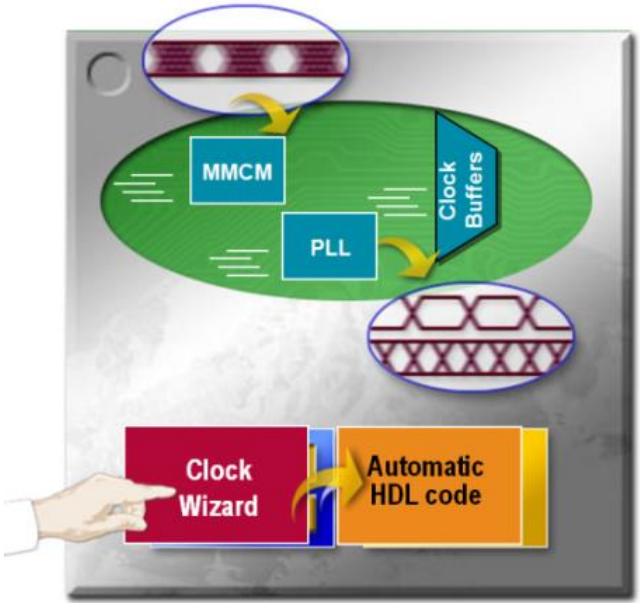


Slide 6-6:

7 Series FPGAs Deliver Powerful Clock Management

13632**slide

- Global clock buffers
  - High fanout clock distribution buffer
- Low-skew clock distribution
  - Regional clock routing
- Clock regions
  - Each clock region is 50 CLBs high and spans half the device



- Clock management tile (CMT)
  - One Mixed-Mode Clock Managers (MMCMs) and one Phase Locked Loop (PLL) in each Clock Management (CMT)
  - Performs frequency synthesis, clock de-skew, and jitter-filtering
  - High input frequency range
- Simple design creation through the Clocking Wizard

13632!*note

Note that the clock region now contains 50 CLB rows, not 40 as in the Virtex-6 device family.

Slide 6-7:

Clock Networks and Buffers

13633**slide

- Overview
- **Clock Networks and Buffers**
- Clock Management Tile
- Usage Models
- Using Clock Resources
- Summary

Slide 6-8:

Clock-Capable Inputs (CCIO)

13634**slide

- All synchronous designs need at least one external clock reference
  - Many designs require several clock sources
- These sources need to be brought into the FPGA and connected to the internal FPGA clock resources
- Every 7 series FPGA has four clock-capable inputs in each bank
  - These inputs are regular I/O pins with dedicated connections to internal clock resources
    - When not used as clock inputs, they can be used as a regular I/O pin
  - Each clock input can be used as a single-ended clock input, or can be paired with an adjacent pin to form a differential clock input
    - Each bank can therefore have four single-ended or four differential clock inputs
  - Two of the four are Multi-Region Clock Capable (MRCC) and the other two are Single Region Clock Capable (SRCC)

13634!*note

Clock-capable pins will have the identifier *CC* in their pin name.

An IBUFG or IBUFGDS primitive can be used in your HDL code to indicate that an input must be mapped to a clock-capable pin.

Slide 6-9:

Clock Networks

13635**slide

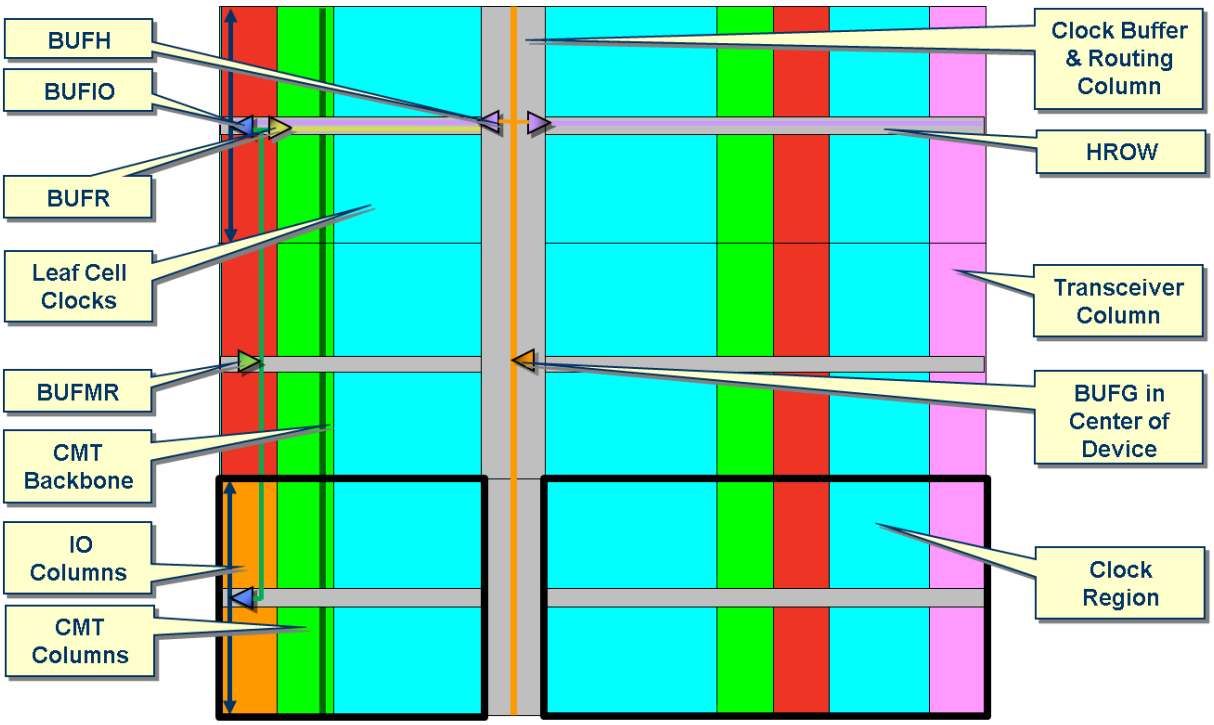
- An FPGA is a regular array of resources
  - Many of these resources require clocks for synchronous operations
    - Slice flip-flops, input / output flip-flops, block RAMs, DSP slices
  - In order to implement synchronous designs, clocks must be distributed to these clocked elements

- For synchronous operation, clocks must arrive at the clocked elements with
  - Extremely low clock skew: Ensures minimal internal hold-time issues
  - Low clock jitter: Allows for highest performance
  - Duty cycle preservation: Important for Double Data Rate (DDR) applications
  - Low insertion latency: Important for synchronous input and output interfaces

Slide 6-10:

Die View

13636**slide



13636!*note

The Clock Buffer and Routing Column contains only the global clock buffers and routing.

The MMCMs and PLLs are located in the CMT columns that are immediately adjacent to the IO columns.

The global clock buffers (BUFG) are in the middle of the chip. These drive the vertical spines of the global clock network.

The horizontal spines of the global clock network run through the center of each clock region. These horizontal spines are driven by BUFH buffers.

There are also regional clock routing resources driven by BUFRs.

Clocks are driven up and down from the center horizontal row (HROW) of each clock region.

The BUFIO are placed within the I/O column, and drive the I/O clock network in that bank.

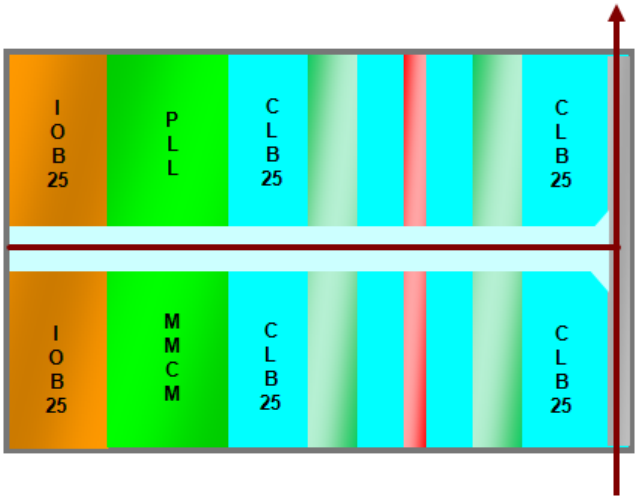
The BUFMR are dedicated buffers that allow clock inputs to drive the BUFIOs and BUFRs of adjacent regions.

Slide 6-11:

7 Series FPGA Clock Regions

13637**slide

- Larger clock region than previous families
  - 50 CLBs high, 50 I/Os high
  - Same size as I/O bank
  - Half width of device
  - 2–24 regions per device for 7-Series



- Resources per clock region
  - 12 global clock networks
    - Driven by BUFH
  - 4 regional clock networks
    - Driven by BUFR
  - 4 I/O clock networks
    - Driven by BUFIO

13637!*note

This is a close-up view of the 7 series FPGA clock region. Each clock region is 50 CLBs high and half the width of the device.

As opposed to earlier technologies, there is one IOB column per clock region. This simplifies the clock networks in the device.

Each clock region contains 12 BUFH buffers that drive the global clock network.

Each clock region has four regional clock networks, driven by the BUFRs within the region.

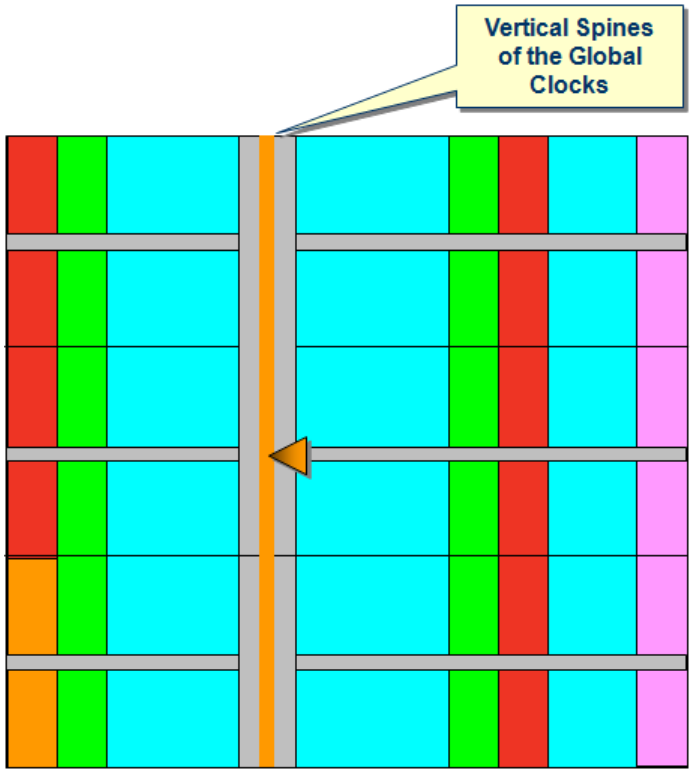
Each clock region has four I/O clock networks per I/O column.

Slide 6-12:

Global Clocking – Vertical Spines

13638**slide

- Global clock networks allow clocks to be distributed to potentially every clocked element on the die
- Global clocks are driven by BUFGCTRL located in the middle of the die
- Each BUFGCTRL drives a vertical spine in the center of the die that runs from the bottom to the top of the die
- All 7 series devices have 32 vertical spines

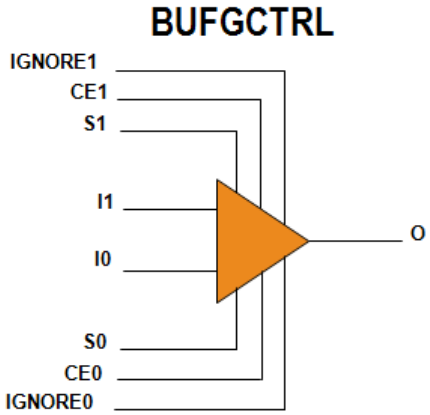


Slide 6-13:

Global Clock Buffer (BUFGCTRL)

13639**slide

- 32 BUFGCTRLs (or BUFG) reside in the center of the device



- BUFGCTRLs can be driven by
  - Clock-capable I/O (CCIO) in the same half
  - CMT outputs in the same half
  - Gigabit transceiver clocks in the same half
  - Other BUFG, interconnect, or BUFR
- BUFGCTRL outputs drive the vertical global clock spine
- BUFGCTRL component implements
  - Simple clock buffer (BUFG)
  - Clock buffer with clock switching (BUFGMUX or BUFGMUX_CTRL)
  - Clock buffer with clock enable (BUFGCE)

13639!*note

The BUFGCTRLs straddle a clock region boundary near the center line of the device; half the BUFGCTRLs are in the top half of the die and the others are in the bottom half. This boundary may not be exactly in the middle of the die, depending on the device. Refer to the floorplan specific to your device.

Each BUFGCTRL can be driven by the CCIO, CMT (PLL or MMCM), or gigabit transceivers in the same half of the device. The BUFGCTRL in the top half of the die can only be driven by the resources in the top half of the die.

In the smaller Artix devices, which do not have a CMT column on one side, the CCIOs on that side cannot access the BUFCTRLs.

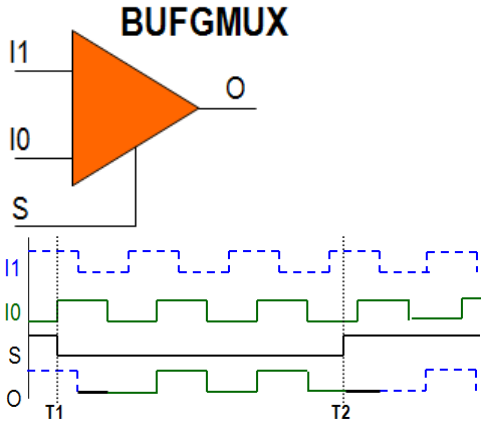
As with all clocking resources, buffers driven by local interconnect have no known phase relationship with their source.

Slide 6-14:

BUFGCTRL for Clock Multiplexing

13640**slide

- A BUFGCTRL can be configured as a clock multiplexer
- Allows for glitchless switching between two clock sources
  - A select signal is generated by the user's design, which selects between two different input clocks
  - The selected clock is driven onto the clock network



- The BUFGMUX primitive can be used to configure a BUFGCTRL as a global clock buffer with clock multiplexing
- Two flavors
  - BUFGMUX: S input must be synchronous to current clock
  - BUFGMUX_CTRL: S input can change any time

13640!*note

The BUFG multiplexer utilizes the S port to implement a selection. The S signal must meet setup time prior to the clock edge of the currently selected clock. Violating the setup time can result in a glitch (check your FPGA user guide for more information).

Assuming setup times are met, the functionality of the BUFGMUX macro is as follows:

- At time T1, S (Select) transitions from high to low.
- Because I1 is currently high, the multiplexer waits for I1 to transition high to low.
- After I1 is low, the multiplexer waits for I0 to transition high to low.
- When I0 transitions from high to low, the outputs are switched to I0.

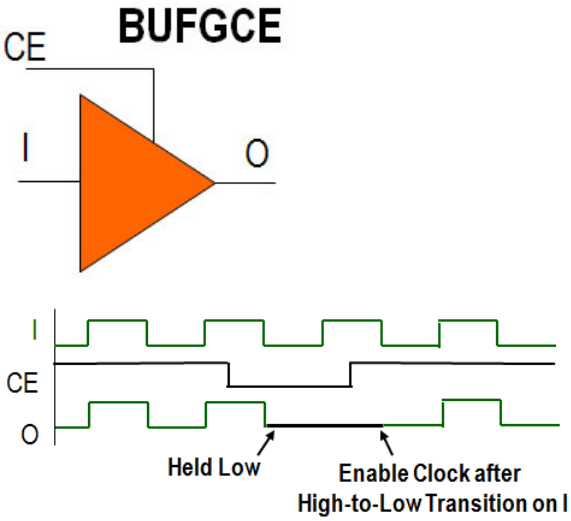
The BUFGMUX_CTRL is similar, but there is no setup / hold requirement on S to guarantee glitchless switching. This primitive replaces the BUFGMUX_VIRTEX4 of previous technologies.

Slide 6-15:

BUFGCTRL for Gated Clock (BUFGCE)

13641**slide

- A BUFGCTRL can be configured as a clock buffer with clock enable functionality
- Allows the clock to turn on and off dynamically
  - Enables dynamic clock gating for power savings and the generation of decimated clocks
  - The Clock Enable (CE) signal is generated by the user's design
  - The gating is glitch and runt free if the CE is generated synchronously
  - The clock is still driven onto the clock network



- The BUFGCE is a global buffer with a built-in CE

13641!*note

Disabling the clock while I is high will not disable the clock until I transitions from high to low; likewise, enabling the clock while I is high will not enable the clock until after I transitions from high to low.

The BUFGCE must be instantiated.

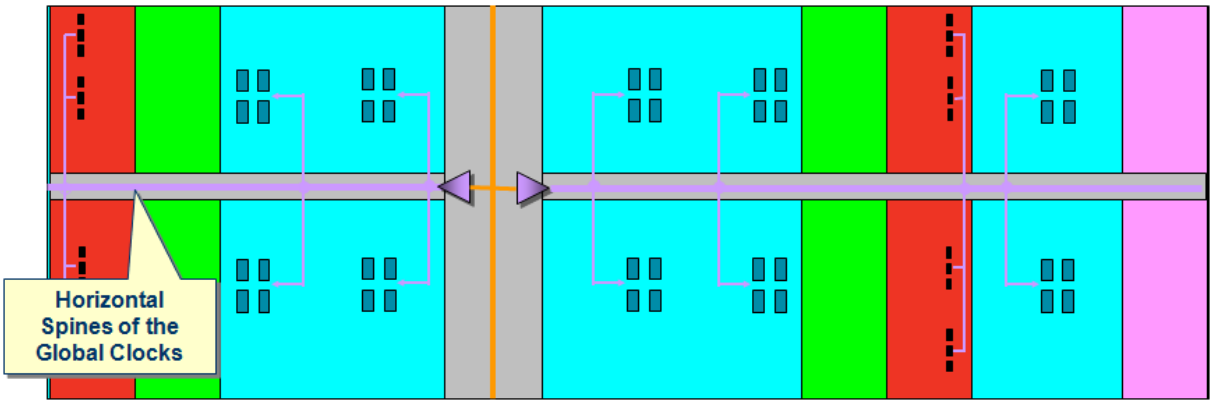
A BUFGCTRL can also be used for other functions, such as an asynchronous multiplexer or a clock multiplexer with clock enable capability. These can be accomplished by instantiating the full BUFGCTRL and managing the control inputs in your design.

Slide 6-16:

Global Clocking – Horizontal Spines

13642**slide

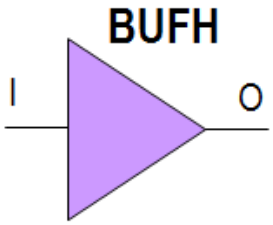
- Each clock region has 12 horizontal spines for carrying global clocks
  - Clock regions on left and right have 12 independent horizontal spines
- These horizontal spines can drive all clocked resources within the region
  - Slice flip-flops, distributed RAMs, SRLs, block RAMs, DSP slices, IOB flip-flops, CLKDIV of ISERDES and OSERDES …
- Each horizontal spine is driven by a BUFH



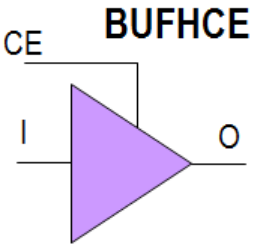
Slide 6-17:

Horizontal Spine Clock Buffer (BUFH)

13643**slide



- Each of the 12 horizontal spines in a clock region is driven by a BUFH
- Driven by
  - CMTs in the same region
  - BUFGCTRL via vertical clock spine
  - Clock-capable inputs in the same horizontal row
    - Can be driven from bank on the left or right
  - Interconnect



- Can be a simple clock buffer (BUFH), or a clock buffer with clock enable (BUFHCE)
  - BUFHCE is glitchless when CE_TYPE set to *SYNC*
- When a global clock (from a BUFGCTRL) is required in a clock region, a BUFH is automatically inferred by the tools

13643!*note

BUFH drives a horizontal clock row into each clock region. Each clock region has 12 BUFHs.

There is also a BUFHCE primitive with a clock enable input, which is helpful for turning off a local clock into a region to help minimize power consumption.

If you instantiate a BUFH or BUFHCE in your design, an area constraint may be required to keep all of the clock loads confined to a single clock region.

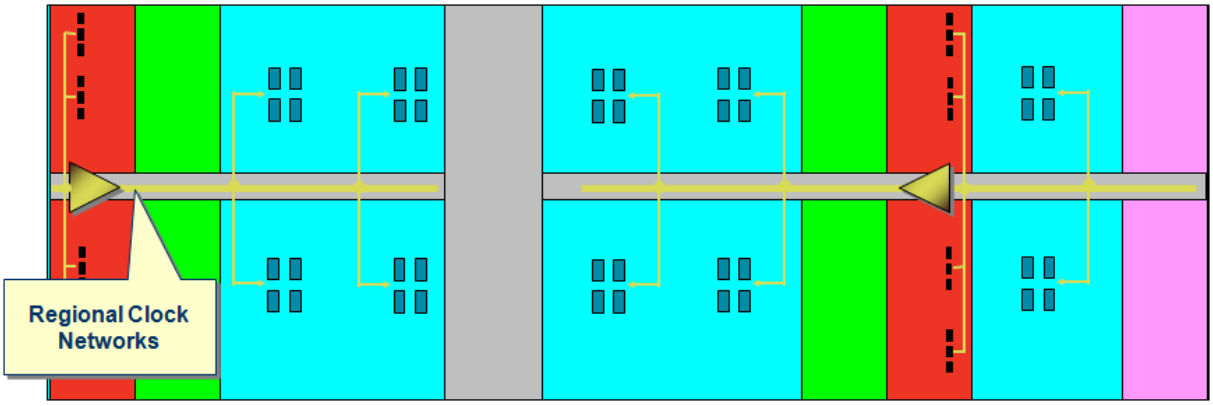
In the smaller Artix devices, which do not have a CMT column on one side, the CCIOs on that side cannot access the BUFHs.

Slide 6-18:

Regional Clock Networks

13644**slide

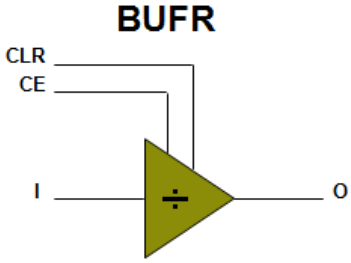
- Each clock region has four regional clock networks
  - Regional clocks in the left and right half are separate
- These networks are in addition to the 12 horizontal spines of the global clock
- These regional networks can drive all clocked resources within the region
- Each horizontal spine is driven by a BUFR



Slide 6-19:

Regional Clock Buffer (BUFR)

13645**slide



- Four BUFR in each clock region – one for each regional clock network
- Driven by
  - Clock-capable inputs in the region
  - Interconnect
  - Gigabit transceivers in the same region
  - MMCM high-performance clocks in the same region
  - BUFMR in the same region or region above or below
- The output of the BUFR can be a divided version of the input clock. Dividers of 1 to 8 are supported
  - CLR and CE can be used to control which input clock rising edge generates the output clock rising edge

13645!*note

The relationship between BUFRs and regional clock networks in 7 series FPGAs is simpler than in previous technologies. Because there are four regional clock networks and four BUFRs per region, the relationship is one to one.

BUFRs cannot directly drive regional clock networks in adjacent regions. This is accomplished using the new BUFMR. The BUFMR has features for synchronizing multiple BUFRs.

Gigabit transceiver clocks can only drive BUFRs if the transceivers are in a separate column. If the transceiver replaces a region in the I/O and CMT column, no BUFR exists in that region.

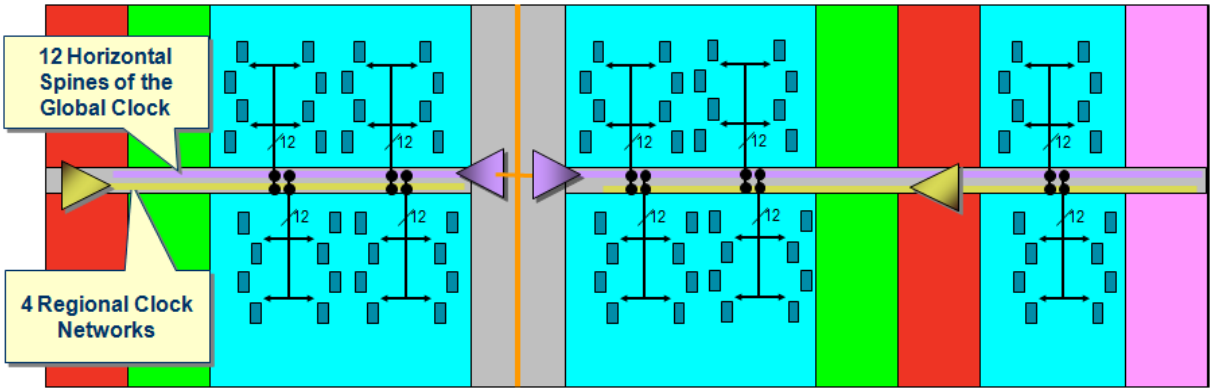
Slide 6-20:

Clocking CLB Resources

13646**slide

- In every region there are 12 horizontal spines of the global clock network and 4 regional clocks
  - All 16 clocks can clock resources within the region
  - Only 12 of the 16 can enter the double column of CLBs in the top and bottom half of the clock region

- Thus, any 12 of the 16 clocks are available for the 50 CLBs in a pair of columns above the HROW and 12 of the 16 clocks are available for the 50 CLBs in the pair of columns below



13646!*note

The CLBs are arranged as symmetrical pairs of columns. Each column is 50 CLBs high, with 25 above the HROW and 25 below the HROW.

The two adjacent half columns share 12 routing tracks from the HROW clocking resources. Thus, for the 50 CLBs that are in two adjacent columns above the HROW, only 12 of the 16 clocks available in the HROW can be used. Similarly, an independent set of 12 of the 16 clocks can be used for the 50 CLBs in the two half columns below the HROW.

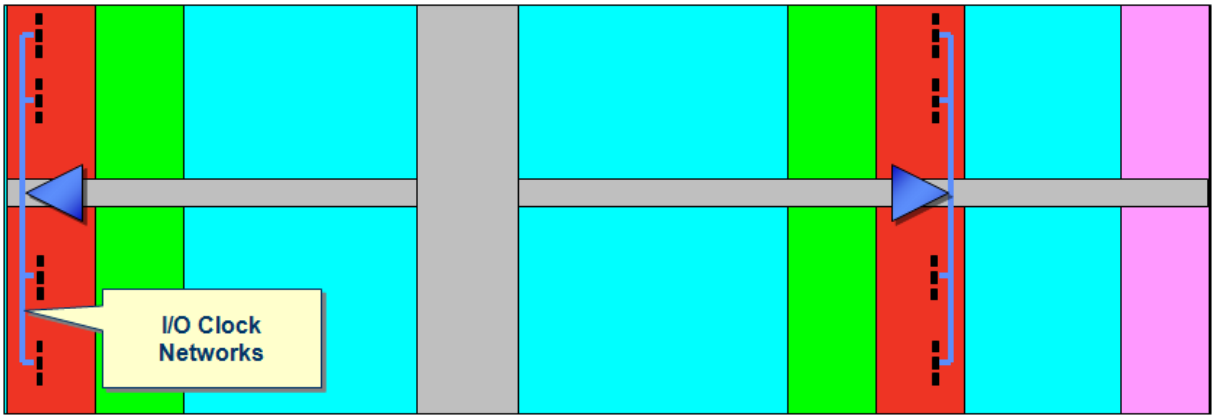
A similar restriction exists for the IOBs in the I/O column. Any 6 of the 12 clocks on the horizontal spines of the global clock can be used for the 25 IOBs above the HROW, and any 6 of the 12 can be used below. In addition to the 6 global clocks, all 4 regional clocks can be used by all IOBs (as can all 4 of the I/O clocks, shown later).

Slide 6-21:

I/O Clock Networks

13647**slide

- Each clock region has four I/O clock networks
- These networks can drive only the clock ports of the ILOGIC / OLOGIC resources and the high-speed clock ports (CLK) of the ISERDES / OSERDES
- Each I/O clock network is driven by a BUFIO
- Highest quality clock available



13647!*note

The I/O clock networks can drive the clock pins of the IOB resources *only*:

- DDR and SDR flip-flops in the ILOGIC / OLOGIC block.
- High-speed clock port (CLK) of the ISERDES and OSERDES resources.

Because of the low fanout, they have extremely low skew and a very short insertion delay (ideal for source-synchronous interface applications).

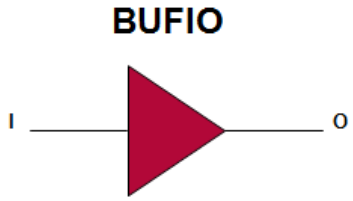
These networks add the least jitter of all clock networks, and have run at the highest speeds (up to 1066 MHz).

Slide 6-22:

I/O Clock Buffer (BUFIO)

13648**slide

- The I/O clock networks are driven by the BUFIO



- The BUFIO is a simple clock buffer with no additional functionality
- Driven by
  - Clock-capable inputs in the same I/O bank
  - MMCM high-performance clocks in the same region
    - These are immediately adjacent to the I/O bank
  - The BUFMR in the same region or region above or below
  - Interconnect

13648!*note

When driven by interconnect, the phase relationship of the resulting clock is unknown.

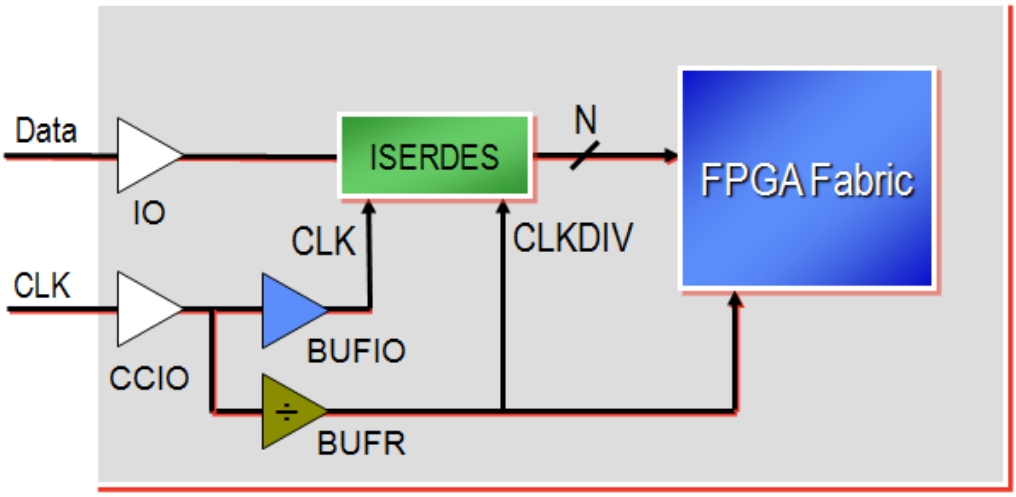
Slide 6-23:

Source-Synchronous Interfaces

13649**slide

- I/O and regional clock networks combined with ISERDES / OSERDES provide powerful tools for creating source synchronous interfaces

- BUFR is set to ÷ N if interface is SDR, or ÷ (N / 2) if DDR
  - N can be 2 to 8 in SDR, and 2 to 14 in DDR



13649!*note

The output of the BUFIO and BUFR are guaranteed to be in phase, allowing proper clock crossing between CLK and CLKDIV in ISERDES / OSERDES.

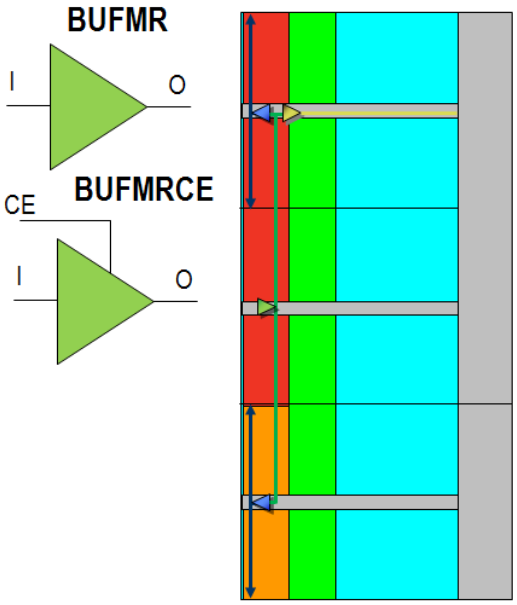
For high-speed output interfaces, the BUFIO and BUFR can be driven by a high-speed clock generated by a high-performance clock from the MMCM in the same region.

Slide 6-24:

Multi-Region Clock Buffer (BUFMR)

13650**slide

- Each bank has two BUFMRs
- Allows a single clock source to be driven to the BUFIO and / or BUFR in the region above and below
  - Can be used to create an interface with more than 50pins (maximum 150)
- Driven by
  - Multi-region clock capable I/O (MRCC) pin
  - Gigabit transceiver clock
- Can drive only the BUFIO and BUFR in the same region, region above and region below



13650!*note

The BUFMRCE has an attribute that selects whether the CE is synchronous, allowing glitch-free enabling and disabling of the clock, or asynchronous.

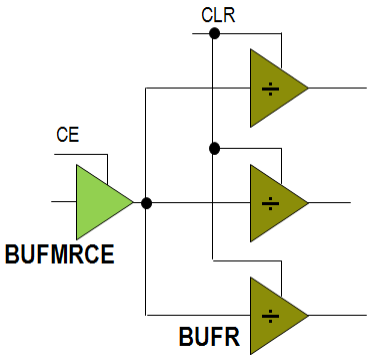
Gigabit transceiver clocks can only drive BUFMRs if the transceivers are in a separate column. If the transceiver replaces a region in the I/O and CMT column, no BUFMR exists in that region.

Slide 6-25:

Synchronizing BUFRs Driven by a BUFMR

13651**slide

- In order to clock a single interface that spans multiple banks, a BUFMR must be used to drive the BUFIO and BUFR in the different regions



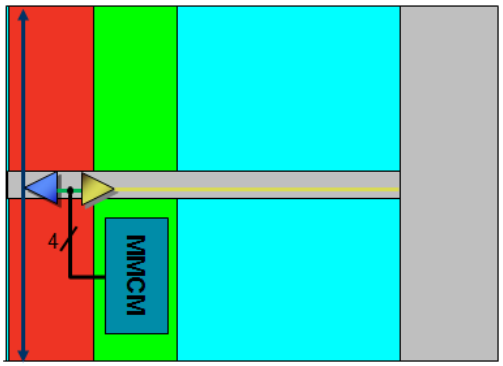
- The dividers on each BUFR are independent; they must be synchronized in order to ensure proper operation of the interface
  - Use a BUFMRCE to disable the clock feeding the BUFRs
  - Assert the CLR on all the BUFRs
    - This resets the dividers in the BUFRs
  - Deassert the CLR on all the BUFRs
    - This allows the dividers to start on the next rising edge the input clock (currently gated)
  - Assert the CE on the BUFMR
    - Starts the clocks to all BUFRs
    - BUFRs are now in sync

Slide 6-26:

High-Performance Clocks

13652**slide

- Four performance paths driving the I/O column clocked resources
- Driven by
  - MMCM outputs O0-O3
- Can drive
  - BUFIO
  - BUFR
  - OSERDES clocks directly



- Cleanest path from MMCM to I/O columns
- Lower jitter than any other routing
- Software automatically places critical signals onto performance path routing

13652!*note

Performance path routing enables fast I/O interfaces.

If you drive a BUFIO and / or BUFR in a region with the O0–O3 outputs of the MMCM in the same clock region, then the software will automatically use the high-performance paths for this route.

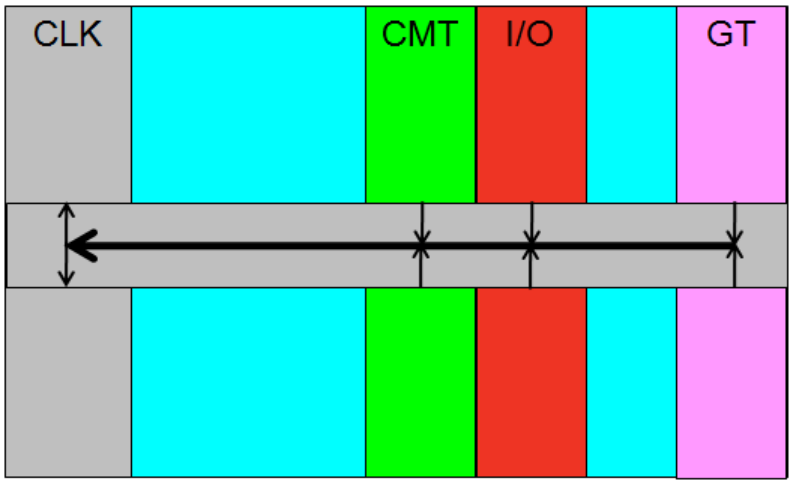
Note: The other outputs of the MMCM and the output of the PLL cannot use these paths.

Slide 6-27:

Connections to the Center Column

13653**slide

- The BUFGCTRL and BUFH reside in the center column of the FPGA
- The CCIO, BUFIO, and BUFR reside in the I/O column
- The MMCM and PLL reside in the CMT column, immediately adjacent to the I/O column
- GTs source clocks in the GT column
- Dedicated routing paths exist in the HCLK row to bring clocks between these resources
  - The tools will use these routes when necessary



13653!*note

There are 14 dedicated nets to bring clock inputs to the central clocking column. These can drive the BUFHs for the region or the BUFGCTRLs for the device.

The 14 nets can be driven by:

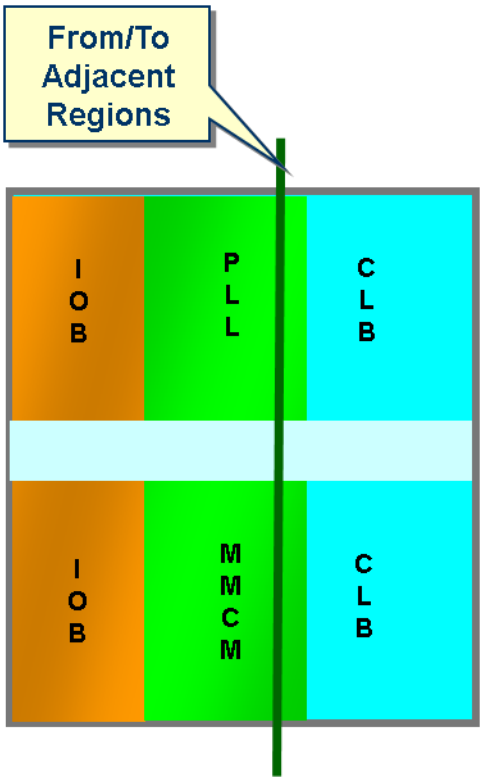
- The four CCIO in the I/O column
- The MMCM and PLL outputs in the CMT column
- The gigabit transceiver clock outputs in the GT column

Slide 6-28:

CMT Backbone

39916**slide

- Dedicated connections from CMT resources in one clock region to adjacent regions
  - Four segmented lines
  - Driven by CCIO and MMCM outputs O0 to O3
- Used primarily by the Memory Interface Generator (MIG)
  - Needed by phaser logic when driving wide interfaces



39916!*note

The CMT backbone is used primarily by the Memory Interface Generator for generating wider memory interfaces (more than one bank). When not used for this purpose, however, it is available for user logic.

The backbone allows the CCIO and first four outputs of the MMCM in one bank to access MMCM / PLL inputs in adjacent banks.

When accessing the CMT backbone from a CCIO, the same CCIO must also access an MMCM / PLL input in the same bank, unless CLOCK_DEDICATED_ROUTE=BACKBONE.

The backbone can be used to access MMCMs / PLLs in un-bonded banks. It can also be used to cascade MMCMs from adjacent regions together.

These should not be confused with the BUFMR routing.

CMT backbone connections are not de-skewed by the MMCM / PLL; additional skew should be anticipated when using a CCIO from a different region to drive an MMCM / PLL.

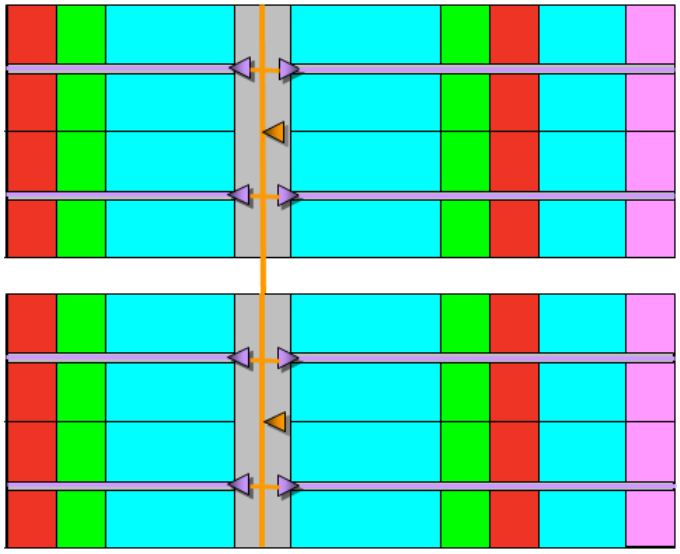
CMT backbone connections can cross between SLRs on devices using stacked interconnect technology.

Slide 6-29:

Stacked Silicon Interconnect Implications

13654**slide

- Larger 7 series devices are made up of Super Logic Region (SLR) blocks
  - SLR boundaries are on clock region boundaries



- The vertical spines of the global clock run through all SLRs
  - Each SLR has 32 BUFGCTRLs
  - Any 32 of the total number can be used
  - Expect increased clock skew between SLRs—the software tools will automatically account for this

- BUFMR tracks cannot pass from one SLR to the next

13654!*note

In devices that are made up of multiple SLRs, there are still 32 vertical spines of the global clock. These spines traverse all the SLRs.

Each SLR has the composition of a full FPGA. There are multiple clock regions, I/O banks, configuration blocks, etc. Therefore, each SLR has 32 BUFGCTRLs that can drive the vertical clock spines. However, only one BUFGCTRL can drive any one clock spine, regardless of which SLR the BUFGCTRL resides in. Thus, though there are Nx32 BUFGCTRLs in the complete device and only 32 of them can be used.

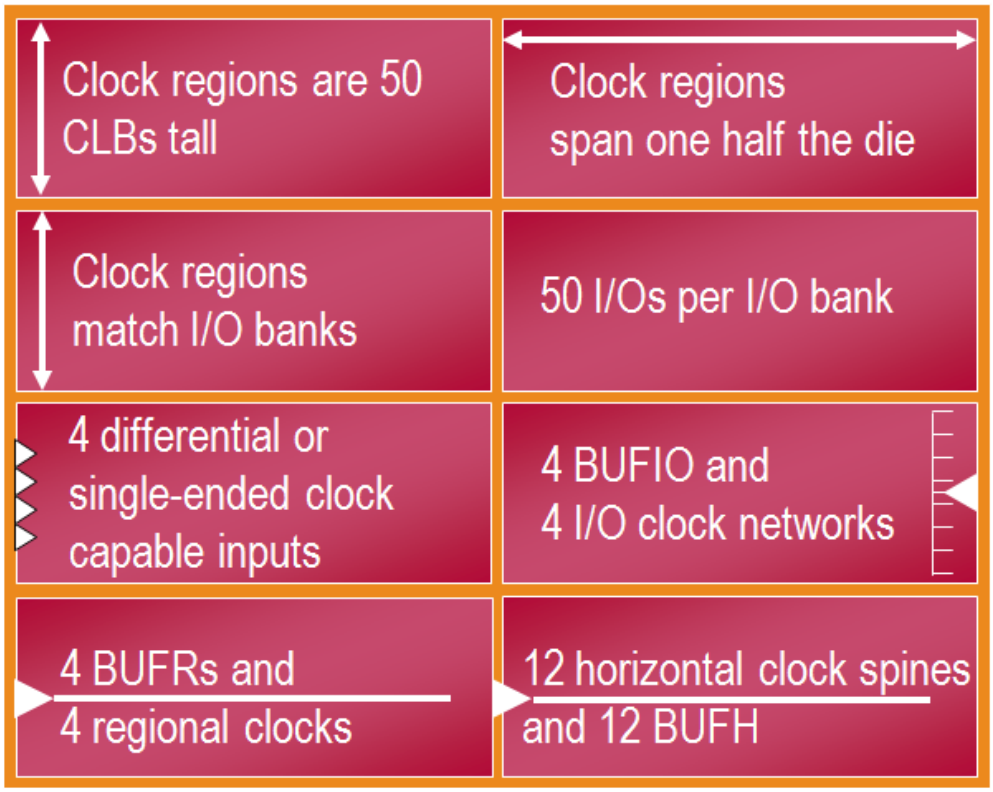
Because each SLR is a different die, its fabrication process can vary more from SLR to SLR than can be expected from a single die. Therefore, it is possible to see more clock skew on the global clock tree between different SLRs. The Vivado® IDE will automatically take this into account during placing and timing analysis. However, to obtain the best performance, it may be a good idea to perform floorplanning to restrict blocks to one SLR.

The tracks that connect the BUFMR outputs to the BUFIO / BUFR in adjacent regions do not cross from one SLR to the next. Therefore, the BUFMR in one SLR cannot drive the BUFIO / BUFR in an adjacent SLR.

Slide 6-30:

7 Series FPGA Clock Network Summary

13655**slide



Slide 6-31:

Clock Management Tile

13656**slide

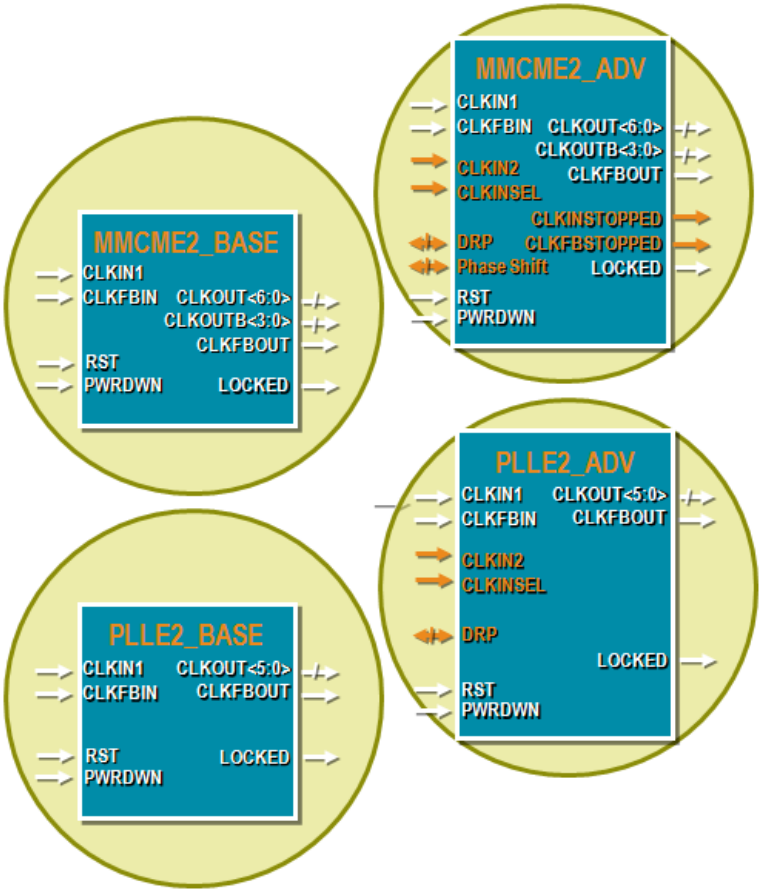
- Overview
- Clock Networks and Buffers
- **Clock Management Tile**
- Usage Models
- Using Clock Resources
- Summary

Slide 6-32:

MMCM and PLL Features

13657**slide

- Up to 24 CMTs per device
  - One MMCM and one PLL per CMT
- Two software primitives
  - *_BASE has only the basic ports
  - *_ADV provides access to all ports
- PLL is primarily intended for use with the I/O phaser for high-speed memory controllers
- The MMCM is the primary clock resource for user clocks



13657!*note

Each CMT has one MMCM and one PLL. The two resources share some of the same basic functionality, but the MMCM has some additional features for manipulating user clocks.

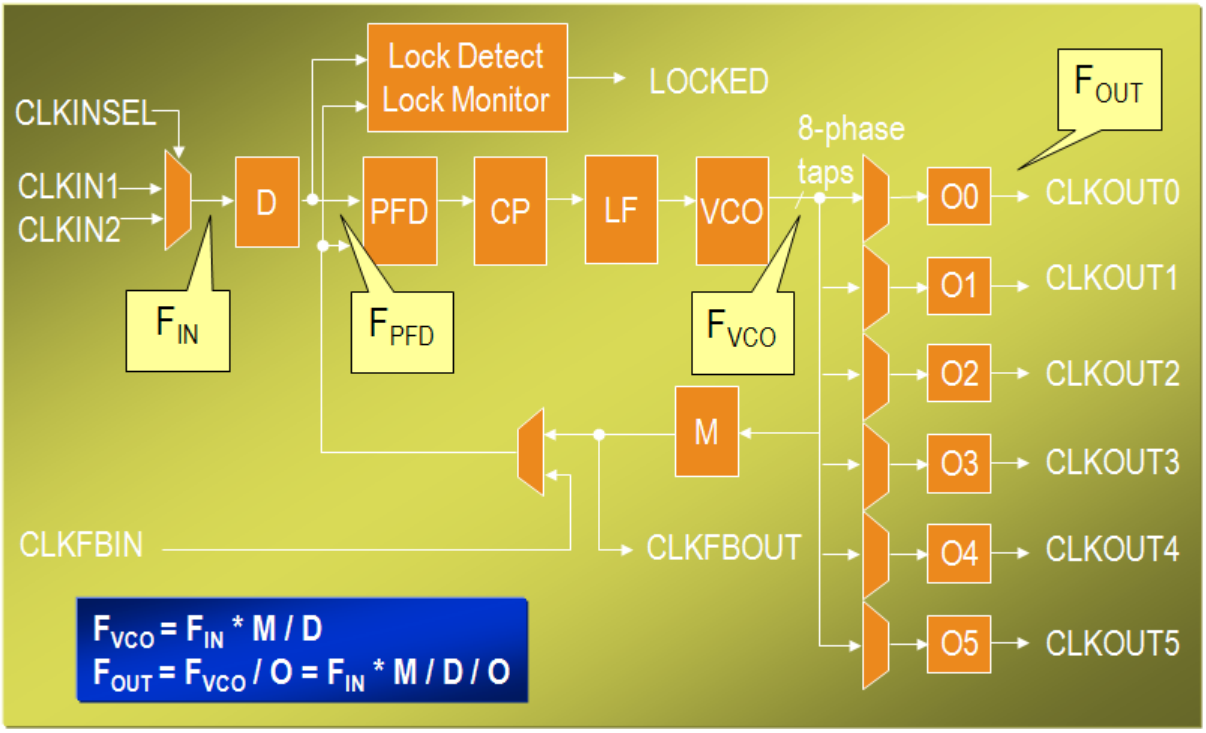
While the PLL can also be used for user clocks, it is primarily intended to be used in conjunction with the I/O phaser and I/O FIFOs for high speed memory controllers.

While there is only one MMCM per CMT (as opposed to two in the Virtex-6 FPGA), there are generally twice as many CMTs in a device, because there are two columns of CMTs (as opposed to one in the Virtex-6 FPGA). This allows the PLLs to be reserved for high-speed memory controllers.

Slide 6-33:

PLL Architecture

13658**slide



13658!*note

D: Input clock Divider.

PFD: Phase Frequency Detector – Compares input and feedback, generating phase proportional voltage.

CP: Charge pump – Integrates change information from PFD.

LF: Filters CP signal – Limits frequency change rate of VCO.

- Programmable Loop Filter characteristic trades off bandwidth vs. locking range.

VCO: Voltage Controlled Oscillator – Generates a clock based on the voltage output of the LF.

- Generates eight phases: 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°.

O0-O5: Output counters – Independent dividers of the VCO frequency.

- Can take in any one of the 8 phases.

M: Feedback counter (Multiplier) – Divides the VCO clock to *re-generate* the input clock.

- Closes the loop.
- Any one of the O0-O5 outputs can be used via the CLKFBIN path.

Slide 6-34:

PLL Features

13659**slide

- Frequency lock between F_{PFD} and CLKFB
- Phase lock between CLKIN and CLKFB
- Frequency synthesis for up to six different outputs
  - Fout = Fin * M / (D*O)
    - One M and one D value per PLL
    - Each PLL output can have its own O value
    - M: 1-64; D: 1-56; O: 1-128
- Static phase shifting increments of 1/8 of the VCO period
  - Each output has independent selection of the phase

- Duty cycle generation in increments of 1/8 of the VCO period
- Reference clock switching
- Lock detection
- PWRDWN signal to disable regulated power supply

13659!*note

There is one M and D divider per PLL. These two dividers determine the frequency of operation of the VCO.

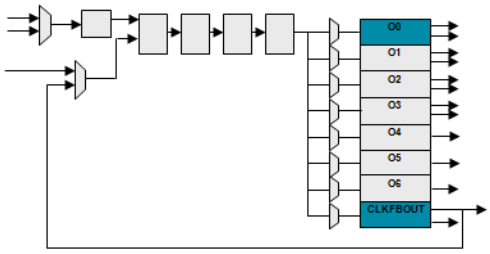
$F_{VCO} = F_{CLKIN} * M / D$

Each output has its own divider, O. The output frequency is F_{VCO} / O.

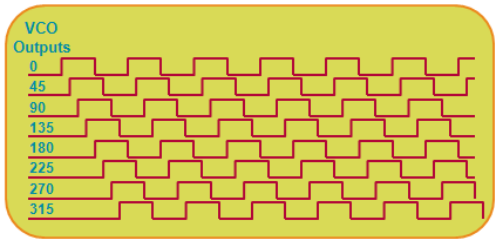
Slide 6-35:

Extra MMCM Features

13660**slide



- Fractional counters
  - Ability to configure O0 and M as a counter with 1/8th granularity (for example, 2.125, 2.250, 2.375, etc.)
  - Enables many more VCO frequencies and output frequencies to be synthesized
- Dynamic phase shifting
- Fine phase shifting
- Divider cascading for wider divider range
  - O4 divider uses resources from O6

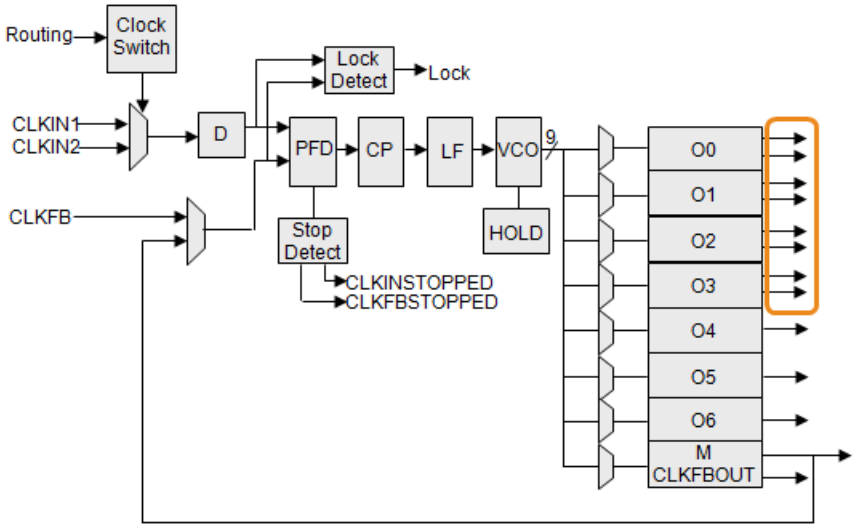


Slide 6-36:

Additional MMCM Signals

13661**slide

- Complement outputs
  - O0-O3 of every MMCM have both true and complement outputs



- CLKOUT6 output (with O6 divider)

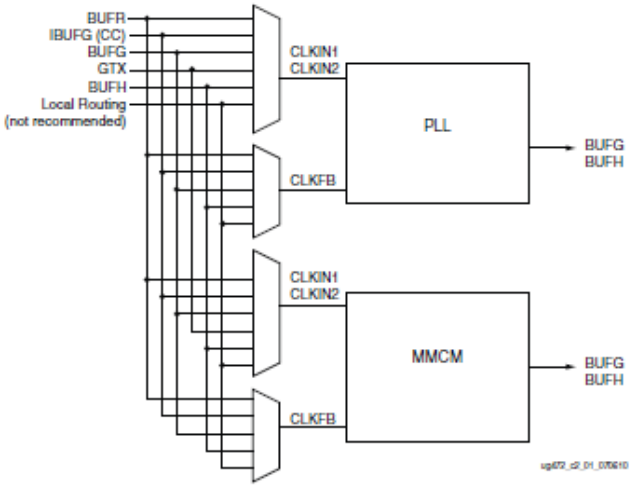
- Dynamic phase shift port
  - PSCLK
  - PSEN
  - PSINCDEC
  - PSDONE

Slide 6-37:

MMCM and PLL Inputs

13662**slide

- PLL and MMCM input clock ports (CLKIN1 and CLKIN2) can be source from
  - CCIO in the same clock regions (recommended)
  - CCIO in a vertically adjacent clock regions
  - BUFG
  - BUFR
  - BUFH
  - Interconnect (not recommended)
  - Gigabit transceiver clock



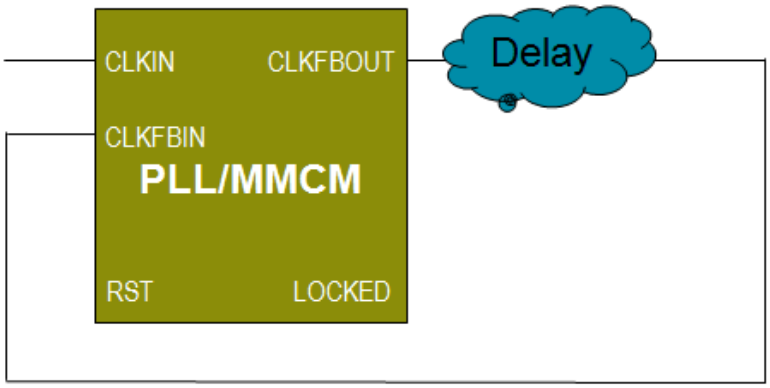
- CLKFB can be sourced by the same sources except the gigabit transceiver clock
- PLL / MMCM outputs can drive BUFG and BUFH directly
- MMCM high-performance clock outputs (O0 – O3) can drive BUFIO, BUFR, and CLK of OSERDES in the same region directly

Slide 6-38:

Phase Locking

13663**slide

- The control loop of the PLL / MMCM works to keep the CLKIN and CLKFBIN clocks in phase
  - Regardless of the delay between the CLKFBOUT and the CLKFBIN
  - The compensation is dynamic
    - Even if the magnitude of the delay changes in time, the PLL will adjust to keep the two clocks in phase

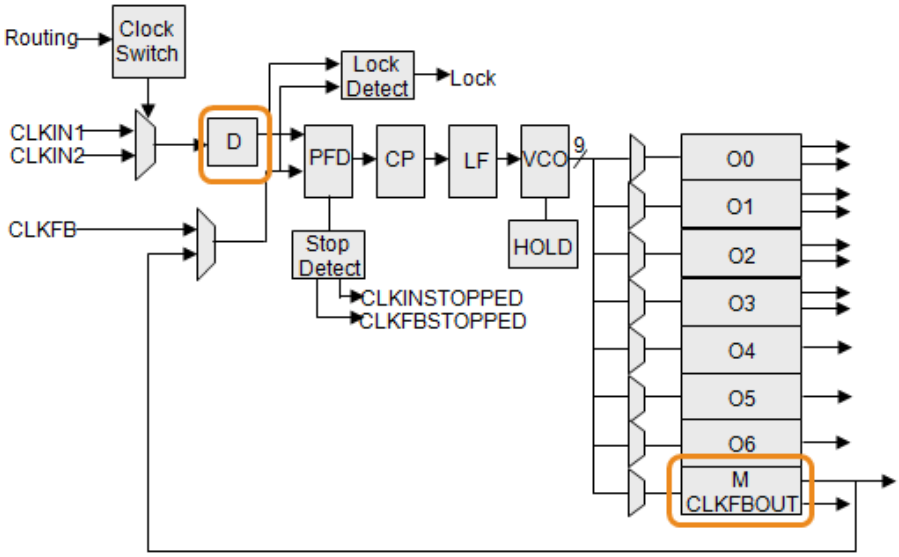


Slide 6-39:

Frequency Generation

13664**slide

- The same mechanism that keeps clocks in phase generates new frequencies
- By placing a divider of M between the VCO and the CLKFBOUT output, the VCO must run at M times the CLKIN frequency in order to achieve phase lock
- Similarly, by placing a D divider between the CLKIN port and the PFD block, the VCO must now run at M / D times the frequency of CLKIN in order to achieve phase lock



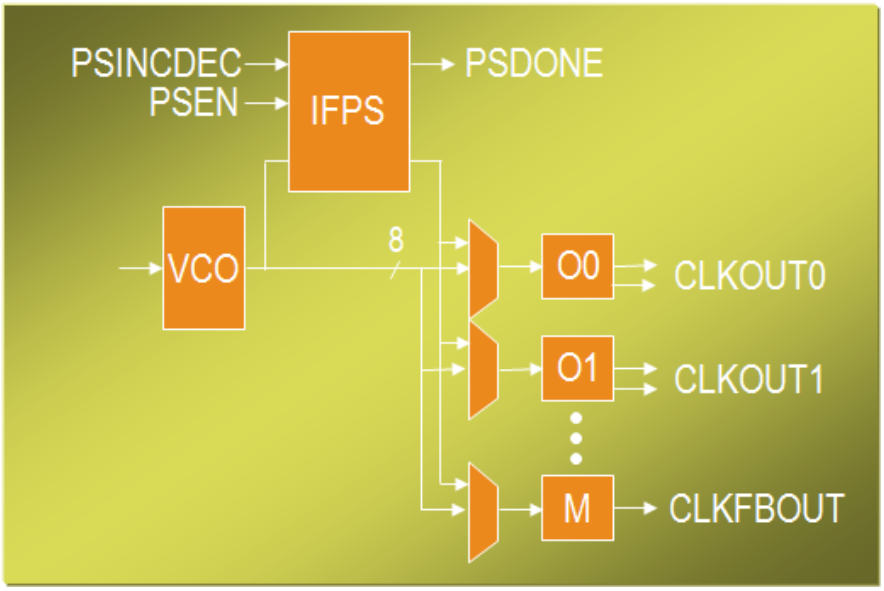
- Finally, the output dividers divide the VCO frequency for the output ports

Slide 6-40:

Interpolated Fine Phase Shift (IFPS)

13665**slide

- In addition to the static phase shifting capability available in the PLL and MMCM, the MMCM can also perform interpolated fine phase shifting
  - The VCO output can be shifted in increments of 1/56 of the VCO period
  - The shift amount can be set statically or can be dynamically controlled
- The shifted VCO output can be selected as the input to any of the clock outputs or the CLKFBOUT output
- The shift amount is 0 to 360°
  - The shift amount wraps around



13665!*note

The VCO generates eight phase shifted outputs, each shifted by 45°.

The Interpolated Fine Phase Shift (IFPS) module generates a ninth output from the VCO, which can be shifted by any amount, in increments of 1/56 of the VCO period (therefore 6.43°).

Any of the eight output dividers (including the M divider for the CLKFBOUT output) can select any eight of the static phase-shifted VCO outputs or the fine phase-shifted VCO output.

Due to the phase locking at the PFD, shifting the CLKFBOUT output clock will result in the VCO phase advancing to maintain lock. Thus, a positive phase shift on the CLKFBOUT will result in all outputs displaying a negative shift.

The value of the fine phase shift can be statically set or can be changed dynamically using the PSCLK, PSEN and PSINCDEC port; on any PSCK where PSEN is asserted, the phase will advance by 1/56 (PSINCDEC=1), or retard by 1/56 (PSINCDEC=0). Once initiated, another shift cannot be attempted until PSDONE is signaled (for one PSCLK).

Slide 6-41:

Usage Models

13666**slide

- Overview
- Clock Networks and Buffers
- Clock Management Tile
- **Usage Models**
- Using Clock Resources
- Summary

13666!*note

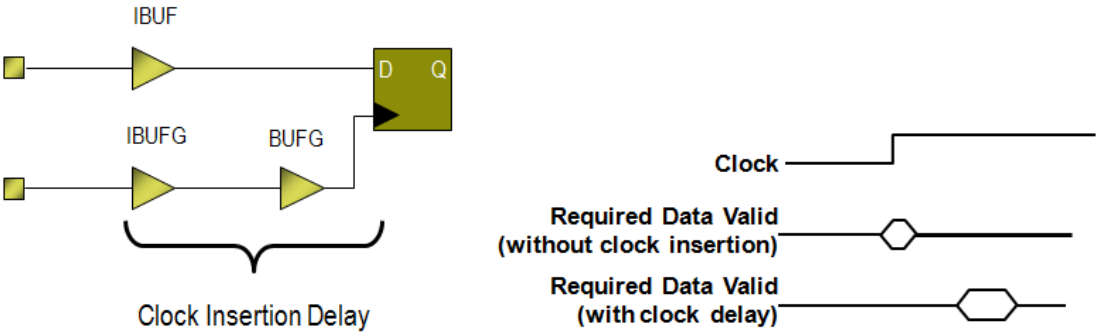
The next several slides illustrate different methods for connecting the MMCM or PLL to manipulate clock signals.

Slide 6-42:

Clock Insertion Delay

13667**slide

- Bringing the clock in from an IBUFG, through a BUFG, and distributing it through the global clock tree introduces delay on the clock path
  - Delay is Process / Voltage / Temperature (PVT) dependent
  - Increases setup / hold window on input flip-flops
  - Increases output valid uncertainty on output flip-flops
  - Restricts high-speed interface design using global clock resources



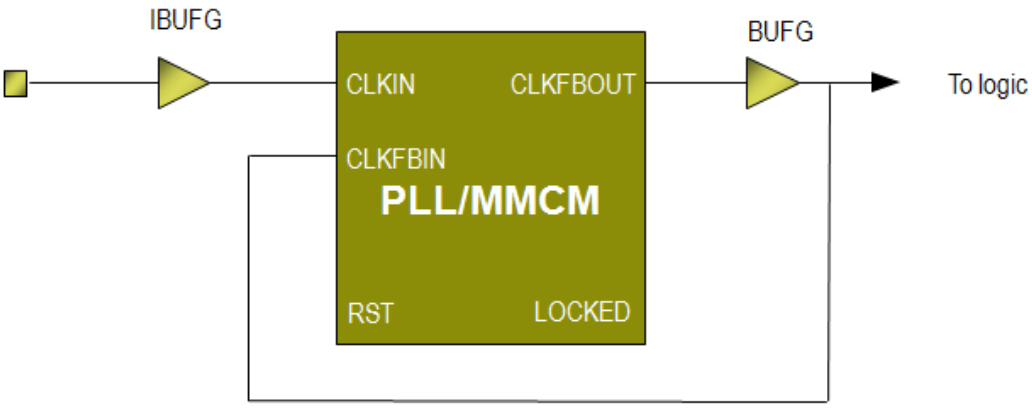
13667!*note

The clock insertion delay depends on the depth of the clock tree, which increases with FPGA size.

Slide 6-43:

Clock Insertion Delay Removal Example

13668**slide



- The PLL / MMCM keeps CLKIN and CLKFBIN in phase, canceling out all delay between CLKFBOUT and CLKFBIN
  - In this case, the delay is that of the BUFG and global clock network
  - The delay is canceled regardless of PVT variation over time
- Thus, the clock arriving at the logic driven by the global clock is in phase with the input clock; the clock insertion delay is removed

13668!*note

The PLL or MMCM can add some additional delay to the feedback path to cancel out the delay of the IBUFG. This is done automatically by the tools by setting the COMPENSATION attribute to ZHOLD. This will guarantee a zero or negative hold time requirement for all IOBs driven by the global clock.

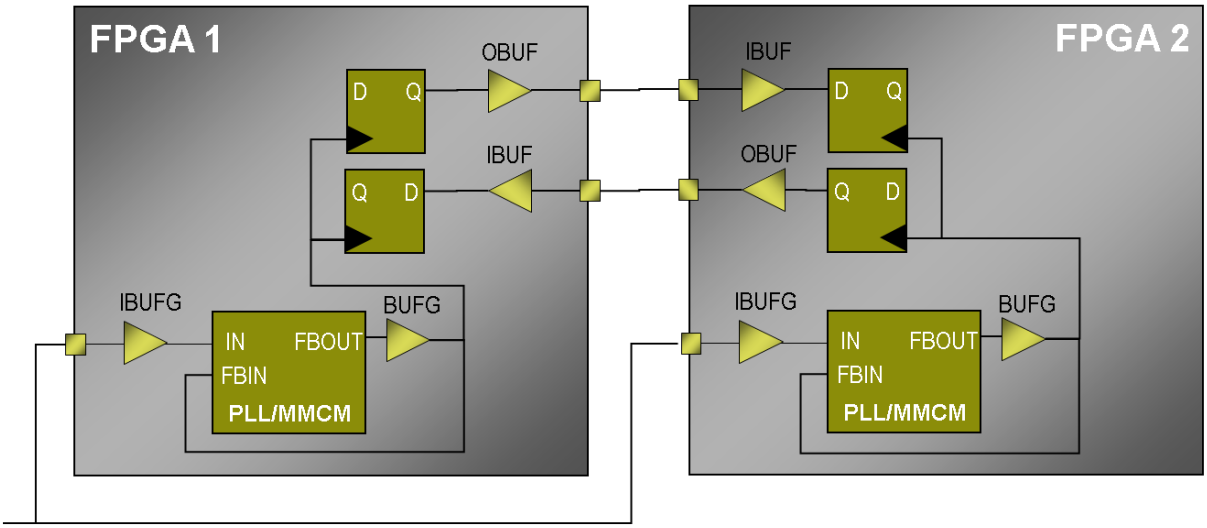
Both PLLs and MMCMs need time after configuration to align the feedback clock to the input clock. For details about lock times, see the datasheet for your device.

Slide 6-44:

System-Synchronous Clocking

13669**slide

- Internal de-skew allows for synchronous interfaces with no hold-time violations
  - The tools will automatically select COMPENSATION=ZHOLD



13669!*note

By using the MMCM or PLL to remove the clock insertion delay, system components using a common clock can easily communicate by using simple synchronous interfaces. Hold-time requirements of the receiving flip-flops are guaranteed to be met. Communication can be done at moderate speeds over significant distances.

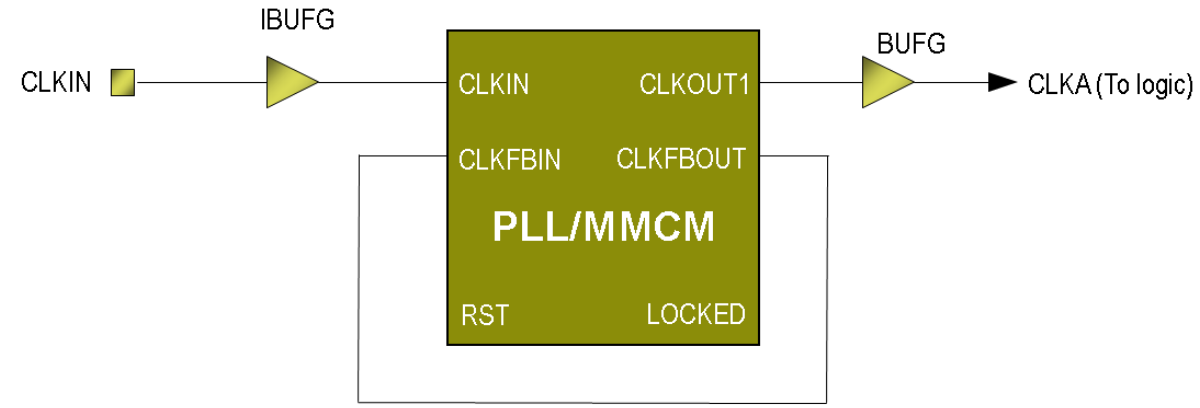
The length of the board routing from the common clock to the IBUFG inputs in both FPGAs should be matched.

Note that the same type of input and output buffers should be used to balance this system.

Slide 6-45:

Using Internal Feedback

13670**slide

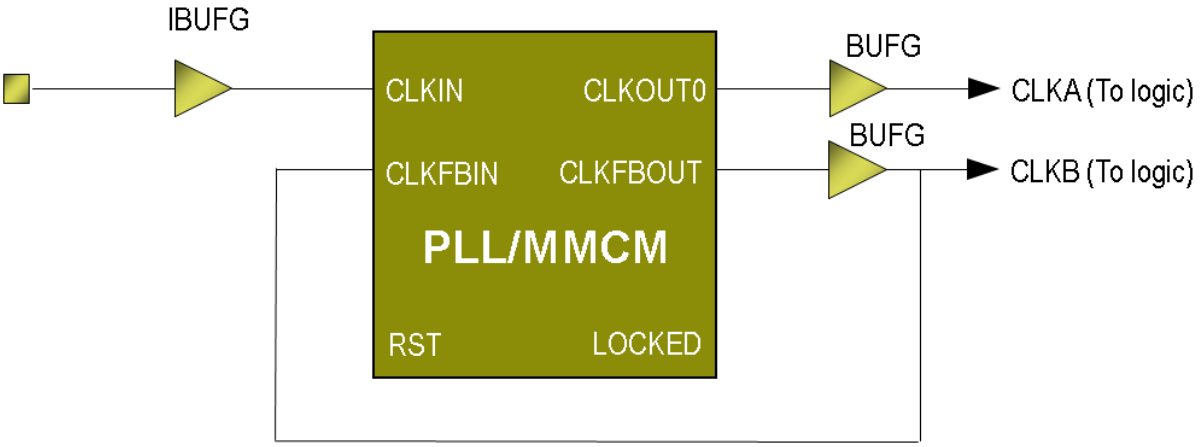


- When using a PLL / MMCM for frequency synthesis or jitter reduction only
  - Phase of CLKA has no requirements with respect to CLKIN
- Tools will select the internal feedback path, which requires no routing or logic outside the PLL / MMCM
  - Tools will automatically set COMPENSATION=INTERNAL

Slide 6-46:

Using Multiple PLL/MMCM Clocks Internally

13671**slide



- Multiple outputs of the MMCM / PLL have known phase relationships
  - For example, CLKA may be the same frequency as CLKB, phase shifted 180°
- The delay through all BUFGs and clock networks are identical
  - So CLKA and CLKB arrive at loads with the same phase relationship
  - Crossing from one clock domain to the other can be accomplished synchronously

13671!*note

Because all BUFGs are identical (running at the same process / temperature / voltage), and the global clock networks are fully buffered (hence, load independent), the frequency and phase relationships of PLL/MMCM outputs are preserved through the clock networks.

The implementation and timing analysis tools understand these relationships and correctly analyze the paths between flip-flops on these different domains.

For example, if CLKOUT0 is programmed to be twice the frequency of CLKFBOUT:

- The tools know that static paths between the two domains are constrained by the period of the faster clock.

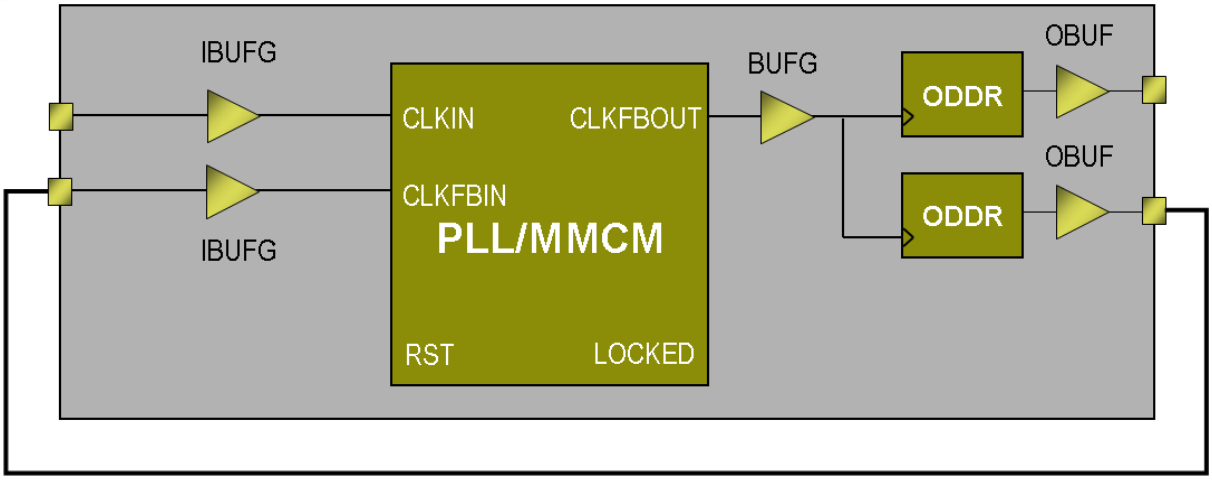
If the frequency is the same, but the phase is shifted (CLKOUT1 advanced by 90° from CLKFBOUT):

- Paths from CLKA to CLKB are constrained to ¾ of a clock period.
- Paths from CLKB to CLKA are constrained to ¼ of a clock period.

Slide 6-47:

Clock Mirror Example

13672**slide



- The PLL or MMCM keeps the input and feedback clock in phase
  - The feedback clock is not restricted to being the output of the global clock network

- In this example, two (or more) identical output clocks are generated
  - One is used as the feedback clock
  - The others are mirrors; hence, in phase with each other and the input clock

13672!*note

The delay through the two ODDRs and OBUFs are matched. They have the same process, temperature and voltage and, hence, will have the same characteristics.

The delay through the two IBUFGs is also matched.

The delay of the BUFG, global clock network, ODDR flip flop, output buffer, and routing track back to the pin driving the CLKFBIN port will be canceled.

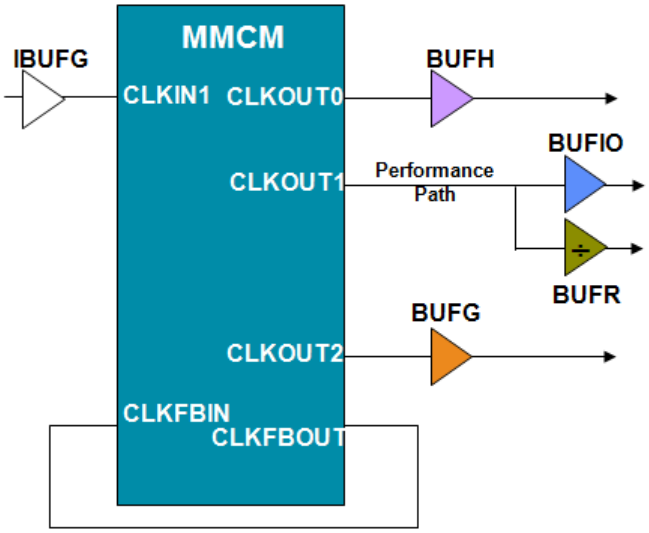
The tools will automatically select COMPENSATION=EXTERNAL.

Slide 6-48:

Design Example

13673**slide

- Requirement
  - 33.3-MHz external oscillator controls
    - 533.3-MHz data being generated by I/O logic (BUFIO)
    - Large amount of logic at 66.6 MHz (BUFG)
    - Small design at 54 MHz (BUFH)
  - Phase relationship between input clock and output clock is irrelevant



- Solution
  - MMCM values
    - M=32, D=1. FVCO = 1066.7
    - O0=19.75 – uses fractional divider
    - O1=2
    - O2=16

13673!*note

This example shows how a single MMCM can be used to generate the multiple clocks required by your system.

This reduces overall system cost.

Slide 6-49:

Using Clock Resources

13674**slide

- Overview
- Clock Networks and Buffers
- Clock Management Tile
- Usage Models
- **Using Clock Resources**
- Summary

Slide 6-50:

Inference

13675**slide

- Clock networks are represented by nets in your RTL design
  - The mapping of an RTL net to a clock network is managed by using the appropriate clock buffer to generate that net
- Certain resources can be inferred
  - A primary input net (with or without an IBUF instantiated) will be mapped to a global clock if it drives the clock inputs of clocked resources
    - The BUFG will be inferred
  - BUFH drivers will be inferred whenever a global clock (driven by a BUFG) is required in a clock region
    - BUFHs for each region required will be inferred
- BUFIO, BUFR, and BUFMR cannot be inferred
  - Instantiating these buffers tells the tools that you want to use the corresponding clock networks

- PLLs and MMCMs cannot be inferred

Slide 6-51:

Instantiation

- All clocking resources can be directly instantiated in your RTL code
  - Simulation models exist for all resources
  - Refer to the Library Guide for HDL Designs
  - Use the Language Templates ( )
- PLLs and MMCMs have many inputs and outputs, as well as many attributes
  - Optimal dividers for obtaining the desired characteristics may be hard to derive
  - The Clocking Wizard IP can help

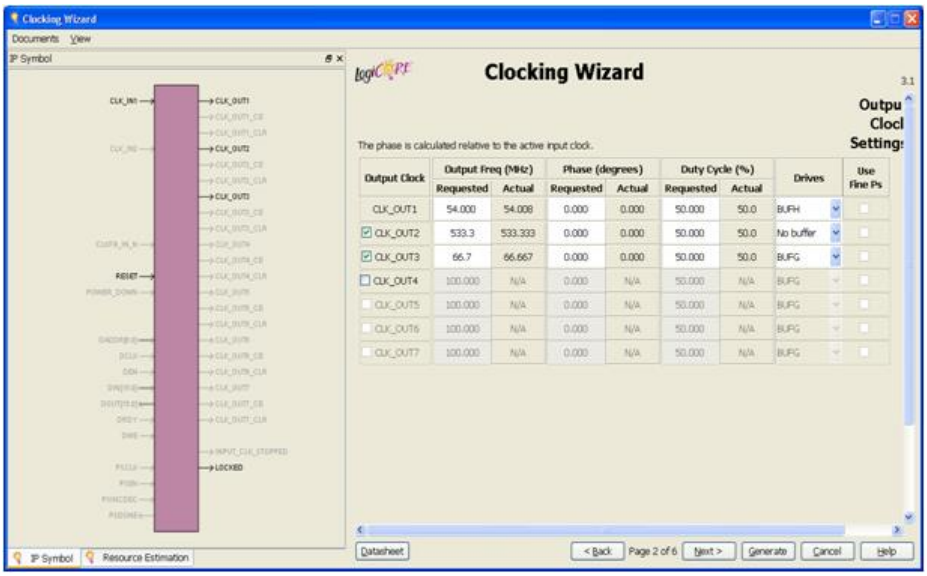
Slide 6-52:

Clocking Wizard

13677**slide

על הטקסט שברצונך שיופיע כאן.1 Heading שגיאה! השתמש בכרטיסיה בית כדי להחיל

- The Clocking Wizard walks you through the generation of complete clocking subsystems
  - Specify the input frequency and desired output frequencies, phases, and duty cycles
  - Specify the desired buffers
- Chooses the optimal PLL / MMCM settings
- Instantiates all clock and input buffers



Slide 6-53:

Summary

13678**slide

- Overview
- Clock Networks and Buffers
- Clock Management Tile
- Usage Models
- Using Clock Resources
- **Summary**

Slide 6-54:

Apply Your Knowledge

13679**slide

7) What types of clock networks are available in the 7 series?

8) What FPGA resources can each clock network drive?

9) How can you use the same I/O clock in more than one I/O bank?

13679!*note

10) What types of clock networks are available in the 7 series?

- Global clock network
  - Driven by a BUFGCTRL via BUFH
  - 32 BUFGCTRLs per device
  - 12 BUFHs per clock region, any 12 global clocks can enter each region
- I/O clock network

  - Four I/O clock networks per region
  - Driven by a BUFIO
  - Four BUFIOs per I/O bank (two are single region, two are multi-region)
- Regional clock network
  - Two regional clock networks per region
  - Driven by a BUFR
  - Two BUFRs per I/O bank

11) What FPGA resources can each clock network drive?

A global clock network can clock all synchronous elements within each region where that clock is used.

An I/O clock network can clock only the resources in the ILOGIC / ISERDES and OLOGIC / OSERDES in that I/O bank.

Regional clock networks can drive all synchronous elements within their region.

12) How can you use the same I/O clock in more than one I/O bank?

A BUFMR can be instantiated on a multi-region clock capable I/O. The output of the BUFMR can drive the BUFIO (and BUFR) in its bank and in the bank above and below.

Slide 6-55:

Summary

13680**slide

- Clock region height = 50 CLBs, 50 IOBs
- Four clock-capable inputs per bank
- Thirty-two global clock buffers
  - Twelve global clocks per region, delivered via BUFH

- Four BUFIOs per I/O bank
- Two BUFRs per I/O bank
- Two BUFMR per I/O bank
- Two columns of Clock Management Tiles (CMTs) per device
  - Adjacent to I/O banks
  - The CMT has one Mixed-Mode Clock Manager (MMCM) and one PLL

Capture Your Notes Here

Basic Timing Constraints (XDC)

Slide 7-1:

57604**slide

2014.1
57604!*note

This module describes basic timing constraints, such as `create_clock`, set_`input_delay` and set_`output_delay`, for communicating timing objectives. 30 minutes.

Slide 7-2:

Objectives

57635**slide

After completing this module, you will be able to:

- Apply the `create_clock`, `set_input_delay`, and `set_output_delay` constraints to a simple synchronous design
- Use the Constraints Editor to specify basic timing constraints

Slide 7-3:

Overview

57606**slide

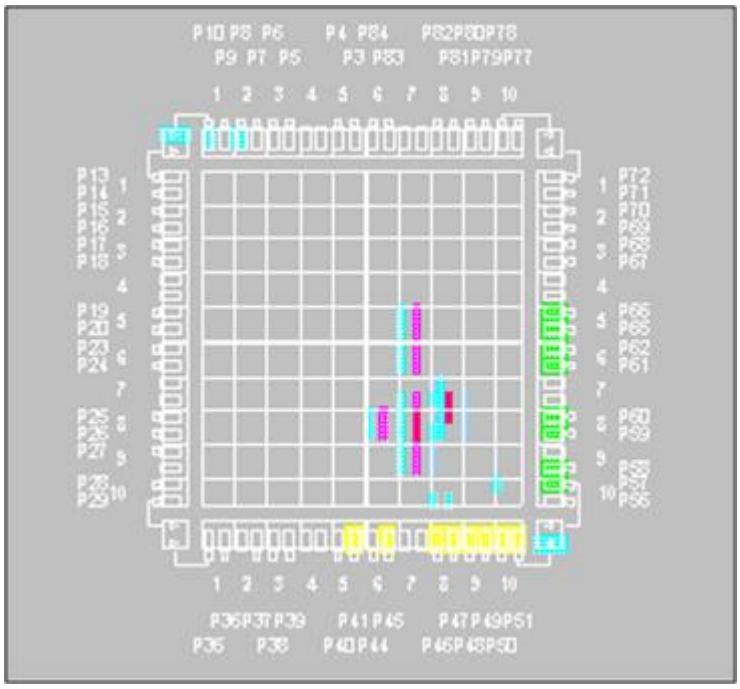
- **Overview**
- Basic Timing Constraints
- Constraints Editor
- Summary

Slide 7-4:

Designs without Timing Constraints Produce Sub-optimal Results

57607**slide

- Note the logical structure of the placement and pins



57607!*note

This design had no timing constraints or pin assignments. Note the logical structure of the placement and pins. The implementation tools do a good job of placing and routing the design without using timing constraints.

From this example, you can see that the logic is grouped closely to provide a good internal frequency and minimize clock skew.

Note that the tools do not anticipate any ground-bounce problems. The tools do not know when the signals will be toggled or how many signals may change simultaneously.

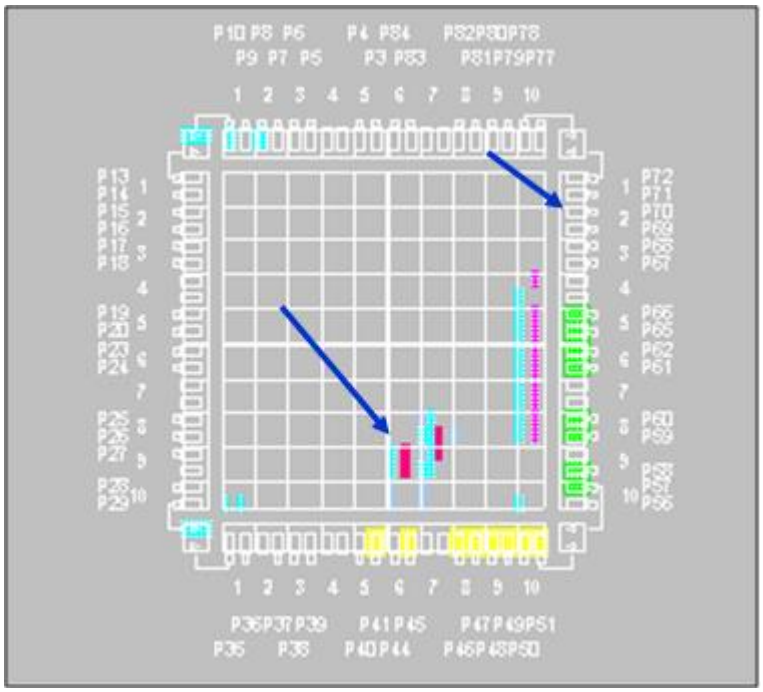
Be certain that you try to disperse switching outputs simultaneously to help avoid any problems with ground bounce and perform a noise analysis to determine if you can anticipate a problem.

Slide 7-5:

Designs with Basic Timing Constraints Have Better Performance

57608**slide

- Note that the logic is placed closer to the I/O pins



57608!*note

In this same design with basic timing constraints, I/O flip-flops were not used. The timing constraints moved the logic closer to the pins to improve on-chip and off-chip timing.

Note that the logic is placed closer to the I/O pins due to input and output timing constraints (`set_input_delay` and `set_output_delay` constraints) in addition to a `create_clock` constraint.

Timing constraints are used to communicate performance objectives to the software—which in turn is used to place and route the design such that it meets those objectives.

Slide 7-6:

Timing Constraints Define Performance Objectives

57610**slide

- Timing constraints define timing objectives
  - Over-constraining gets you nothing, but costs extra place & route time
  - Always apply timing constraints, even when the timing objective is modest
- Unrealistic timing constraints will cause the tools to stop
  - To avoid wasting implementation time, Xilinx recommends following Performance Baseline procedure discussed in the *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949)
- After implementing the design, review the Timing Summary report to determine if objectives were met
  - If constraints failed, use the same report to determine the cause

57610!*note

Timing constraints should be used to define the performance objectives of your design. Tight timing constraints will increase software compile time.

After implementing, review the Post-Place & Route Timing Summary report to determine whether the design performance objectives were met. If constraints were not met, use the Timing Summary report to determine the cause.

You can learn more about interpreting timing reports in the V*ivado Design Suite Static Timing Analysis and Xilinx Design Constraints* course.

Slide 7-7:

I/O Pads and Synchronous Elements Make Path Endpoints

57611**slide

- I/O pads
- Synchronous elements

57611!*note

There are two types of path endpoints: I/O pads and synchronous elements (synchronous elements include flip-flops, latches, RAMs, and DSP48 slices).

Slide 7-8:

Creating a Timing Constraint Is a Two-Step Process

57612**slide

- Step 1: Create groups of path endpoints
- Step 2: Specify a timing requirement between the groups

57612!*note

Creating a timing constraint is done in two steps. First, you need to create groups of path endpoints, and then you need to specify a timing requirement between the groups.

When you create basic timing constraints, the Vivado® tools make it easy by automatically creating the necessary groups of endpoints (all synchronous elements in a clock domain, all input pins, etc.).

Slide 7-9:

Apply Your Knowledge

57613**slide

13) It drives synthesis, packing, placement, and routing. What is it?

57613!*note

14) It drives synthesis, packing, placement, and routing. What is it?

- A timing constraint.

Slide 7-10:

Basic Timing Constraints

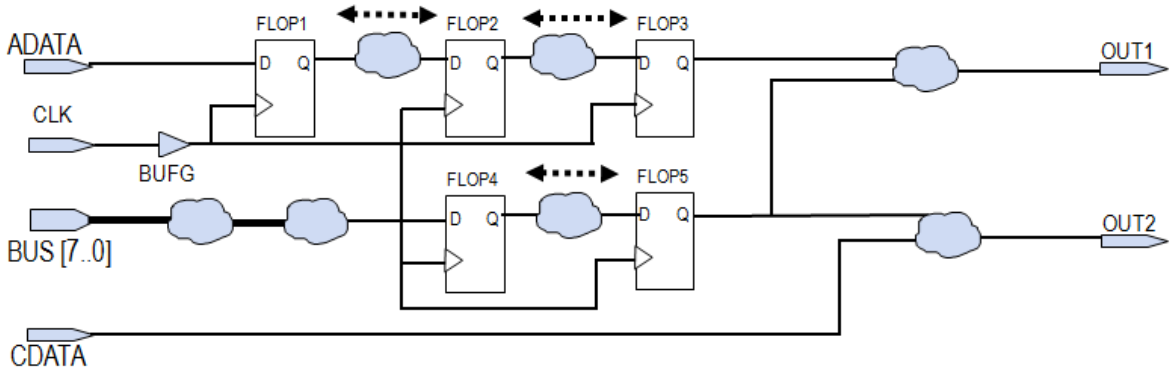
57614**slide

- Overview
- **Basic Timing Constraints**
- Constraints Editor
- Summary

Slide 7-11:

create_clock Constraint Covers Paths Between Synchronous Elements

57615**slide



57615!*note

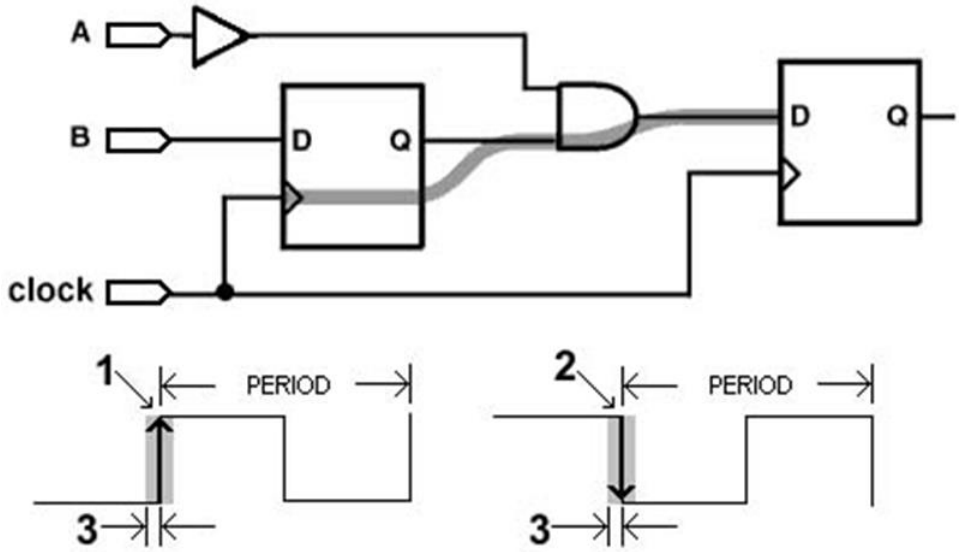
`create_clock` constraints cover paths between synchronous elements clocked by the reference net.

Slide 7-12:

create_clock Constraints Use the Most Accurate Timing Information

57616**slide

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



57616!*note

The `create_clock` constraint uses the most accurate timing information so that they can automatically account for clock skew between the source and destination flip-flops, synchronous elements clocked on the negative edge, unequal clock duty cycles, and clock input jitter.

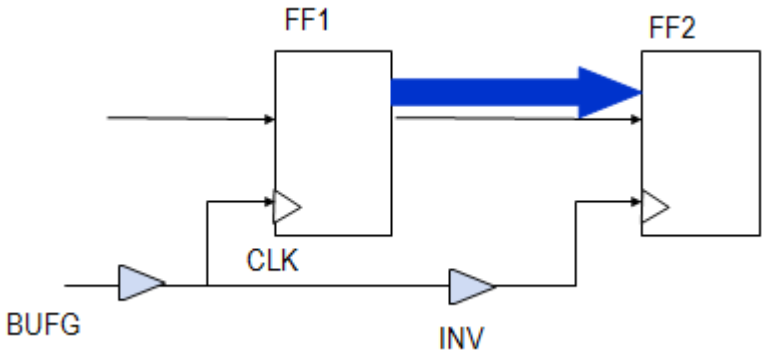
These features make constraining your design much easier and give the tools the most flexibility in meeting all of your timing constraints.

Slide 7-13:

Example of the create_clock Constraint

57617**slide

- Assume
  - 50-percent duty cycle on CLK
  - `create_clock` 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



57617!*note

Assume that there is a 50-percent duty cycle on CLK and a `create_clock` constraint of 10 nanoseconds.

Because FF2 will be clocked on the falling edge of CLK, the path between the two flip-flops will actually be constrained to 50 percent of 10 nanoseconds, which equals five nanoseconds.

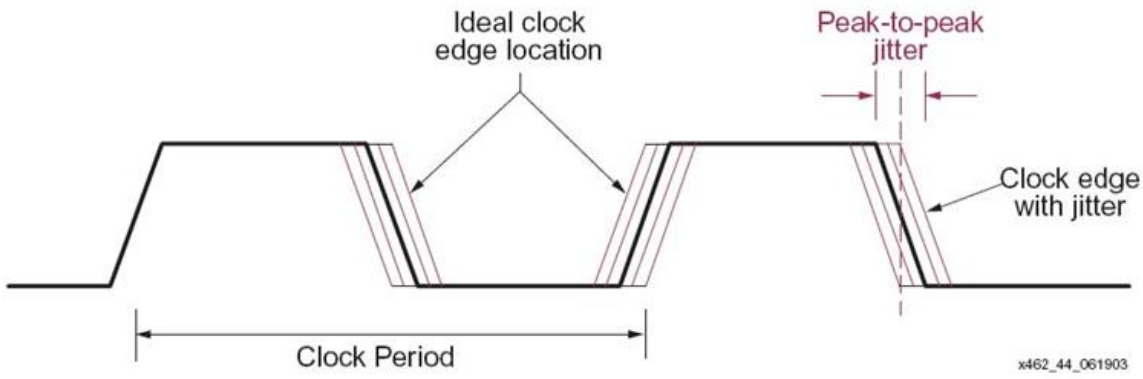
Clock input jitter will be discussed next.

Note that whenever you use the MMCM component, the `create_clock` constraint is placed on the input clock signal going into the MMCM component. The `create_clock` constraint will be propagated according to the MMCM outputs.

Slide 7-14:

Clock Input Jitter is One Source of Clock Uncertainty

57618**slide



57618!*note

Clock input jitter is one source of clock uncertainty. Add a Phase-Locked Loop (PLL) to your clock circuitry if your input clock has excessive jitter.

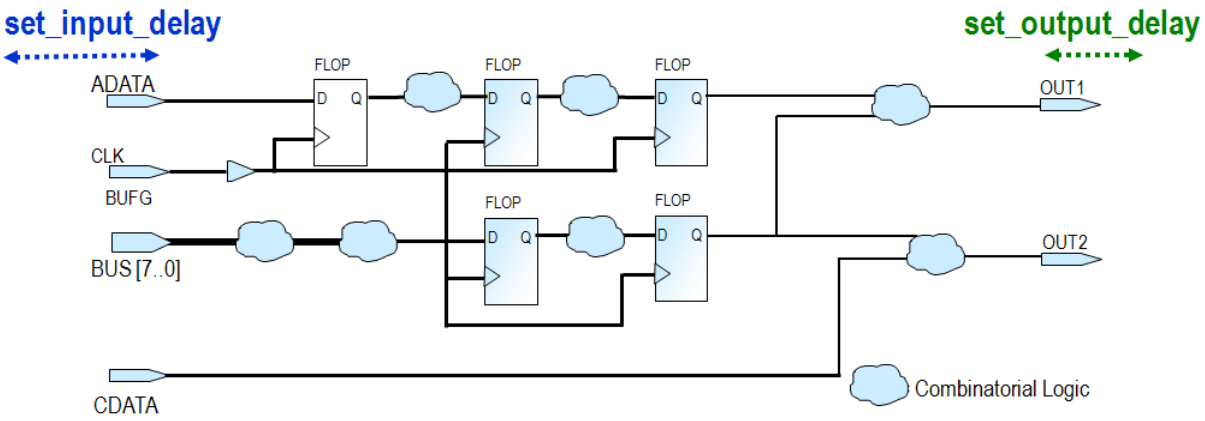
Clock uncertainty is automatically subtracted from the `create_clock` constraint setup paths and `input_delay` constraint setup paths.

Clock uncertainty is automatically added to `create_clock` constraint hold paths, the `input_delay` constraint hold paths, and `output_delay` constraint paths.

Slide 7-15:

set_input_delay and set_output_delay Constraints Effectively Cover I/O Pads To/From Synchronous Elements

57619**slide



57619!*note

`set_input_delay` and `set_output_delay` constraints cover paths from input ports to synchronous elements and synchronous elements to output ports, respectively.

`set_input_delay` constraints are associated with a clock net, which identifies the synchronous elements that will be considered as valid path endpoints for the constraint. Synchronous elements include flip-flops, latches, synchronous RAM components, and DSP slices.

Note that even after specifying a `set_input_delay`, `set_output_delay`, and `create_clock` constraints, it is still possible to have unconstrained paths, such as purely combinatorial paths. Remember that the `set_input_delay` and `set_output_delay` constraints do not cover paths between synchronous elements or purely combinatorial delay paths.

Slide 7-16:

set_input_delay and set_output_delay Constraints Account for Delay and Jitter

57620**slide

Path 61 - timing_1

Summary

Name	Path 61
Slack	1.250ns
Source	rxd_pin (input port clocked by clk_pin_p {rise@0.000ns fall@2.500ns period=5.000ns})
Destination	uart_rx_i0/meta_harden_rxd_i0/signal_meta_reg/D (rising edge-triggered cell FDRE clocked by clk_out1_clk_core
Path Group	clk_out1_clk_core
Path Type	Setup (Max at Slow Process Corner)
Requirement	5.000ns (clk_out1_clk_core rise@5.000ns - clk_pin_p rise@0.000ns)
Data Path Delay	4.830ns (logic 1.043ns (21.588%) route 3.787ns (78.412%))
Logic Levels	2 (IBUFCTRL=1 INBUF=1)
Input Delay	0.000ns
Clock Path Skew	1.157ns
Clock Uncertainty	0.137ns
Clock Net Delay (Destination)	1.573ns (routing 0.733ns, distribution 0.840ns)

Path 131 - timing_1

Summary

Name	Path 131
Slack	0.685ns
Source	samp_gen_i0/led_o_reg[4]/C (rising edge-triggered cell FDRE clocked by clk_tx_unbuf {rise@0.000ns fall@2.581ns period=5.16
Destination	led_pins[4] (output port clocked by clk_tx_virtual {rise@0.000ns fall@2.580ns period=5.161ns})
Path Group	clk_tx_virtual
Path Type	Max at Slow Process Corner
Requirement	5.161ns (clk_tx_virtual rise@5.161ns - clk_tx_unbuf rise@0.000ns)
Data Path Delay	2.495ns (logic 2.429ns (97.355%) route 0.066ns (2.645%))
Logic Levels	1 (OBUF=1)
Output Delay	0.500ns
Clock Path Skew	-1.344ns
Clock Uncertainty	0.137ns
Clock Net Delay (Source)	1.499ns (routing 0.331ns, distribution 1.168ns)
Clock Domain Crossing	Inter clock paths are considered valid unless explicitly excluded by timing constraints such as set_clock_groups or set_false_path.

57620!*note

`set_input_delay` and `set_output_delay` constraints also account for clock input jitter. They use the jitter defined on the associated `create_clock` constraint. They also take the clock distribution delay into account. Therefore, the tools evaluate the effective input path as the data input delay minus the clock delay.

Likewise, the effective output path is the data output delay plus the clock delay. Because the clock distribution delay is tangible in these cases, the input delays are reduced, and the output delays are increased when the clock distribution delay is taken into account.

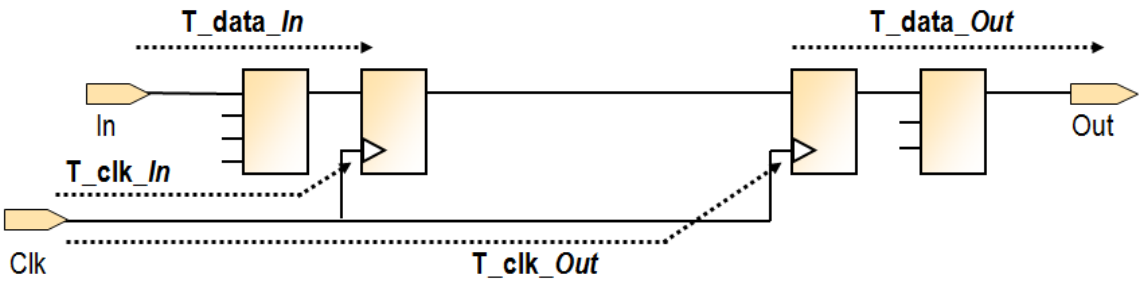
Because the tools account for the clock distribution delay, you can specify your pin-to-pin timing goals without knowing what the internal clock distribution delay will be. When analyzing timing for `set_input_delay` and `set_output_delay` constraints, the Vivado tool examines the associated `create_clock` constraint to check for clock input jitter.

Slide 7-17:

set_input_delay and set_output_delay Constraints Account for Clock Delay

57622**slide

- `set_input_delay = input_clock - (T_data_In - T_clk_In)`
- `set_output_delay = input_clock - (T_data_Out + T_clk_Out)`



!*note

This example shows how the tools use the clock distribution delay to calculate the effective input and output delays.

Because the input data path and clock path are in parallel for input paths, the tools subtract the clock distribution delay from each input path. Likewise, because the output data path and the clock path are in series for output paths, the tools add the clock distribution delay to each output path.

Therefore, having a positive clock distribution delay helps your input times, but it hurts your output times. To compensate, the Kintex® devices include a delay-locked loop (DLL) to remove the clock distribution delays from your input and output paths. The DLL is very important for good chip-to-chip timing.

Remember that the DLL does not remove clock skew. The DLL only removes clock distribution delay. The global routing resources are designed to minimize clock skew, but they cannot remove it entirely.

Note that the data paths include the IOB, net, logic, and setup times or hold times associated with the registers.

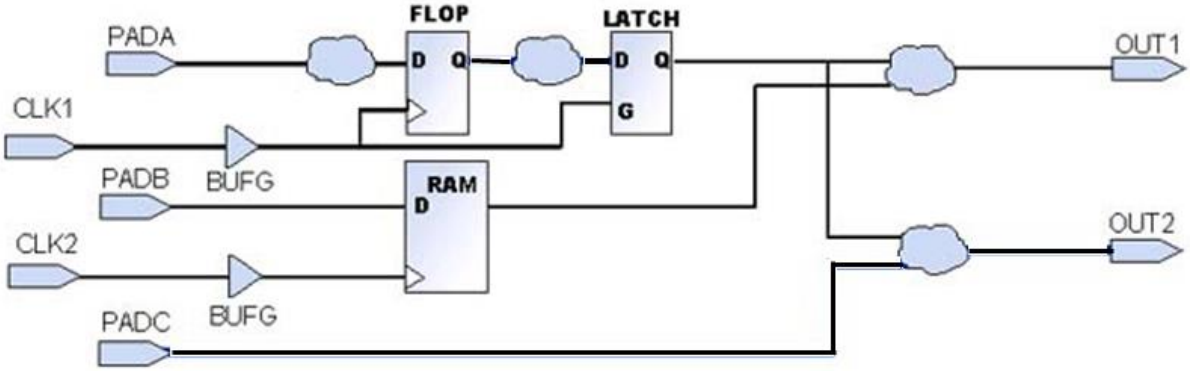
Also note that a PLL can also be added to condition the input clock or the generated DLL clock.

Slide 7-18:

Apply Your Knowledge

`57637**slide`

15) Which paths are constrained by a `create_clock` constraint on CLK2?



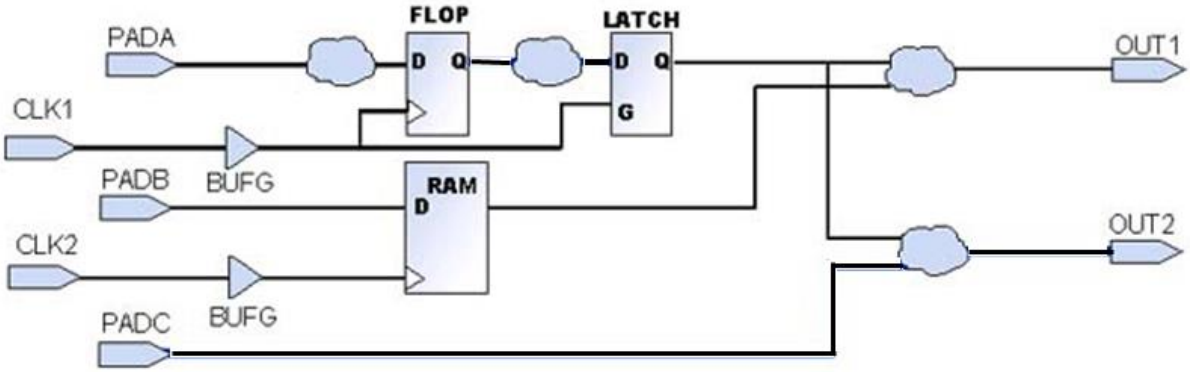
Slide 7-19:

Answer

`57623**slide`

16) Which paths are constrained by a `create_clock` constraint on CLK2?

– No paths are constrained



Slide 7-20:

Constraints Editor

57624**slide

- Overview
- Basic Timing Constraints
- **Constraints Editor**
- Summary

Slide 7-21:

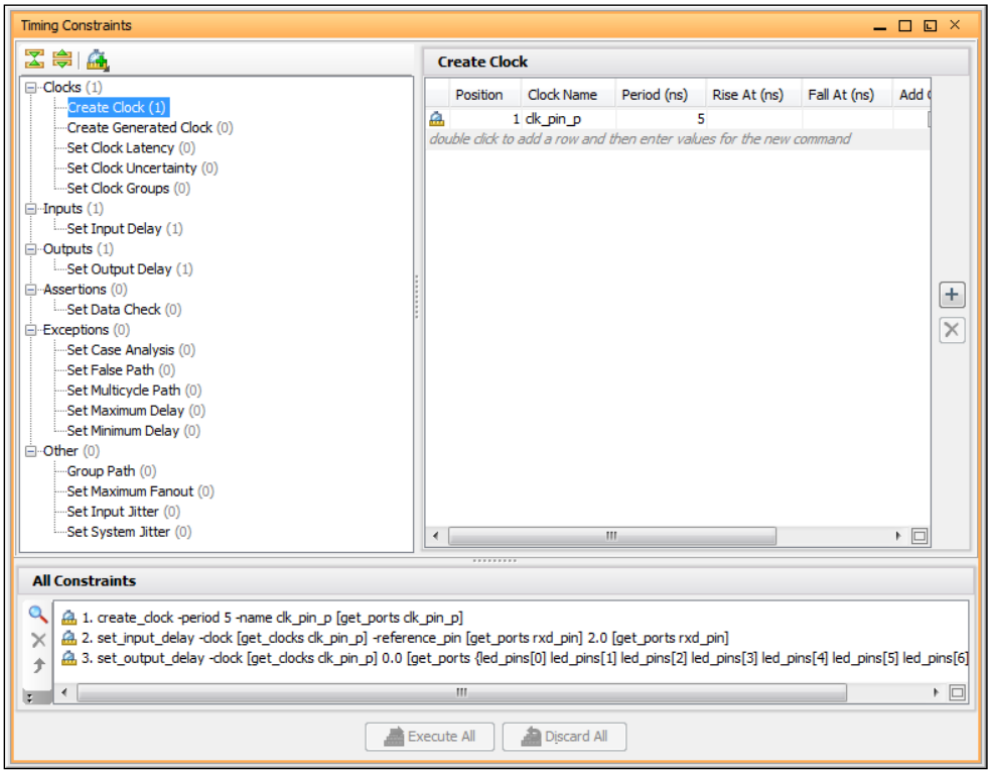
Launching the Timing Constraints Editor

57625**slide

- From the Flow Navigator, click the **Edit Timing Constraints** button

OR

- Select **Window > Timing Constraints** (available only if synthesized or implemented design is opened)

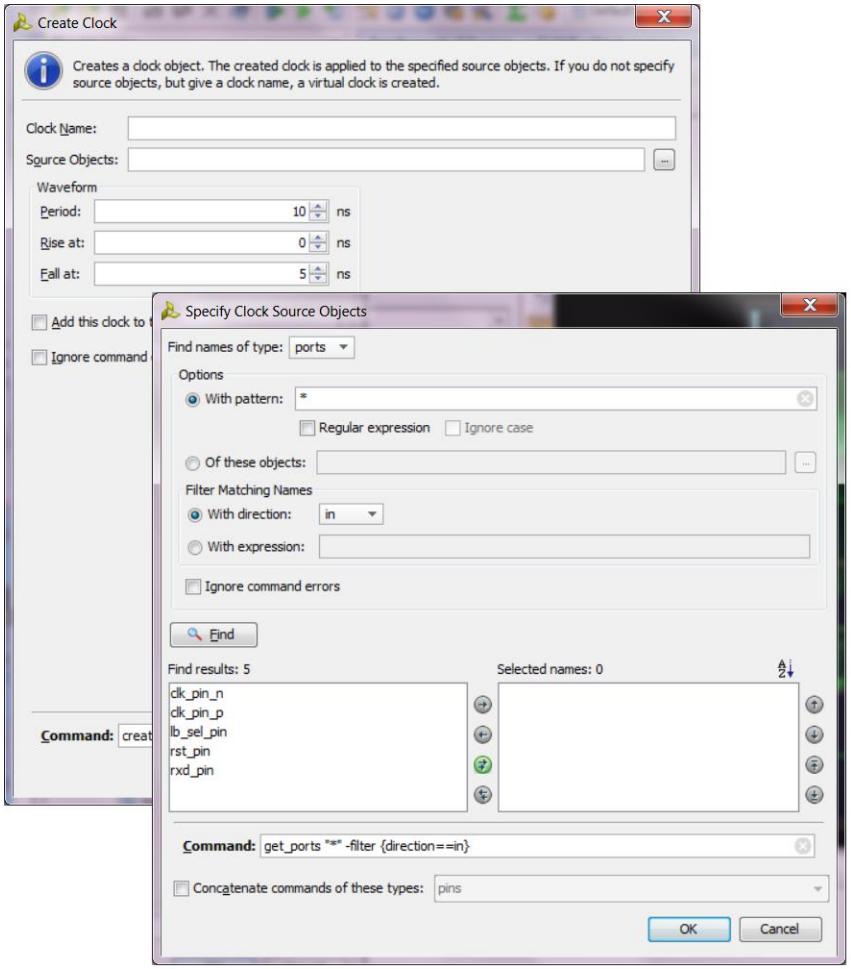


Slide 7-22:

Entering a create_clock Constraint

57626**slide

- Double-click the Create Clock button (see previous slide) and complete the Create Clock dialog box
  - Enter a constraint name
  - Use the Source Objects dialog box to search for clock nets in the design
  - Select ports with a direction of in and select the clock desired
  - Enter a period and a rise / fall time for the clock



57626!*note

Note that unlike the ISE® tools, until the project has been saved, the constraint will not exist in the XDC file. However, the constraint has been applied and is stored in memory. In fact, users can generate timing reports and use the `check_timing` command immediately. However, until the project has been saved (constraint stored in the XDC file) and the design resynthesized and reimplemented, the constraint will have no effect on the implementation solution found by the tools.

Designers can also create generated clocks that are derived from MMCMs.

Slide 7-23:

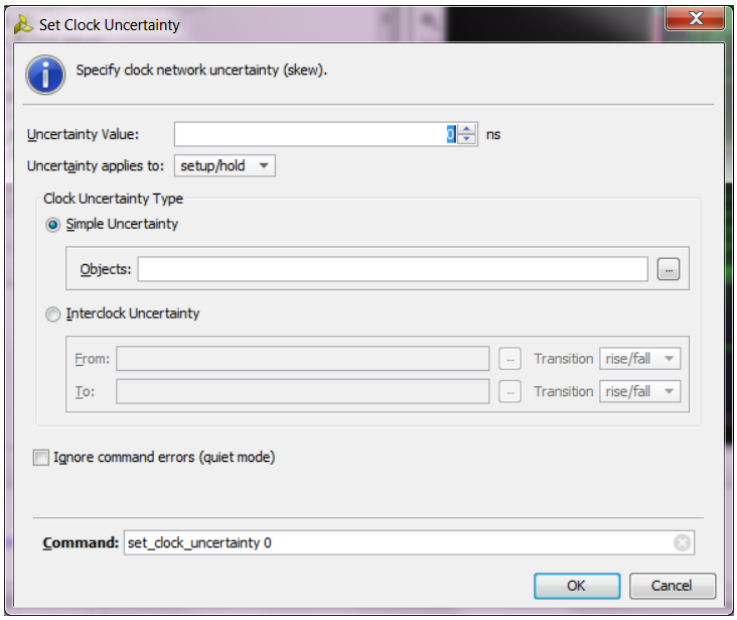
set_clock_uncertainty (Clock Skew)

57627**slide

- Several clock attributes can be set with the Constraints Editor directly in the Tcl Console

- set_clock_uncertainty
  - Clock network skew

- set_clock_latency



57627!*note

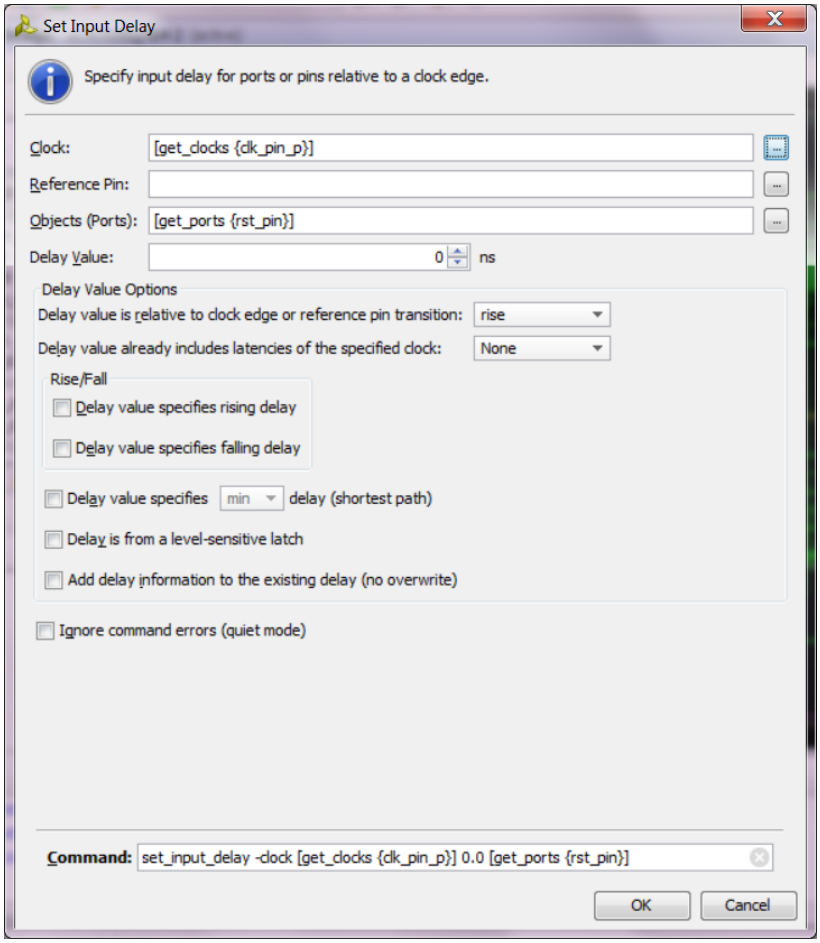
The clock network latency is the delay that a clock takes to propagate from the on-chip source to on-chip register clock pins.

Slide 7-24:

Entering set_ input_delay/set_output_delay Constraints

57628**slide

- Double-click the Set Input Delay or Set Output Delay buttons in the Constraints Viewer (see previous slide)
  - Find your clock port and input / output port
    - Note that ports refer to pins of a design (pins refer to ports of primitive)
    - Note that the Tcl syntax to enter the same constraint in the Tcl Console is put at the bottom of the GUI



57628!*note

Remember that these constraints will optimize all input paths and output paths to the selected clock domain.

This method creates port-specific delay constraints. If you wanted to add additional ports, you could select more from the GUI or enter the names in the Tcl syntax.

Slide 7-25:

Summary

57629**slide

- Overview
- Basic Timing Constraints
- Constraints Editor
- **Summary**

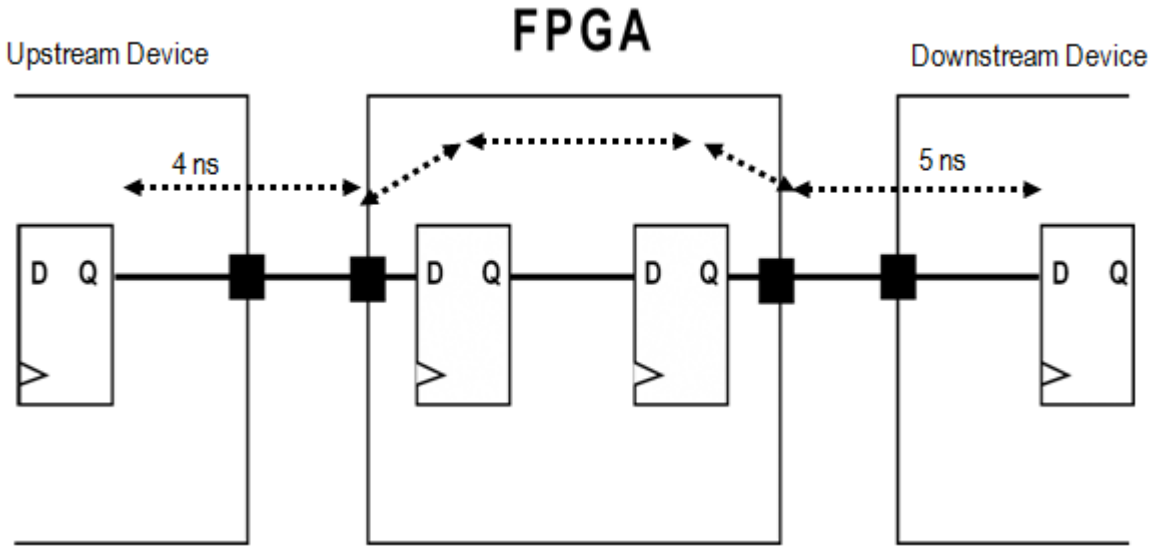
Slide 7-26:

Apply Your Knowledge

57631**slide

17) Given the system diagram below, what values would you put in the Constraints Editor so that the system will run at 100 MHz?

– Assume no clock skew between devices

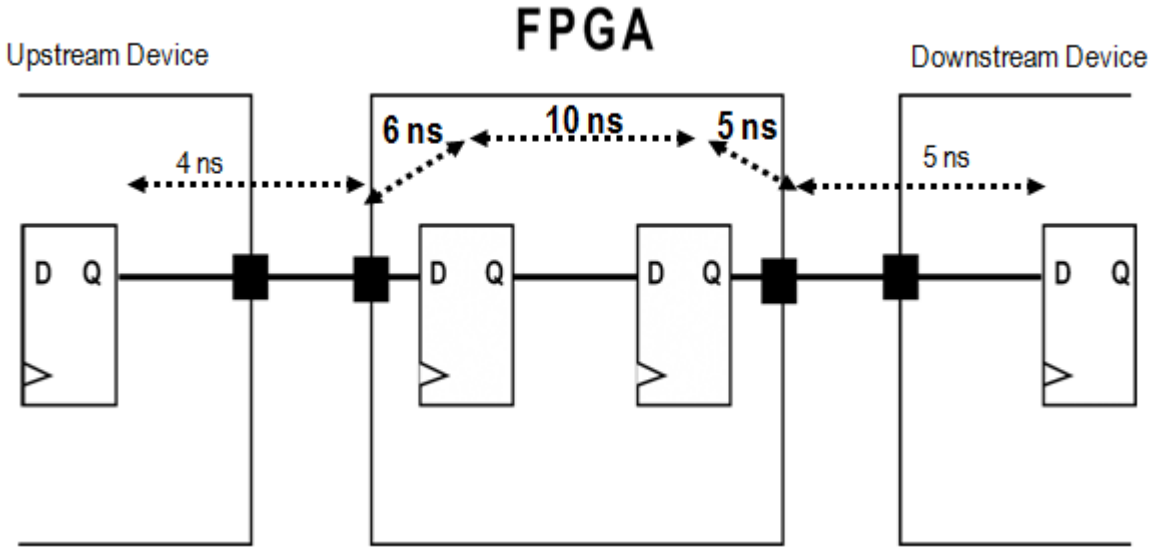


Slide 7-27:

Answer

57632**slide

18) Given the system diagram below, what values would you put in the Constraints Editor so that the system will run at 100 MHz?



- Answer: `input_clock = 10 ns`, `set_input_delay = 4 ns`, and `set_output_delay = 5 ns`

57632!*note

Because you want the entire system / board to run at 100 MHz (10 ns), the delays from one synchronous element to another synchronous element must be no more than 10 ns. Therefore, the `create_clock` constraint, which is from one synchronous element to another, will be 10 ns.

The `set_input_delay` and `set_output_delay` constraints must be calculated because the values are related to the upstream and downstream device.

For `set_input_delay`, you need the delay from the synchronous element in the upstream device to the synchronous element in the FPGA to be 10 ns. The delay through the synchronous element in the upstream device plus the trace delay is 4 ns, as shown in the figure. Hence, the `set_input_delay` can be no more than 4 ns.

A similar calculation is completed for the `set_output_delay` constraint.

Slide 7-28:

Summary

57633**slide

- Performance expectations are communicated with timing constraints
- The `create_clock` constraint covers delay paths between synchronous elements
- The `set_input_delay` and `set_output_delay` constraints cover delay paths from input pins to synchronous elements and from synchronous elements to output pins, respectively
- Use the Constraints Editor to create timing constraints

Slide 7-29:

Where Can I Learn More?

57634**slide

- On timing constraints: *Vivado Design Suite Using Constraints User Guide* (UG903)
  - Timing constraints
  - XDC syntax
  - Placement and area constraints
  - Tcl syntax for entry in Tcl Console
  - Constraint entry strategies

Capture Your Notes Here

Timing Constraints Wizard

Slide 8-1:

```
112072**slide

2019.1
112072!*note
```

This module discusses how the Timing Constraints Wizard in the Vivado® IDE helps you create and validate timing constraints. 15 minutes.

Slide 8-2:

Objectives

```
112073**slide
```

After completing this module, you will be able to:

- Use the Timing Constraints Wizard to create timing constraints
- Validate the completion of timing constraints using Timing Constraints Wizard

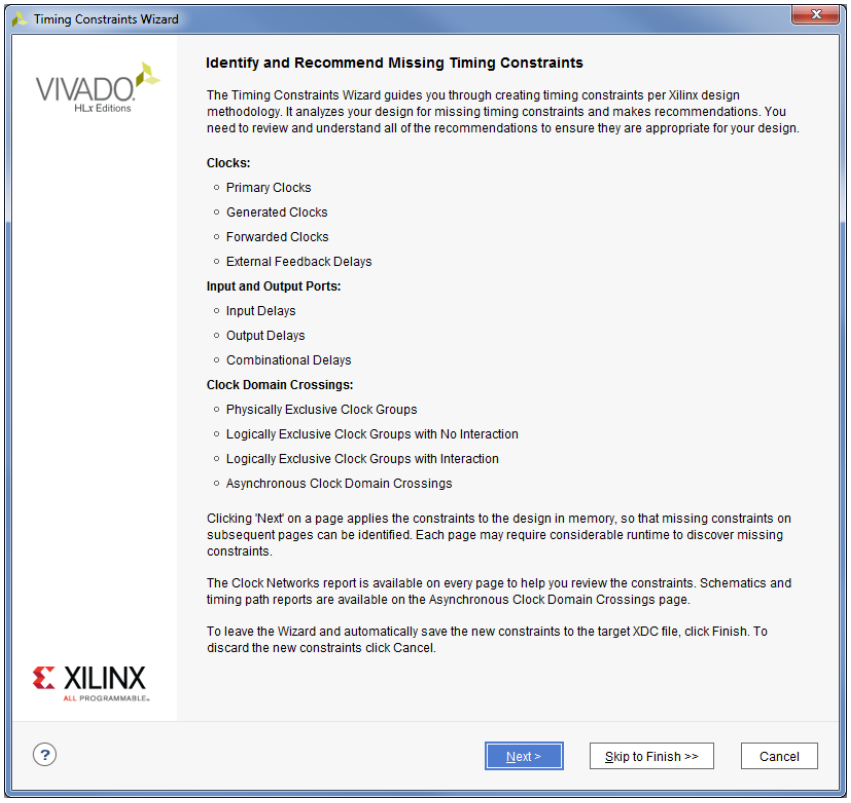
© Copyright 2019 Xilinx

Slide 8-3:

Timing Constraints Wizard

137059**slide

- The Timing Constraints Wizard is a tool to aid in writing and validating constraints
  - It simplifies constraint creation and validation



- Analyzes the gate-level netlist for missing timing constraints and recommends constraints based on recognized topologies
- Follows the UltraFast™ design methodology
- Useful to both novice and advanced XDC users
  - Completely constrains the design
  - Accuracy is the responsibility of the user

137059!*note

© Copyright 2019 Xilinx

The Timing Constraints Wizard identifies missing timing constraints on a synthesized or implemented design.

The timing constraints are divided into three categories:

- Clocks
- Input and Output Delays
- Clock Domain Crossings

© Copyright 2019 Xilinx

Slide 8-4:

Creating Complete and Accurate Constraints

112075**slide

- There are four key steps in creating constraints
  - Create clocks
  - Define clocks interactions
  - Set input and output delays
  - Set timing exceptions

- Constantly validate your constraints
  - Debug constraints post-synthesis
  - Revisit your constraints every design step
  - Verify that the Vivado IDE obeys your constraints

112075!*note

© Copyright 2019 Xilinx

Constraining the primary and generated clocks with a specified frequency and constraining the I/O interface with I/O delay values ensures that you have created complete and accurate timing constraints in your design. Any asynchronous clocks in your design will need to be defined as asynchronous clock groups so that the timing paths between these clock domains are analyzed.

The Vivado Timing Constraints Wizard enables you to create accurate and complete timing constraints by identifying where you may be missing timing constraints in your design.

The Timing Constraints Wizard tool identifies the following and helps you to constrain them in order to ensure that your design is completely constrained.

- Primary clocks
- Generated clocks in the fabric
- Forwarded clocks
- External delays
- Input delay constraints
- Output delay constraints
- Constraints for combinatorial paths
- Clock Group Constraints
- Physically and logically exclusive
- Asynchronous clock group

© Copyright 2019 Xilinx

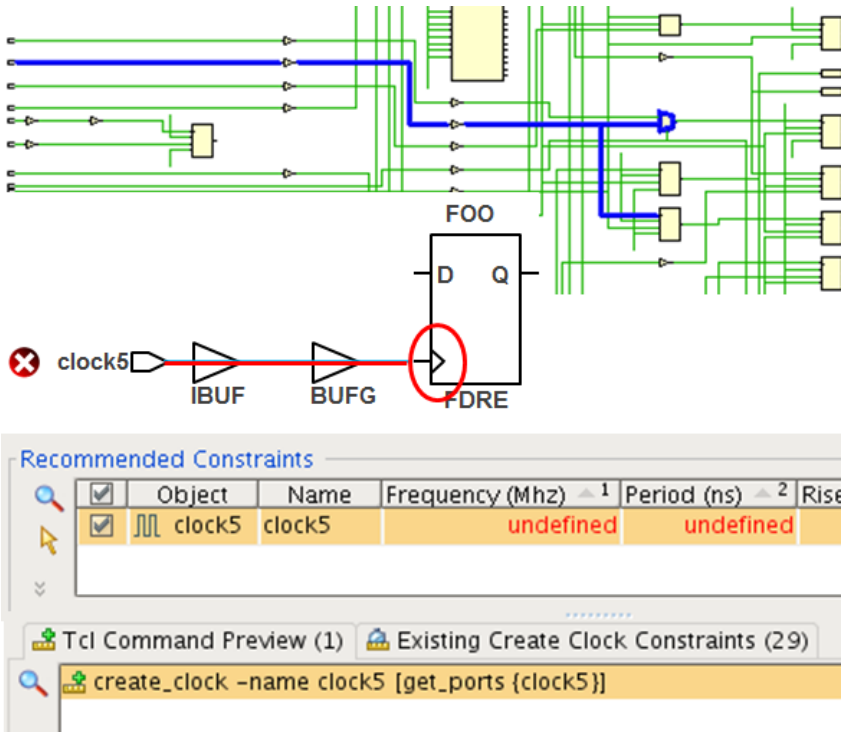
Slide 8-5:

Missing Primary Clocks

112076**slide

- Primary clocks usually enter the design through input ports

- They must be defined with a period in order to ensure that all the elements (flip-flops, block RAMs, and DSPs) that are clocked by primary clocks are analyzed for timing



112076!*note

© Copyright 2019 Xilinx

Constraining the primary clocks helps you understand whether the design meets timing or not, which determines whether the design is working with your primary clock frequency or not.

The Vivado Timing Constraints Wizard identifies the missing primary clocks in your design and helps you to constrain them.

clock5 shown here is a primary clock whose period is undefined. The constraints wizard identifies and reports this.

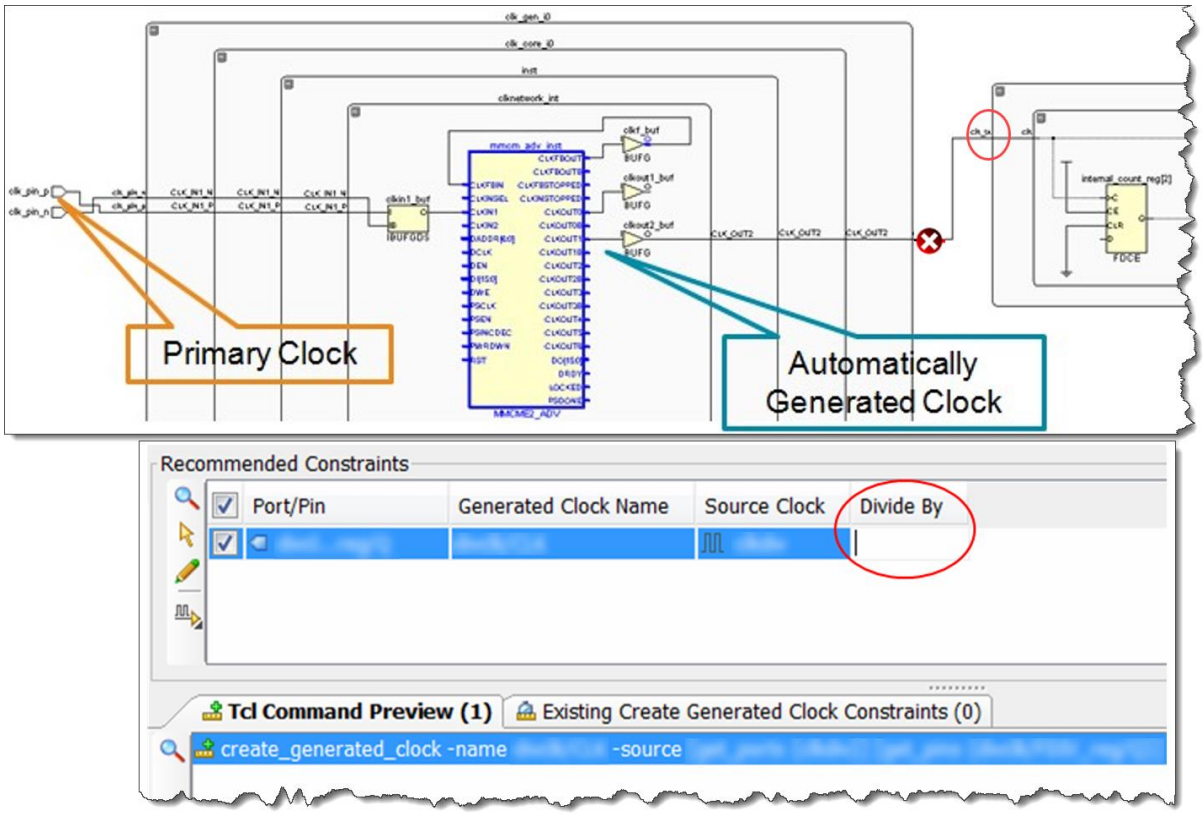
© Copyright 2019 Xilinx

Slide 8-6:

Missing Generated Clocks in the Fabric

112077**slide

- Generated clocks are derived from master clocks



112077!*note

The Vivado Design Suite automatically derives generated clocks for all Clock Modifying Blocks such as MMCM/PLL, clock buffers, and UltraScale™ device GTs.

This Constraints Wizard identifies missing generated clocks on user logic only, such as when creating a clock divider using user logic instead of using Clock Modifying Blocks.

As shown here, the primary clock clk_pin_p goes to MMCM and clk_OUT2 is the automatically generated clock that comes out. The relationship between the primary and generated clock is undefined (shown by the empty Divide By column).

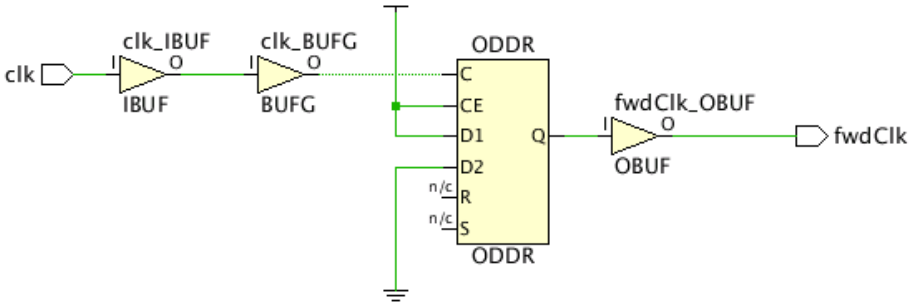
© Copyright 2019 Xilinx

Slide 8-7:

Forwarded Clocks

137089**slide

- Forwarded clocks are the generated clocks defined on output ports that replicate internal clocks
- A forwarded clock is usually a time reference for other output ports to form a source synchronous interface



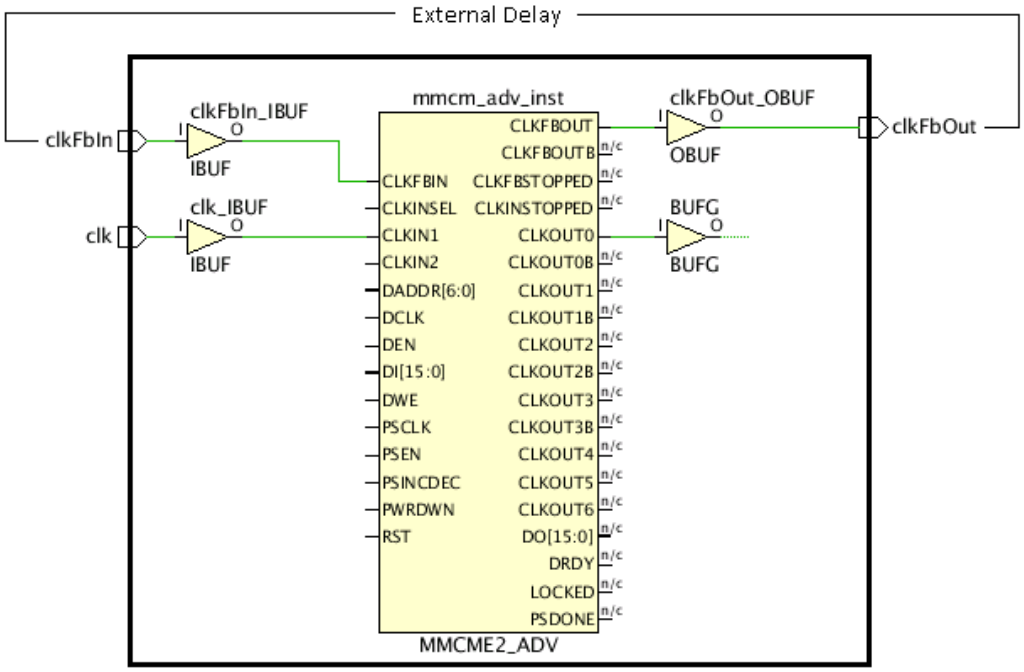
Slide 8-8:

External Feedback Delays

137090**slide

© Copyright 2019 Xilinx

- External feedback delays are the board delays between pairs of device input and output ports which are used in a MMCM or PLL feedback loops
- Both min and max delay values must be specified



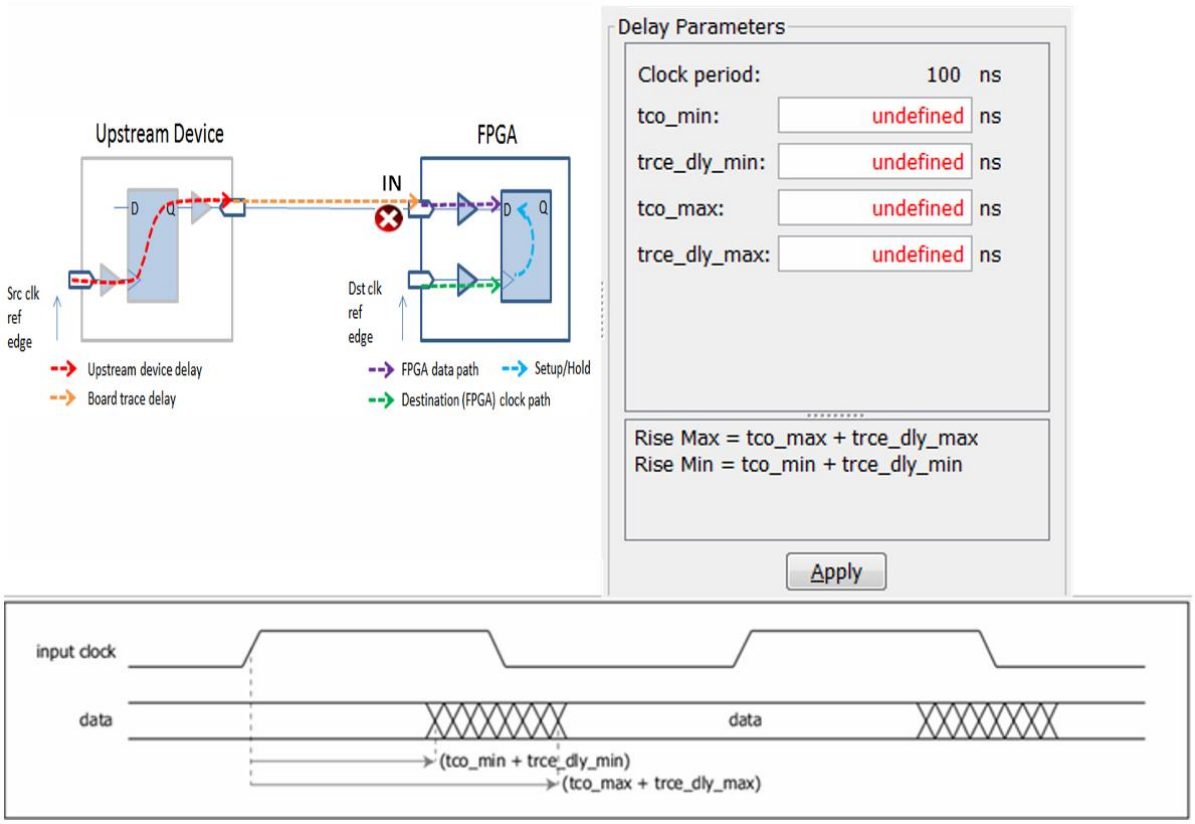
© Copyright 2019 Xilinx

Slide 8-9:

Input Constraints

112078**slide

- Input constraints correspond to the input delay between the flip-flop at the upstream device to the flip-flop in the FPGA



112078!*note

Input delays describe the relationship between an input signal and its reference clock at the boundary of the FPGA. This relationship is used by the timing engine to determine if the input arrives with sufficient margin to be captured correctly.

The Timing Constraints Wizard identifies missing input delay constraints and helps you to constrain it with a proper input delay value that is determined by maximum/minimum board trace delay values and maximum/minimum clk-to-out delay values of launching flip-flop.

© Copyright 2019 Xilinx

The input delay constraint is calculated as follows:

The set input delay value for the maximum delay (setup analysis) = Tco_max + maximum board trace delay (trce_dly_max).

The set input delay value for the minimum delay (hold analysis) = Tco_min + minimum board trace delay (trce_dly_min).

As per the example given in the slide, an input IN which is dependent on Dst clk has missing input constraints, which are shown by the red-and-white "X."

An input delay constraint for this would look like:

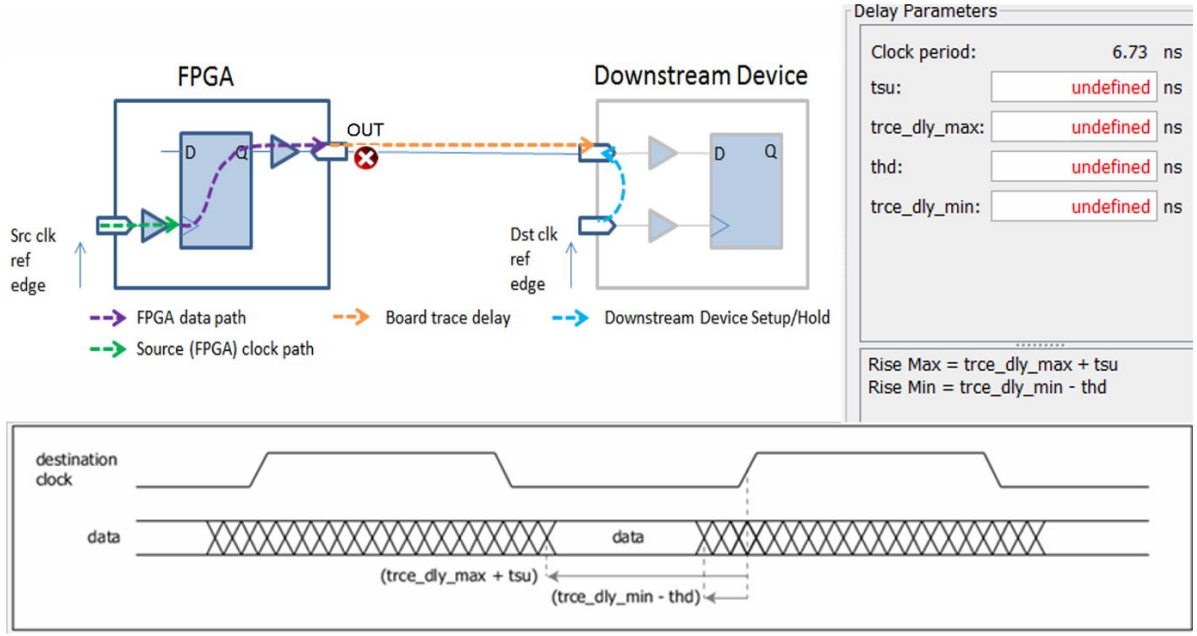
```
set_input_delay -clock [get_clocks {Dst clk}] -max -add_delay 0.1
[get_ports {IN}]
```

Slide 8-10:

Output Constraints

112079**slide

- Output constraints correspond to the output delay between the flip-flop in the FPGA to the flip-flop at the downstream device



112079!*note

© Copyright 2019 Xilinx

Output delays describe the relationship between an output signal and its launching clock inside the FPGA. This relationship is used by the timing engine to determine if the output signal will be captured correctly at the receiver or the downstream device.

The Timing Constraints Wizard identifies missing output delay constraints and helps you to constrain them with a proper output delay value that is determined by the maximum/minimum board trace delay values and setup/hold requirement values of the capturing flip-flop at the downstream device.

The output delay constraint will calculated as follows:

The set output delay value for the maximum delay (setup analysis) = maximum board trace delay (trce_dly_max) + setup requirement of the destination flip-flop (Tsu).

The set output delay value for the minimum delay (hold analysis) = minimum board trace delay (trce_dly_min) - hold requirement of the destination flip-flop (Th).

As per the example given in the slide, an output OUT which is dependent on Src clk has missing output constraints which are shown by the red-and-white "X."

An output delay constraint for this would look like:

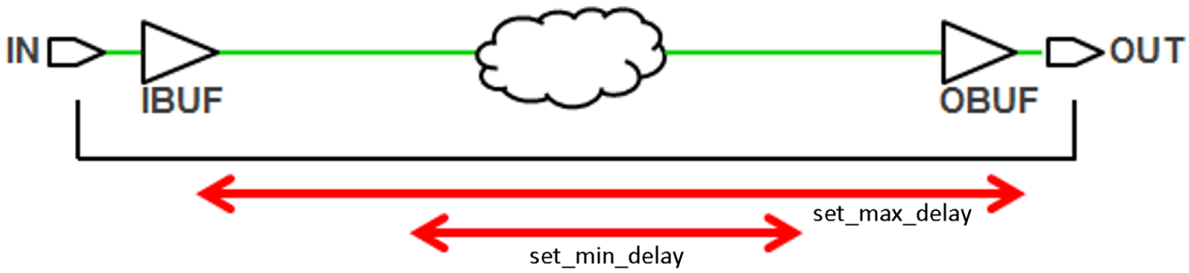
```
set_output_delay -clock [get_clocks {Src clk}] -max -add_delay
0.2 [get_ports {OUT}]
```

Slide 8-11:

Combinational Delays

112081**slide

- Combinational delays are paths that enter and exit the FPGA without being captured by any sequential elements



© Copyright 2019 Xilinx

112081!*note

These combinational paths can be constrained with the set_max_delay and set_min_delay constraints.

Below is an example of a delay which constrains a combinational path:

Slowest: `set_max_delay 5.0 -from [get_ports {IN}] -to [get_ports{OUT}]`

Fastest: `set_min_delay 2.0 -from [get_ports {IN}] -to [get_ports {OUT}]`

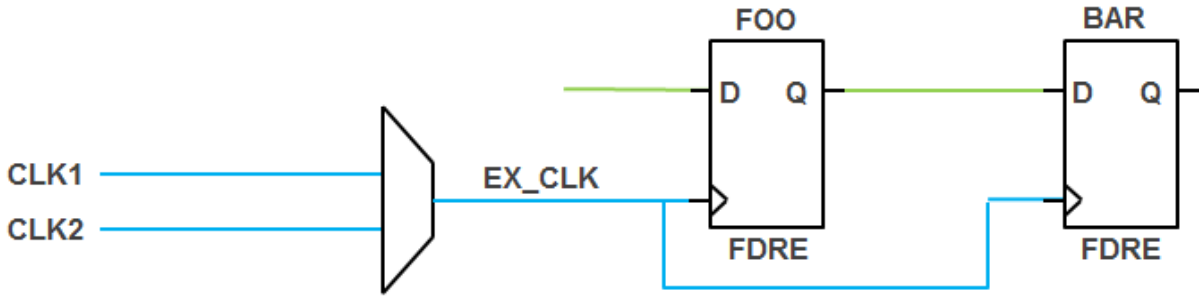
© Copyright 2019 Xilinx

Slide 8-12:

Exclusive Clocks

112080**slide

- Two types of exclusive clocks
  - Physically exclusive clock groups
    - Clocks that are defined on the same source point and propagate on the same clock tree
  - Logically exclusive clock groups
    - Clocks that are defined on different source points but share part of their clock tree due to a multiplexer or other combinational logic



112080!*note

Physically exclusive clocks are clocks that do not exist in the design at the same time.

In the example, the CLK1 and CLK2 clocks are logically exclusive, so a logically exclusive clock group is defined to prevent analysis between the two clocks.

```
set_clock_groups -logically_exclusive -group CLK1 -group CLK2
```

Slide 8-13:

Summary

121298**slide

© Copyright 2019 Xilinx

- The Timing Constraints Wizard simplifies constraint creation and validation
- The Timing Constraints Wizard identifies the missing primary clocks and generated clocks and helps to constrain them
- The Timing Constraints Wizard identifies missing input and output delay constraints and helps to constrain them

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Timing Constraints Editor

Slide 9-1:

108339**slide

2019.2
108339!*note

This module describes the process of applying basic timing constraints using the Constraints Editor. 5 minutes.

Slide 9-2:

Objectives

108340**slide

After completing this module, you will be able to:

- Apply basic timing constraints to your design through the Constraint Editor

© Copyright 2019 Xilinx

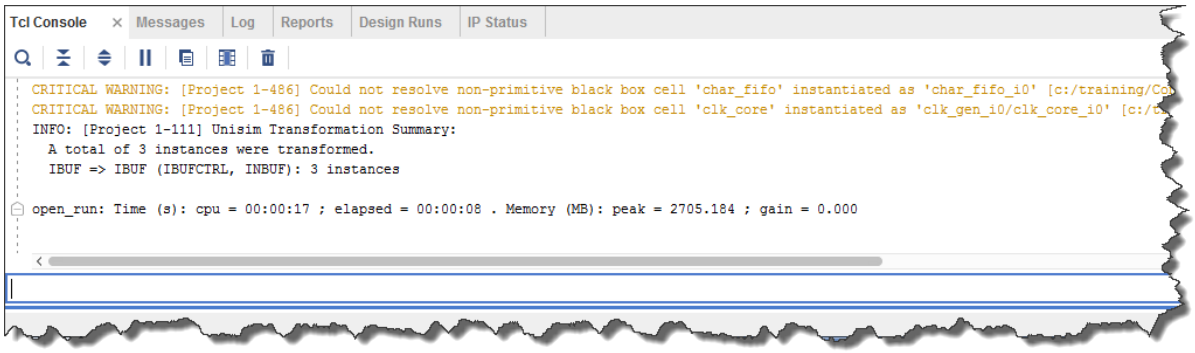
Slide 9-3:

Timing Constraint Entry

108337**slide

- The Constraints Editor lets you to specify
  - Global timing constraints
  - Groups on which to apply constraints
  - Input timing constraints
  - Output timing constraints
  - Timing exceptions for paths and nets

- Tcl Console
  - All constraints can be entered by Tcl commands
  - The proper syntax for XDC constraints must be used



www.xilinx.com
© Copyright 2019 Xilinx

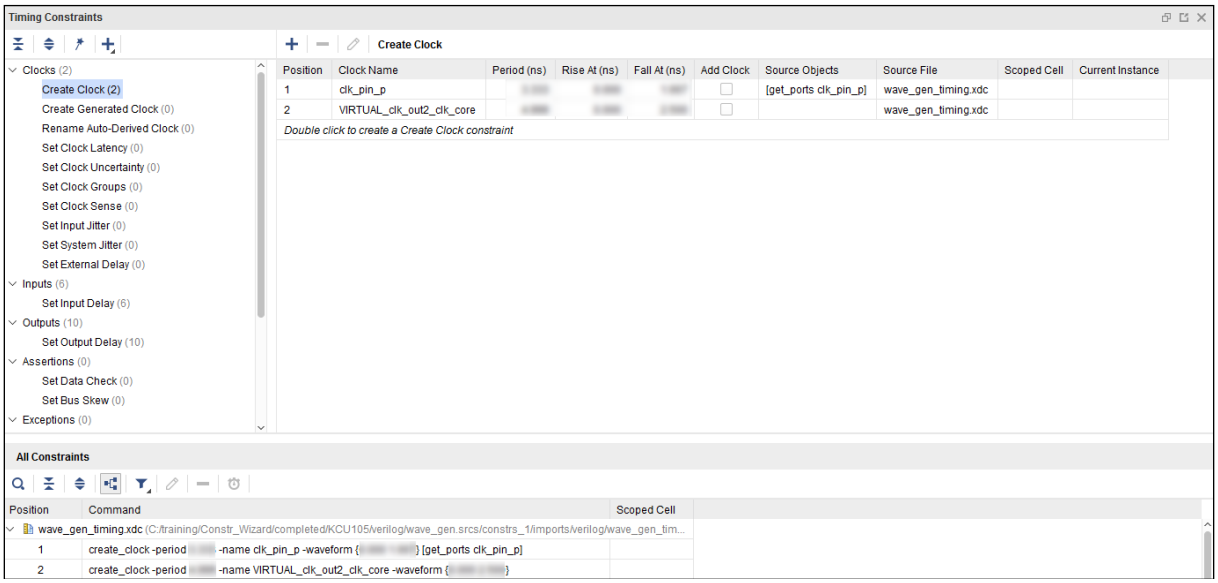
Slide 9-4:

Launching the Timing Constraints Editor

121391**slide

- To launch the Constraints Editor, click the **Edit Timing Constraints** button from the Flow Navigator

- This can also be done by selecting **Tools** > **Edit Timing Constraints**
  - This option is available only if the synthesized or implemented design is opened



121391!*note

`create_clock`: Creates a clock object with the specified period or waveform defined in nanoseconds (ns). This command defines primary clocks which are used by the timing engine as the delay propagation starting point of any clock edge.

`create_generated_clock`: Generates a new clock object from an existing physical clock object in the design.

Auto-Derived Clocks: Most generated clocks are automatically derived by the Vivado Design Suite timing engine which recognizes the Clock Modifying Blocks (CMB) and the transformation they perform on the master clocks.

© Copyright 2019 Xilinx

`set_input_delay`: Specifies the external system-level path delay on a primary input port relative to a clock edge at the interface of the design.

`set_output_delay`: Specifies the external system-level path delay on a primary output port relative to a clock edge at the interface of the design.

`set_data_check`: Performs a setup and hold check for a data pin with respect to another data pin.

`set_bus_skew`: Sets the bus skew requirement on bus signals that cross clock domains. The bus skew constraint defines the maximum spread between the fastest and slowest signals of the bus.

`set_case_analysis`: Specifies that a pin or port is in a steady state of 1, 0, rising, or falling.

`set_false_path`: Sets false timing paths in the design that are ignored during timing analysis.

`set_multicycle_path`: Lets you choose a path multiplier, N, to establish a timing path that takes N clock cycles from the start clock edge to the capture clock edge.

`group_path`: Groups a set of paths for cost function calculations, primarily for timing analysis.

`set_disable_timing`: Disables timing arcs within a specified cell or cells that lead to the output pins of the cell.

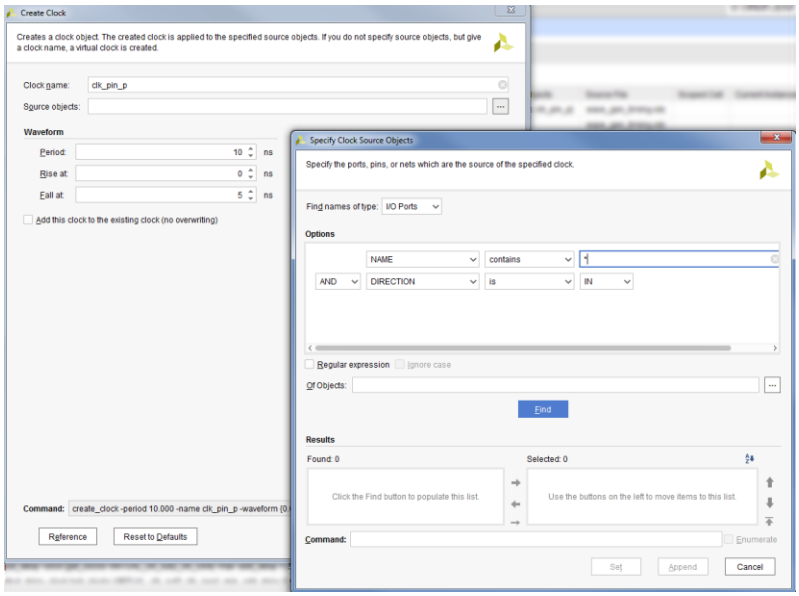
Slide 9-5:

Entering a create_clock Constraint

108334**slide

© Copyright 2019 Xilinx

- Double-click the **Create Clock** button and complete the Create Clock dialog box
  - Enter a constraint name
  - Use the Source Objects dialog box to search for clock nets in the design
  - Select ports with a direction of in and select the desired clock
  - Enter a period and a rise/fall time for the clock



© Copyright 2019 Xilinx

108334!*note

Note that unlike the ISE® tools, until the project has been saved, the constraint will not exist in the XDC file. However, the constraint has been applied and is stored in memory.

In fact, users can generate timing reports and use the `check_timing` command immediately. Until the project has been saved (the constraint stored in the XDC file) and the design resynthesized and reimplemented, however, the constraint will have no effect on the implementation solution found by the tools.

Designers can also create generated clocks that are derived from MMCMs.

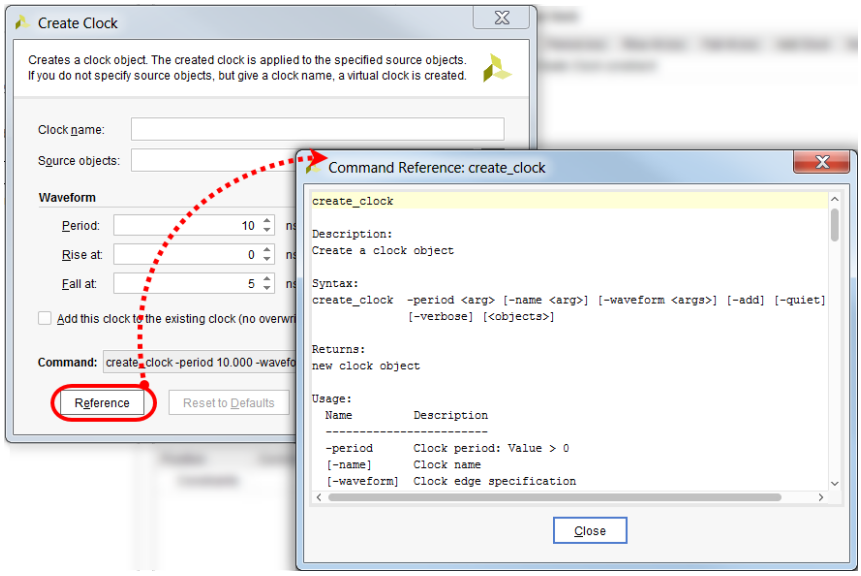
Slide 9-6:

Reference Option

135112**slide

© Copyright 2019 Xilinx

- Click the **Reference** button to get more information for the respective command



- It gives information such as
  - Description
  - Syntax
  - Returns
  - Usage
  - Categories
  - Arguments
  - Examples

135112!*note

The Reference option is same as the `help` Tcl command.

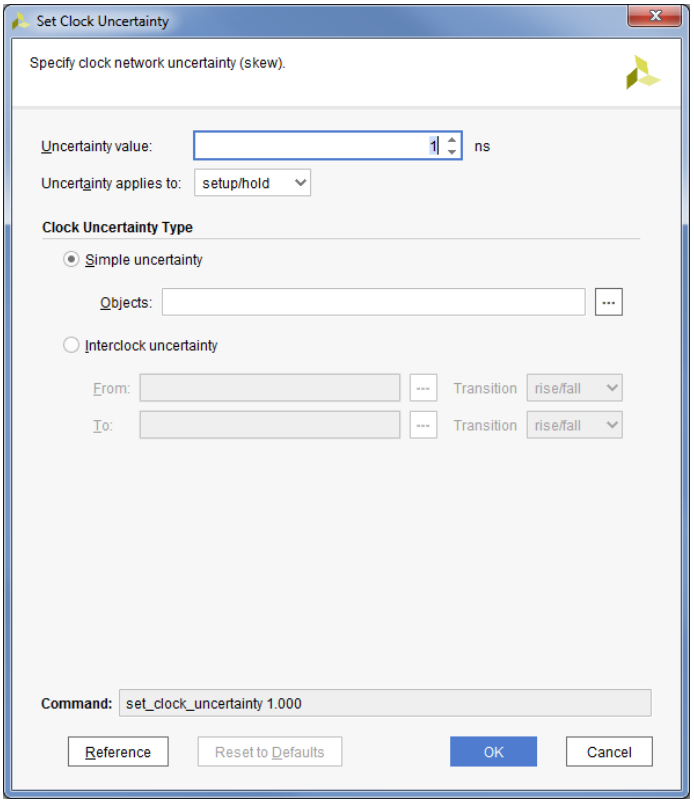
© Copyright 2019 Xilinx

Slide 9-7:

Other Clock Constraints

108335**slide

- Several clock attributes can be set with the Constraints Editor
- set_clock_uncertainty
  - Clock network skew
- set_clock_latency
- create_generated_clock



108335!*note

The clock network latency is the delay that a clock takes to propagate from the on-chip source to on-chip register clock pins.

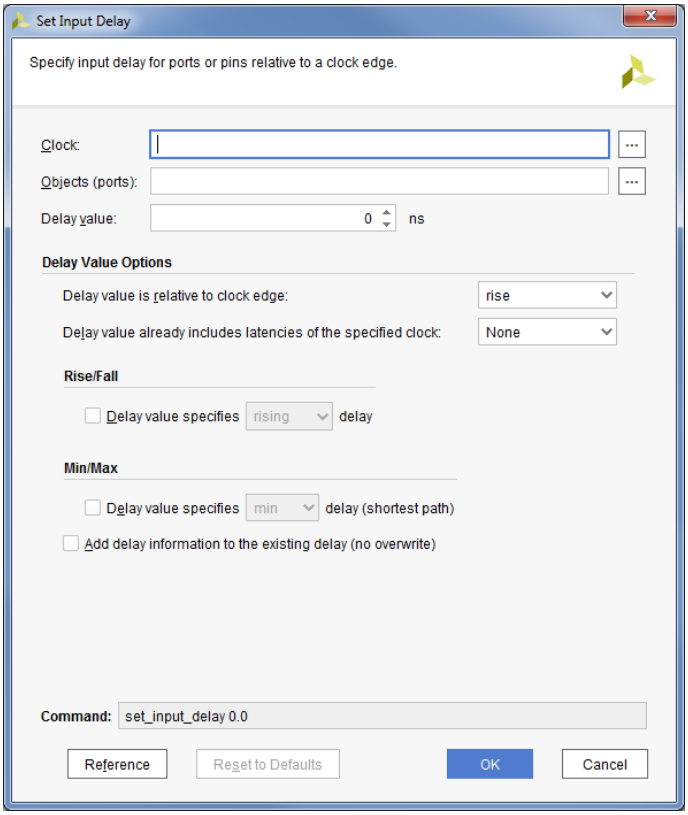
© Copyright 2019 Xilinx

Slide 9-8:

Entering set_ input_delay/set_output_delay Constraints

108336**slide

- Double-click the **Set Input Delay** or **Set Output Delay** buttons in the Constraints Viewer
  - Find your clock port and input/output port
    - Note that ports refer to the pins of a design (pins refer to the ports of a primitive)
    - Note that the Tcl syntax to enter the same constraint in the Tcl Console is shown at the bottom of the GUI



- Select the clock and object for which delay needs to be specified

- You also have the option to apply the setup and hold constraint separately

108336!*note

Remember that these constraints will optimize all input paths and output paths to the selected clock domain.

This method creates port-specific delay constraints. If you wanted to add additional ports, you could select more from the GUI or enter the names in the Tcl syntax.

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Timing Summary Report

Slide 10-1:

113050**slide

2019.2
113050!*note

This module introduces the Timing Summary Report in the Vivado® IDE. 10 minutes.

Slide 10-2:

Objectives

113062**slide

After completing this module, you will be able to:

- Describe the complete timing status of the paths in a design
- Use the `report_timing_summary` command

Slide 10-3:

Timing Summary Report

113057**slide

- Single command/report that provides a complete "sign-off" report on the timing of your design
  - Report contains several sections for complete analysis
  - Available in the flow after synthesis
  - Equivalent Tcl command: `report_timing_summary`

© Copyright 2019 Xilinx

- Design Timing Summary gives you a quick overview of timing status

113057!*note

The Timing Summary Report can be opened in both the synthesized design and implemented design:

- In a synthesized design, the timing engine estimates the net delay based on fanout and connectivity.
- In an implemented design, the timing engine estimates net delays based on actual routing information.

The Timing Summary Report or `report_timing_summary` command reports timing analysis done for the design, asynchronous paths, WNS, WHS values, constraint coverage, etc.

The report also allows you to analyze the paths in the design and optimize critical paths to meet timing.

Slide 10-4:

Timing Summary Report Sections

113063**slide

- General Information section
  - Displays design name, target device, package, speed grade, equivalent Tcl command
- Timer Settings section
  - Details timing engine settings used to generate the timing information
- Design Summary section
  - Helps you understand if the design has met timing requirements
  - Design Timing Summary includes

© Copyright 2019 Xilinx

- Worst and total negative slack (WNS/TNS)
- Worst and total hold slack (WHS/THS)
- Failing endpoints

© Copyright 2019 Xilinx

Slide 10-5:

Worse Timing Path

113051**slide

- In the timing path summary verify
  - Slack: Large or small?
  - Requirement: Reasonable or not, expected value?
  - Source and destination clocks: Same clock or different clocks?
  - Datapath delay: Large or small relative to the requirement?
  - Levels of logic: Large or small?
  - Skew and uncertainty: large or small?

Slide 10-6:

© Copyright 2019 Xilinx

Check Timing Report Sections

121315**slide

121315!*note

© Copyright 2019 Xilinx

Check Timing: Contains the same information that is produced by the `check_timing` command, which summarizes potential timing issues.

Description

Checks the design elements of ports, pins, and paths against the current timing constraints. Use this command to identify possible problems with design data and timing constraints before running the `report_timing` command. The `check_timing` command runs a series of default timing checks and reports a summary of any violations found. To get detailed information about violations, use the `-verbose` option.

Default Timing Checks

- `generated_clocks`: Checks that derived clocks defined by the `create_generated_clock` command reference primary clocks, defined by the `create_clock` command, rather than referencing another generated clock.
- `loops`: Checks for and warns of combinational feedback loops in the design.
- `multiple_clock`: Warns if multiple clocks reach a register clock pin. If more than one clock signal reaches a register clock pin, it is unclear which clock will be used for analysis. In this case, use the `set_case_analysis` command so that only one clock will propagate to the register clock pin.
- `no_clock`: Reports unclocked registers. In this case, no setup or hold checks are performed on data pins related to the register clock pin.
- `no_input_delay`: Reports the input ports without an input delay constraint. Input delays can be assigned with the `set_input_delay` command. Input ports that are unclocked will not be checked for input delays.
- `no_output_delay`: Reports the output ports without an output delay constraint. Output delays can be assigned with the `set_output_delay` command. Output ports that are unclocked will not be checked for output delays.
- `partial_input_delay`: Reports the input ports having partially defined input delay constraints. Assigning `set_input_delay -max` or `set_input_delay -min` to an input port without assigning the other creates a partially defined input delay. In such cases, paths starting from the input port may become unconstrained and no timing checks will be done against the port. Assigning `set_input_delay` without specifying either `-min` or `-max` allows the tool to assume both min and max delays and so does not result in a partial input delay. Note that unclocked input ports are not checked for partial input delays.

© Copyright 2019 Xilinx

- `partial_output_delay`: Reports the output ports having partially defined output delay constraints. Assigning `set_output_delay -max` or `set_output_delay -min` to an output port without assigning the other creates a partially defined output delay. In such cases, paths reaching the port may become unconstrained and no timing checks will be done against the port. Assigning `set_output_delay` without specifying either `-min` or `-max` allows the tool to assume both min and max delays, and so does not result in a partial output delay. Note that unclocked output ports are not checked for partial output delays.
- `unconstrained_endpoints`: This warning identifies timing path endpoints that are not constrained. Endpoints at register data pins are constrained by clock assignment with the `create_clock` command. Endpoints at output ports are constrained by output delays with the `set_output_delay` or `set_max_delay` commands.
- `unexpandable_clocks`: Reports clock sets in which the period is not expandable with respect to each other, when there is at least one path between the clock sets. A clock is unexpandable if no common multiples are found within 1000 cycles between the source and destination clocks.

Slide 10-7:

Timing Summary Report Sections

`113066**slide`

- Intra-Clock paths summarizes
  - Worst slack and total violations of the timing path with the same source and destination clocks
- Inter-Clock paths summarizes
  - Worst slack and total violations of the timing path with different source and destination clocks
- Other Path groups
  - Default path groups and user-defined path groups created by the `group_path` command

© Copyright 2019 Xilinx

- Unconstrained Paths
  - Details paths that do not have timing constraints or timing exception paths

113066!*note

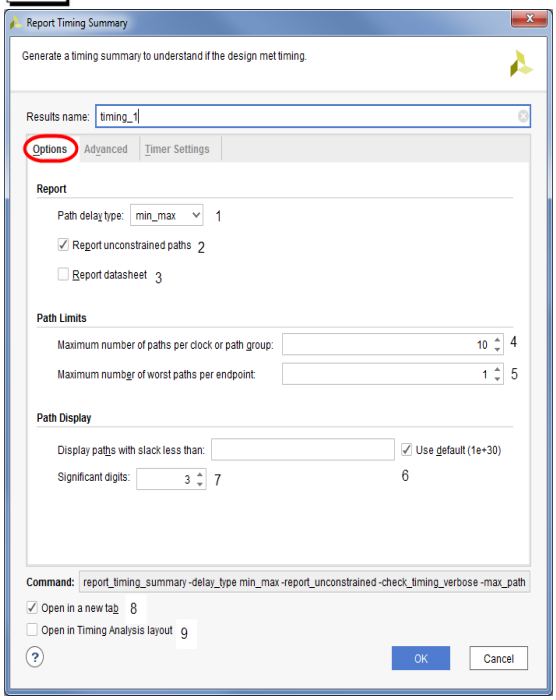
© Copyright 2019 Xilinx

The **async_default** path group is automatically created and includes all clock paths ending with an asynchronous timing check.

Slide 10-8:

Report Timing Summary (Options)

129116**slide



Option	Description
1 [-delay_type]	Type of path delay Values: max, min and min_max
2 [-report_unconstrained]	Report unconstrained path Default: Enabled
3 [-report_datasheet]	Reports the datasheet and added to summary Default: Disable
4 [-max_paths]	Maximum number of paths to report per clock group or path group Default: 1
5 [-nworst]	Lists up to N worst paths per endpoints Default:1
6 [-slack_lesser_than]	Displays paths with less than this value Default: 1e+30
7 [-significant_digits]	Number of digits to display, Range: 0 to 13 Default:
8 [-name]	Output results to GUI panel: name Default: timing_<number>
9	Opens the timing summary report in a Timing Analysis report with number in the bracket

129116!*note

Note: The Tcl command is shown in the Command box for the selected options.

Path Delay Type (What type of analysis would you like?)

- `-delay_type` arg (Optional): Specifies the type of delay to analyze when running the timing report. The valid values are `min`, `max`, and `min_max`. The default setting for `-delay_type` is `min_max`.

© Copyright 2019 Xilinx

Report Unconstrained Paths (Do you want to include unconstrained paths in your report?)

- `-report_unconstrained` (Optional): Report timing on unconstrained paths in the current design. As a default, the `report_timing_summary` command will not include unconstrained timing paths.

Path Limits (How many total paths and max paths/endpoint do you want in your report?)

- `-max_paths` arg (Optional): The maximum number of paths to report per clock or per path group. This is specified as a value greater than or equal to 1.
- `-nworst` arg (Optional): The number of timing paths to output in the timing report. The timing report will return the N worst paths to endpoints based on the specified value, greater than or equal to 1. The default setting is 1.

Path Display (Do you want to specify a range of slack values of specific interest?)

- `-slack_lesser_than` arg (Optional): Report timing on paths with a calculated slack value less than the specified value. Used with `-slack_greater_than` to provide a range of slack values of specific interest.
- `-significant_digits` arg (Optional): The number of significant digits in the output results. The valid range is 0 to 13. The default setting is three significant digits.

Open in a New Tab

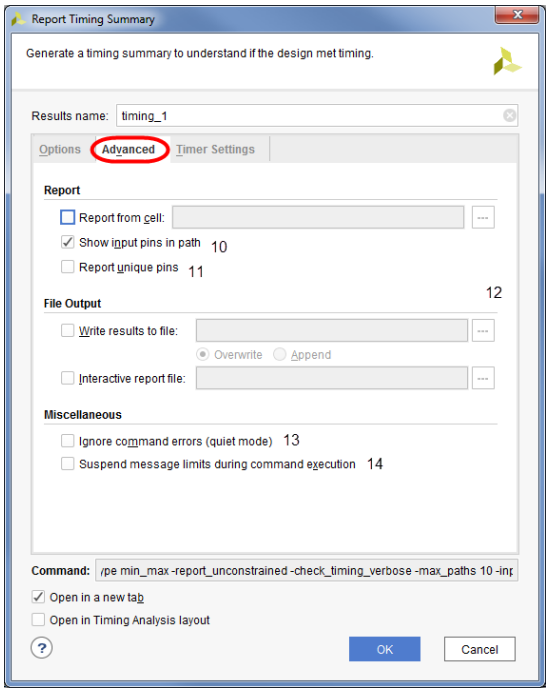
- `-name` arg (Optional): Specifies the name of the results set for the GUI. Timing summary reports in the GUI can be deleted with the `delete_timing_results` command.

Slide 10-9:

Report Timing Summary (Advanced)

`129115**slide`

© Copyright 2019 Xilinx



Option	Description
10 [-input_pins]	Show input pins in the path Default: Enabled
11 [-unique_pins]	Report timing paths only through unique set of pins, reporting one path per group Default: Disabled
12 [-file]	Write output results to a file
13 [-quiet]	Ignore command errors
14 [-verbose]	Suspend message limit during command execution

129115!*note

© Copyright 2019 Xilinx

Pins and Nets

- `-input_pins` (Optional): Show input pins in the timing path report. For use with `-path_type full`, `full_clock`, and `full_clock_expanded`.
  `-unique_pins` (Optional): Only report timing paths through each unique set of pins, reporting one path per path group.

File Output

- `-file` arg (Optional): Write the report into the specified file. Note that if the path is not specified as part of the file name, the file will be written into the current working directory, or the directory from which the tool was launched.
- `-append` (Optional): Append the output of the command to the specified file rather than overwriting it. Note that the `-append` option can only be used with the `-file` option.

Miscellaneous

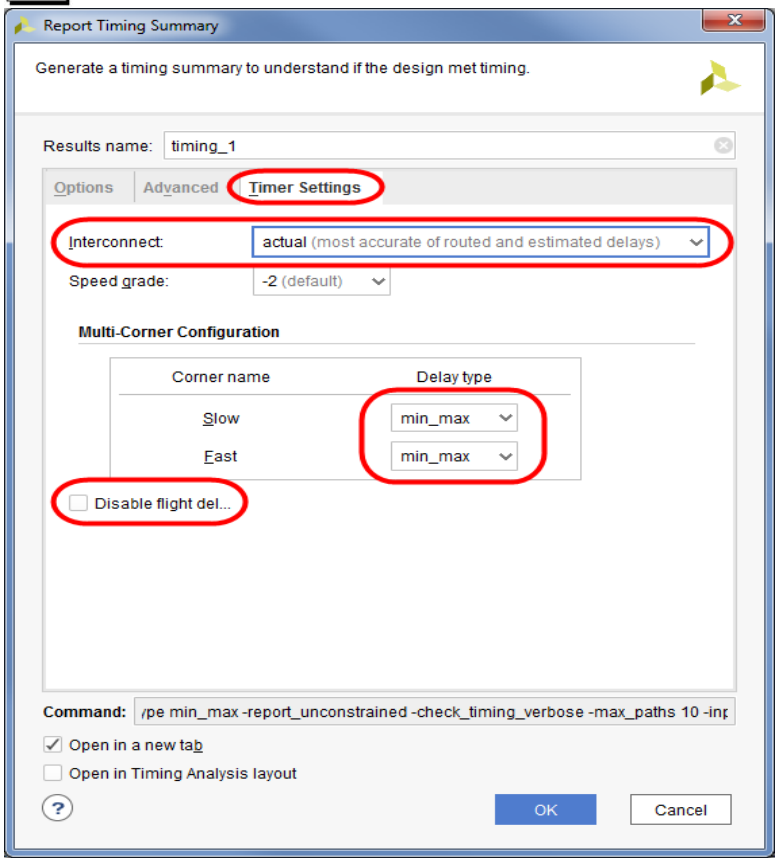
- `-quiet` (Optional): Execute the command quietly, ignoring any command line errors and returning no error messages if the command fails. The command also returns `TCL_OK` regardless of any errors encountered during execution.
- `-verbose` (Optional): Temporarily override any message limits and return all messages from this command. Note that message limits can be defined with the `set_msg_limit` command.

© Copyright 2019 Xilinx

Slide 10-10:

Report Timing Summary (Timer Settings)

129117**slide



129117!*note

Note that for a post-implementation design, the delay model is set to *actual* by default. If routed, actual corresponds to routed net delays.

The timing engine runs in quad timing mode, analyzing min and max delays for both slow and fast corners. The type of analysis performed can be configured with the `config_timing_corners` command. However, changing the default is not recommended because this reduces the timing analysis coverage.

© Copyright 2019 Xilinx

Slide 10-11:

Reviewing Timing Paths

113069**slide

- Critical paths are highlighted in red
- Show Only Failing Paths button will display critical paths
- Expand the section to view the setup and hold paths
- Individual path report can be obtained to analyze requirements, data path delays, etc.

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Introduction to Timing Exceptions

Slide 11-1:

112526**slide

2019.2
112526!*note

This module introduces the application of multicycle paths, false paths, and max/min delay exception timing constraints. 25 minutes.

Slide 11-2:

Objectives

121356**slide

After completing this module, you will be able to:

- Identify where timing exceptions are needed in a design
- Define multicycle path exceptions
- Define max and min path delay exceptions

Slide 11-3:

Timing Exceptions

112528**slide

- Situations exist where the normal setup and hold checks are not correct for a portion of a design
  - Some aspects of the architecture require modifications of these checks

© Copyright 2019 Xilinx

  - These are accomplished by timing exceptions

- Timing exceptions modify the existing setup/hold check
  - Provide the designer with the ability to accurately describe "non-default" timing relationships on static timing paths

© Copyright 2019 Xilinx

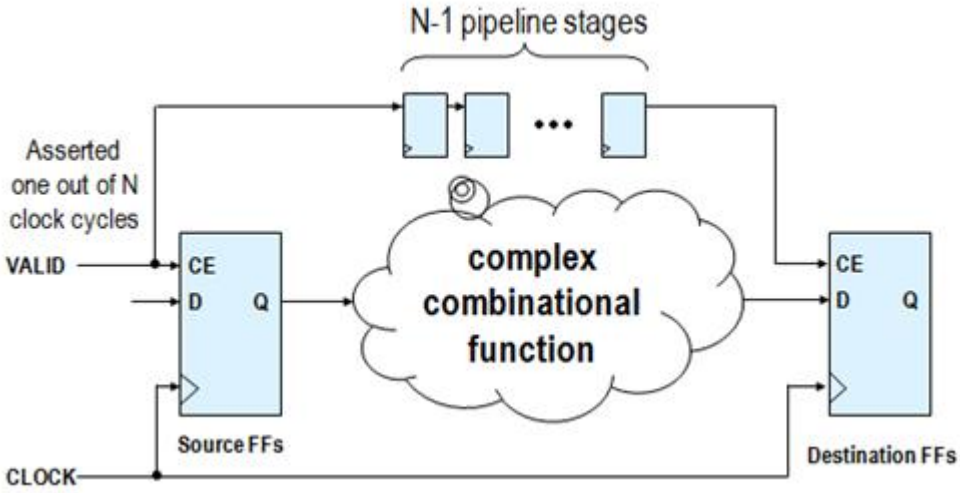
Slide 11-4:

Multicycle Paths

112529**slide

- Multicycle paths occur when registers are not updated on consecutive clock cycles
  - Always at least N clock cycles between the time the source flip-flops update and the destination flip-flops capture the result
  - Typically, the registers are controlled by a clock enable

- Example shows one form of an N-cycle, multicycle path



112529!*note

© Copyright 2019 Xilinx

Multicycle paths can take many forms. The one shown above is an intentionally designed multicycle path for a complex combinatorial function.

As long as VALID is asserted for only one clock and with at least N clock periods between assertions, this circuit ensures N complete clock cycles for the static timing path that starts at the source flip-flops and ends at the destination flip-flops.

The N-1 pipeline stages on the VALID signal can be omitted if it can be ensured that VALID will run periodically.

The pipelining ensures that the destination flip-flops update N cycles after each and every assertion of VALID, even if no further assertions of VALID occur.

The path from the source flip-flops to the destination flip-flops is an N-cycle, multicycle path and should be constrained as such.

When coding designs that use multicycle paths, it is a good idea to have a comment in the module description that explains the multicycle path and to add the multicycle constraint to the constraint file as soon as possible.

Slide 11-5:

Creating a Multicycle Path Exception

`112530**slide`

- Multicycle paths are defined with the `set_multicycle_path` command
  - `set_multicycle_path <path_enumeration> <check_type> <multiplier>`
    - `<path_enumeration>` are options to identify the set of paths to which to apply the `set_multicycle_path` command
      - Uses the `-from`, `-through`, and `-to` options (among others)
    - `<check_type>` determines the static timing check affected
      - `-setup` or `-hold`
    - `<multiplier>` determines how many clock periods to use for the multicycle path

`112530!*note`

© Copyright 2019 Xilinx

The `set_multicycle_path` command is used whenever the designer wants to instruct the tool to use non-standard launch and capture edges for deriving the requirement on a static timing path.

Often this is done for paths where the circuit structure allows more than one clock cycle for the propagation through a path (hence the name), but there are other applications as well.

For example, in certain types of I/O interfaces, it is necessary to use non-standard edges for launch and capture, which will also require a `set_multicycle_path` command, even though the path does not actually take “multiple cycles”.

© Copyright 2019 Xilinx

Slide 11-6:

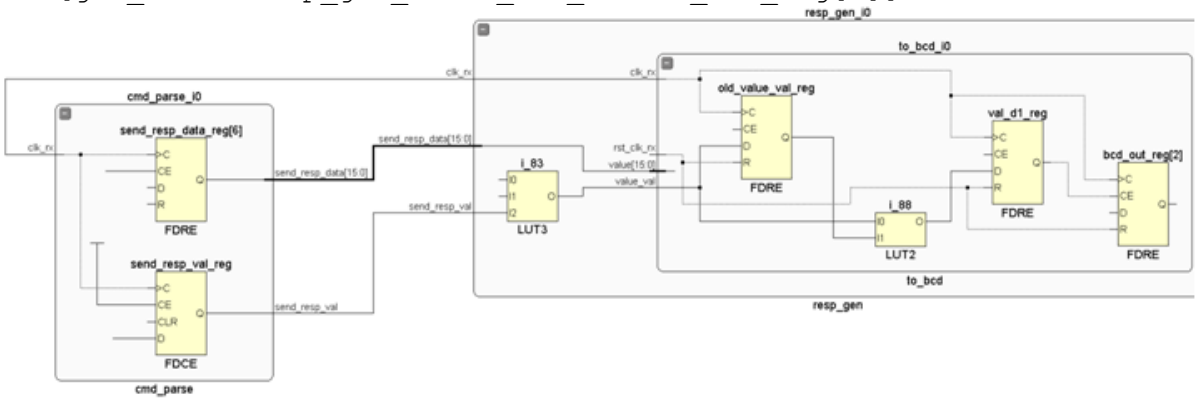
Multicycle Path Example

112533**slide

- Signal send_resp_data changes infrequently
  - Signal send_resp_val indicates when send_resp_data is valid
    - After edge detection and one cycle of delay, this is the enable for bcd_out which captures the result of the BCD calculation of send_resp_data

- Intentionally coded two-cycle, multicycle path

```
create_clock -name my_clk -period 5 [get_ports clk_pin_p]
set_multicycle_path -from [get_cells
cmd_parse_i0/send_resp_data_reg[*]] \
-to [get_cells resp_gen_i0/to_bcd_i0/bcd_out_reg[*]] 2
```



112533!*note

In addition to the command above, this path would also require:

```
set_multicycle_path -from [get_cells
cmd_parse_i0/send_resp_data_reg[*]] -to [get_cells
resp_gen_i0/to_bcd_i0/bcd_out_reg[*]] -hold 1
```


© Copyright 2019 Xilinx

על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

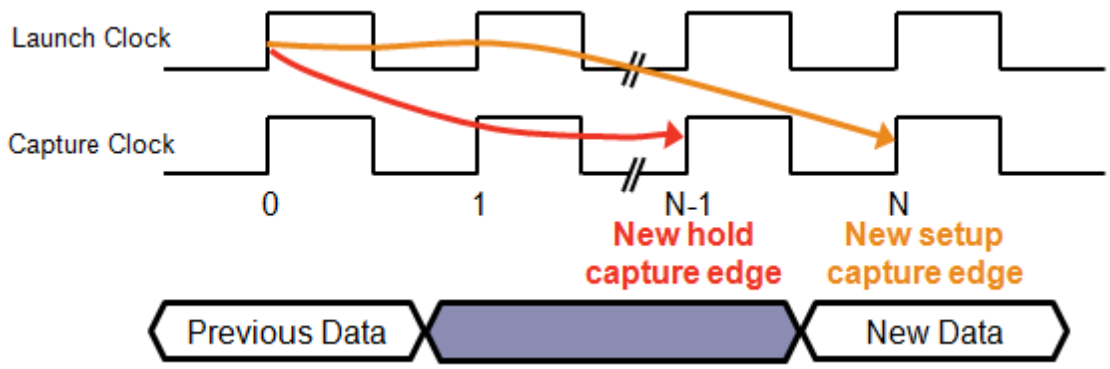
Slide 11-7:

Modifying the Setup and Hold Capture Edge

121352**slide

- The setup capture edge can be modified by specifying a multiplier
  - `set_multicycle_path -from $from_list -to $to_list <N>`
  - The new setup capture edge will be <N> edges after the setup launch edge

- The hold capture edge usually needs to be modified with the `set_multicycle_path -hold` command
  - `set_multicycle_path -from $from_list \ -to $to_list -hold <N-1>`
  - Returns the hold capture edge to the correct edge



121352!*note

© Copyright 2019 Xilinx

By default, the setup check is performed on the edge following the launch clock edge, while the hold check is performed on the same edge as that of the launch clock edge. In the case of a multicycle path, the setup capture edge gets shifted to Nth edge by using a `set_multicycle_path` command, as demonstrated in the image.

The hold capture edge is one clock period before the setup capture edge. This is generally not the correct hold relationship to check for a multicycle path, and this check is almost guaranteed to fail.

The hold edge needs to be modified using multicycle path command, the command moves the hold capture edge back by N-1 clock periods. Because its default was at clock N-1, this returns it to the correct position, which is the same time as the launch edge.

© Copyright 2019 Xilinx

Slide 11-8:

Enumerating Paths

112531**slide

- A static timing path starts at a clocked element, ends at a clocked element, and traverses through some number of combinatorial elements

- A path can be enumerated by specifying information about the path
  - -from <objects>: enumerates all paths that start at the specified objects
    - List of clock pins of clocked cells, clocked cells themselves, ports, or clocks
  - -to <objects>: enumerates all paths that end at the specified objects
    - List of data pins of clocked cells, clocked cells themselves, ports, or clocks
  - -through <objects>: enumerates all paths that go through the specified objects
    - List of cells, nets, or pins
    - Can use multiple –through <objects> options

- The <objects> options can be lists of objects, usually obtained with the get_* commands

- The above options can be combined; the enumerated paths are only those that match all the specified criteria

112531!*note

The Vivado Design Suite timing engine can independently process different transitions (rising/falling) as they propagate through static timing paths. To distinguish different paths based on the transitions, you can use the -rise_from, -rise_to, -rise_through, -fall_from, -fall_to, and -fall_through options.

The -from and -to options can be used to specify the startpoints and endpoints of static timing paths.

The startpoints of a path are:

- Clock pins of clocked cells.

© Copyright 2019 Xilinx

- The clocked elements themselves; it is acceptable to specify the cell, rather than the clock pin of the cell.
- A clock. When a clock is used, all clocked elements clocked by that clock are considered part of the `-from` specification.
- A port of the design. The “virtual flip-flop” created using the `set_input_delay` command does not have a name and cannot be found using a `get_*` command. Therefore, the port is used as the startpoint.

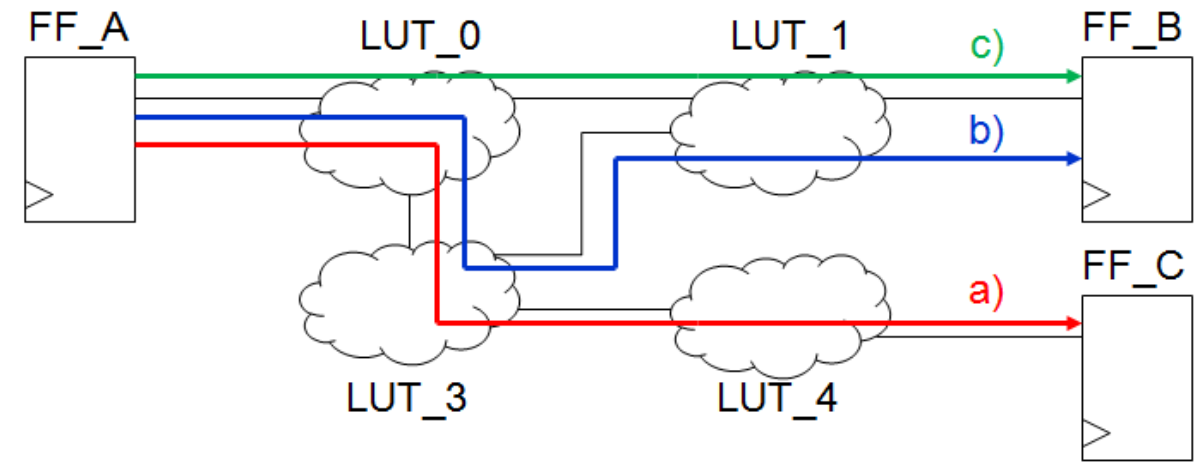
The endpoints of a path are similar. They are normally the data input pins of clocked elements, but may be the cells themselves, clocks, or output ports.

The mechanism for enumerating paths is used consistently throughout the Vivado Design Suite timing engine. The same mechanism is used for specifying other timing exceptions and is also used for generating custom timing reports with the `report_timing` command.

Slide 11-9:

Enumerating Paths Example

57826**slide




```
a) … -from FF_A -through LUT_0 -through LUT_4 -to FF_C
b) … -from FF_A -through LUT_3 -through LUT_1 -to FF_B
c) … -from FF_A -through LUT_0 -through LUT_1 -to FF_B
```

57826!*note

© Copyright 2019 Xilinx

Each `-through` option can specify a list of objects. A path matches the `-through` if it passes through any of the objects in the list. Therefore, if a command as two `-through` lists, it will match any path that passes through any element from the first list and any element from the second list.

© Copyright 2019 Xilinx

Slide 11-10:

Creating a False Path Exception

112534**slide

- False paths are defined with the `set_false_path` command
  - `set_false_path <path_enumeration> <check_type>`
    - `<path_enumeration>` are options to identify the set of paths to which to apply the `set_false_path` command
      - Uses the same options as the `set_multicycle_path` command
    - `<check_type>` determines the static timing check affected
      - `-setup` or `-hold`
      - When not specified, both setup and hold checks are affected

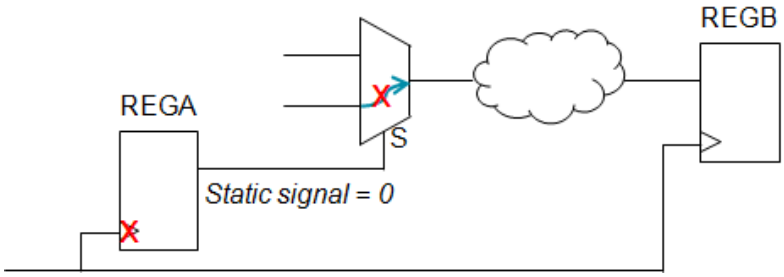
Slide 11-11:

Constants and Pseudo-Constants

112535**slide

© Copyright 2019 Xilinx

- If a design has a signal that cannot make any transitions, then no static timing event can propagate from that point forward
  - All paths that start at that point should not be checked by static timing analysis
  - Startpoint can be a true constant or a pseudo-constant
    - A pseudo-constant only makes transitions when the destination clocked elements are inactive; i.e., in reset or held inactive by some enable
- When a net is a constant value, some other timing paths may be disabled
  - When the S of a MUX is held at 0, nothing can propagate through the I1 input
  - `set_false_path -from [get_cells REGA]`
    `set_false_path -through [get_pins MUXA/I1]`



112535!*note

© Copyright 2019 Xilinx

The fact that REGA cannot make a transition (it is always at a constant value) means that no propagation can start at the output of REGA. Therefore, the static timing paths that start at REGA/C should be disabled.

Furthermore, because the output of REGA is always a 0, this permanently selects the I0 input of MUXA. Thus, any propagation through the I1 input of the MUXA is blocked, and any static timing path that goes through that pin should also be disabled.

It is also possible to declare paths false for certain transitions. If, for example, REGA was reset to 0, but at any point in time can make one (and only one) transition to a 1, then this path can have a rising transition but not a falling transition. This can be constrained using `set_false_path -fall_from [get_cells REGA]`.

Slide 11-12:

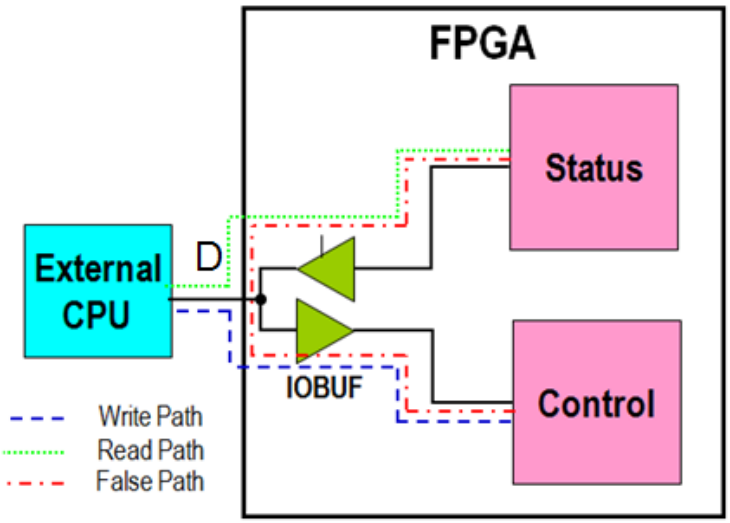
Mutually Exclusive Sub-Paths

112536**slide

- If a path can be viewed as two sub-paths where, at any one time, one of the two sub-paths blocks propagation, the path is false
  - Often occurs through bidirectional interfaces

© Copyright 2019 Xilinx

```
set_false_path -from
[get_cells Status/*] \
-through [get_ports D[*]]
-to [get_cells Control/*]
```



112536!*note

In the above design, a bidirectional data bus is used as part of the communication between an FPGA and an external CPU.

The CPU can perform two operations, a read and a write:

- During writes, the output driver in the FPGA is disabled so that the CPU can send data to the flip-flops in the Control module.
- During reads, the output driver is enabled so that the FPGA can send information from the Status registers to the CPU.
- During the write cycle, presumably, the Control flip-flops do not capture the contents of the D bus.

A third static timing path is seen by the tool—from the Status flip-flops, through the bidirectional D port back to the Control flip-flops. This path is false.

Even though 3-states are not used internally in the FPGA, it is possible to have similarly mutually exclusive subpaths using multiplexers or other logic.

Slide 11-13:

Setting Maximum and Minimum Delays

112538**slide

© Copyright 2019 Xilinx

- Setup check requirements can be overridden with the `set_max_delay` command
  - `set_max_delay <path_enumeration> <delay>`
    - `<path_enumeration>` are options to identify the set of paths to which to apply the `set_max_delay` command
      - Uses the same options as the `set_multicycle_path` command
    - `<delay>` determines the requirement for the setup check

- Hold check requirements can be overridden with the `set_min_delay` command
  - `set_min_delay <path_enumeration> <delay>`

`112538!*note`

To override the requirements for the setup check, the `set_max_delay` command must be applied to the static timing path.

The `-from` (if it exists) must specify the startpoint of a static timing path—either the clock pin of the clocked element, the cell, a clock, or a port.

The `-to` (if it exists) must specify the endpoint of a static timing path—either the data pin of the clocked element, the cell, a clock, or a port.

The same is true for a hold check and the `set_min_delay` command.

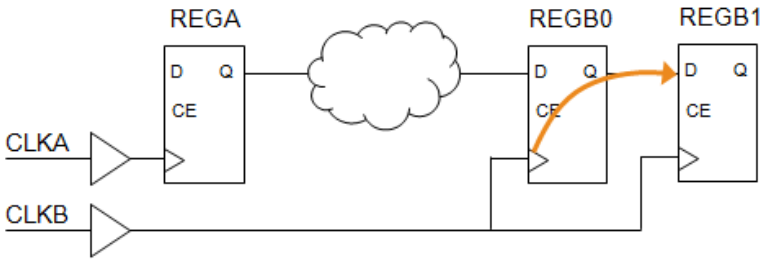
© Copyright 2019 Xilinx

Slide 11-14:

Constraining Metastability Flip-Flops

112539**slide

- When a flip-flop (REGB0) samples an asynchronous input, the flip-flop can go metastable
  - The metastability will probabilistically resolve after some time
    - Back to back flip-flops allow one clock period for the metastability to resolve before the second flip-flop samples it (REGB1)



- The Vivado timing engine sees the REGB0 → REGB1 path as a normal static timing path, subject to a normal setup check
  - Allows one clock period of CLKB for the propagation, including routing
- To leave time for metastability resolution, the requirement on this path should be changed
  - `set_max_delay -from [get_cells REGB0] -to [get_cells REGB1] 2`

112539!*note

At higher clock frequencies, more than two back-to-back flip-flops may be needed for metastability resolution.

In this specific case, another way of placing the two flip-flops together is by setting the ASYNC_REG property on the two flip-flops. Setting the ASYNC_REG property set on the flip-flops prevents optimization of these cells during synthesis, `opt_design`, and `phys_opt_design` and forces packing of the flip-flops into the same slice.

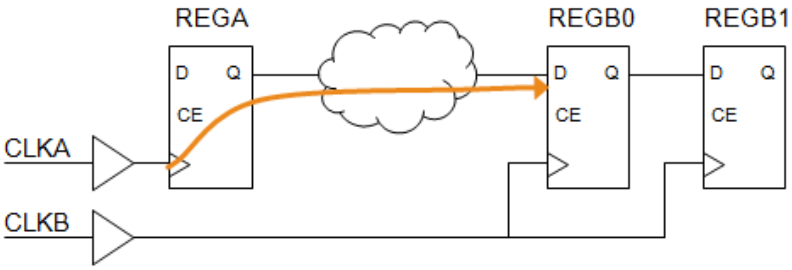
© Copyright 2019 Xilinx

Slide 11-15:

Paths Between Different Clock Domains

112540**slide

- When crossing asynchronously between different clock domains a clock crossing circuit is required
  - Many approaches exist
    - Use Gray code
    - Use an enable to determine a stable point
    - Use a FIFO (which generally uses the above for the empty/full generation)
  - In these cases, the clock propagation analysis is irrelevant and should be overridden with the –datapath_only option
  - `set_max_delay -from \ [get_cells REGA] -to \ [get_cells REGB0] 5 -datapath_only`



  - All of these approaches have implicit assumptions
    - Generally, the skew between different bits of the bus to be crossed cannot exceed one clock period
    - It is not possible to constrain skew, but it is possible to constrain the maximum delay on all bits of the bus

Slide 11-16:

Combinatorial Paths Between Ports

112541**slide

- The `set_max_delay` and `set_min_delay` commands can be used to constrain paths between primary inputs and primary outputs
  - –datapath_only is not necessary because no clocks are associated with the startpoint and endpoint

© Copyright 2019 Xilinx

```
set_max_delay -from [get_ports CombIn] -to [get_ports CombOut] 8
```

112541!*note

This path can also be constrained by adding a `set_input_delay` to the CombIn port and a `set_output_delay` to the CombOut port. This would complete this partial path on both ends.

Slide 11-17:

Max/Min Delay on a Non-Startpoint

112542**slide

- Applying a `set_max_delay -from` or `set_min_delay -from` to a pin or net which is NOT a start point of a static timing path will cause the path to be segmented
  - A new path will start at the point applied
  - No clock propagation will be considered to the new startpoint
- If done in error, this can cause unexpected results
  - That is, specifying the output pin of a flip-flop as the `-from` object instead of the clock pin
    - Static timing paths start at the clock pin of clocked elements
- When a `set_max_delay` or `set_min_delay` is attached to a non-startpoint, the tool will issue the following warning
  - WARNING: [Constraints-402] set_max_delay: 'REGA/Q' is not a valid start point

© Copyright 2019 Xilinx

112542!*note

Note that a `set_max_delay` or `set_max_delay` can be applied to any path using the mechanisms of enumerating paths.

If the enumeration contains only `-from` and `-to` to valid start/endpoints, or `-through` to any point on the path, the path will not be segmented.

Path segmentation only occurs specifically when a `-from` or `-to` is applied to a non-startpoint/endpoint.

© Copyright 2019 Xilinx

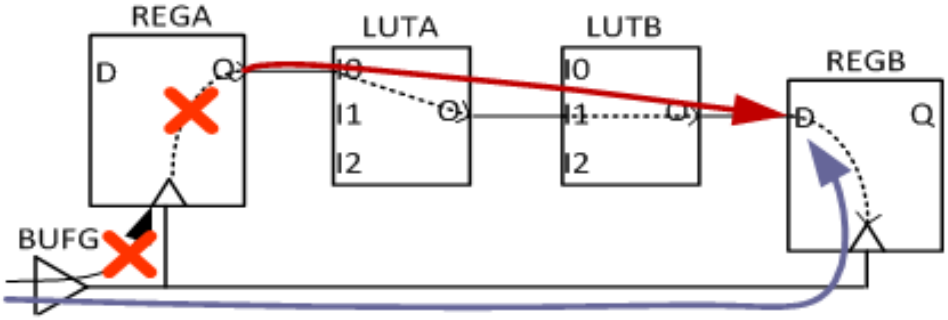
Slide 11-18:

Path Segmentation on a Startpoint

112543**slide

- REGA/Q becomes a new start point in order to apply the max delay constraint
  - ALL timing paths going through REGA/Q are impacted
    - The path from REGA/C to REGB is no longer considered a static timing path
    - No source clock delay is used for paths from REGA/Q

```
set_max_delay -from [get_pins REGA/Q] -to [get_pins REGB/D] 6
```



112543!*note

Path segmentation can be useful in breaking combinatorial feedback paths.

Slide 11-19:

Path Segmentation on an Endpoint

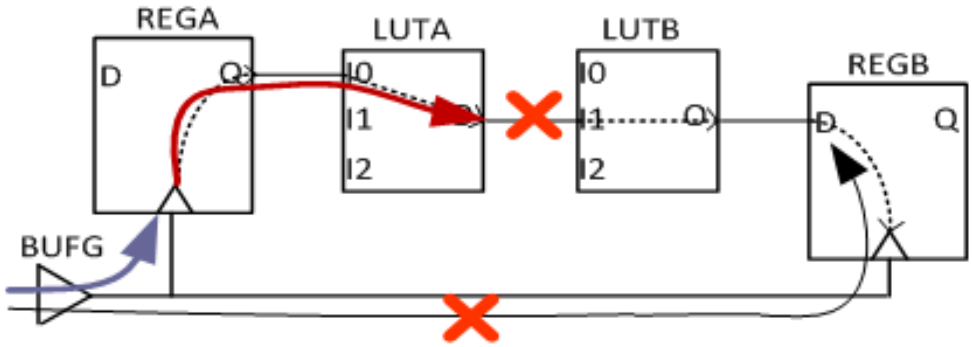
112544**slide

- Applying a `set_max_delay -to` to a pin that is not an endpoint will result in the path being segmented at the new endpoint
  - ALL timing paths going through the new endpoint are impacted

© Copyright 2019 Xilinx

- The path from REGA to REGB is no longer considered a static timing path
- No destination clock delay is used for the new endpoint

```
set_max_delay -from [get_pins REGA/C] -to [get_pins LUTA/O] 6
```



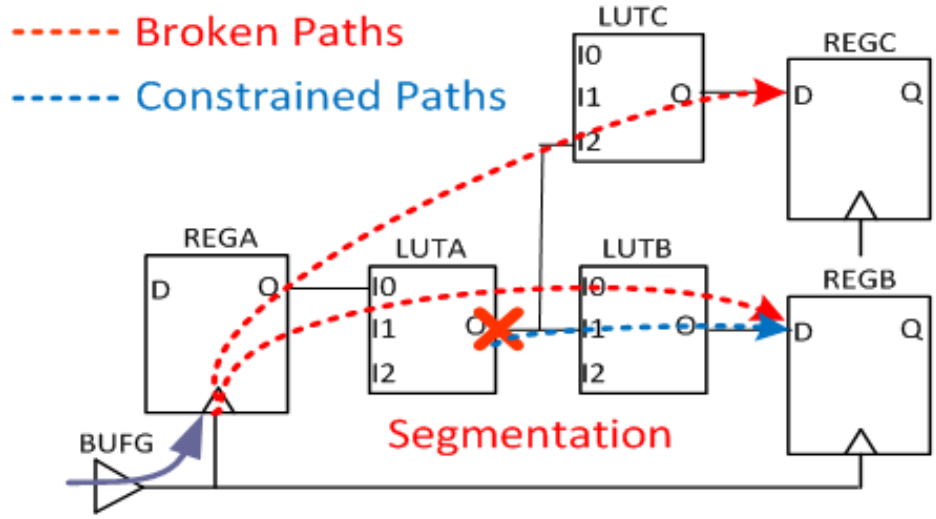
Slide 11-20:

Path Segmentation – Breaking Other Paths

112545**slide

- Path segmentation on a datapath pin will break all the paths through that pin
  - A pin cannot be both a start point (or endpoint) AND a combinatorial pin
  - No clock reaches start point/endpoints created by path segmentation

```
set_max_delay -from [get_pins LUTA/O] -to [get_pins REGB/D] 6
```



© Copyright 2019 Xilinx

112545!*note

The paths from REGA/C to both REGB/D and REGC/D are broken. Only the path specified by the `set_max_delay` is constrained.

© Copyright 2019 Xilinx

Slide 11-21:

Summary

112546**slide

- Normal setup and hold checks can be overridden with exceptions
- Multicycle path exceptions instruct the tool to allow multiple clock periods for propagation
- False path exceptions remove setup and hold checks from a path
- Max and min delay exceptions override the normal calculation for the setup/hold check requirement
- Max and min delay exceptions applied to non-start/endpoints will result in path segmentation

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Introduction to Vivado Reports

Slide 12-1:

```
113166**slide

2019.1
113166!*note
```

This module describes the different reports generated by the Vivado® IDE. 25 minutes.

Slide 12-2:

Objectives

```
113167**slide
```

After completing this module, you will be able to:

- Describe the different reports generated by Vivado IDE
- Explain some of the various timing reports and commands used by the Vivado IDE

© Copyright 2019 Xilinx

Slide 12-3:

Vivado Reports

113168**slide

- The Vivado IDE generates different reports at different stages (after synthesis, after implementation)

- `report_timing_summary`, `check_timing`, `report_clock_networks`, `report_utilization`, `report_drc`, `report_methodology`, `report_design_analysis`, `report_clock_interaction`, `report_cdc`, `report_exception`, `report_bus_skew`, `report_power`, `report_qor_suggestion`, `report_clocks`, etc.

- Timing reports help you determine why your design fails to meet its constraints
  - Reports contain detailed descriptions of paths that fail their constraints
  - These reports are customizable and have a variety of timing information
  - These reports can provide timing estimates or true timing depending on the phase (synthesis or implementation)
  - Use timing reports after implementation as a final analysis of whether your design has met timing

- The slack histogram helps you understand which paths are the most timing critical

113168!*note

After implementing a design, use timing reports to determine overall design performance.

You should review the details of each failed constraint to determine why the design does not meet performance objectives.

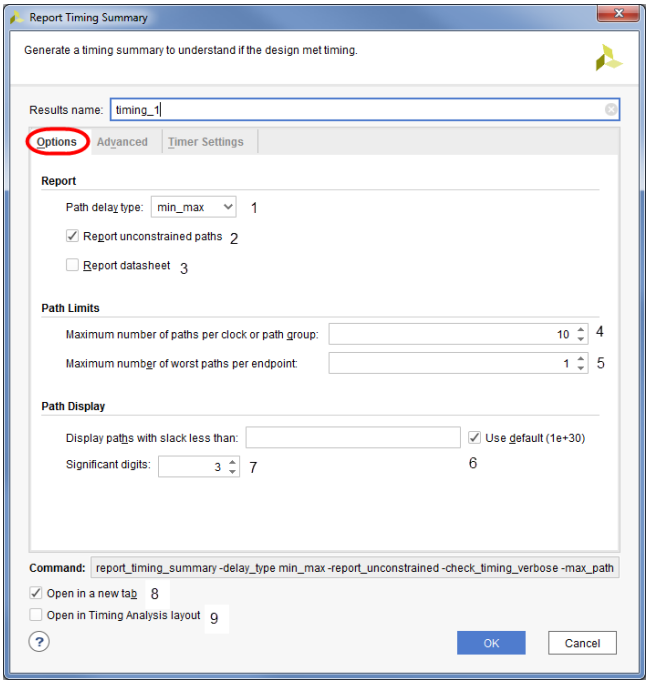
© Copyright 2019 Xilinx

Slide 12-4:

Report Timing Summary and Report Timing

113169**slide

- Post-implementation timing sign-off
  - report_timing_summary
- Interactive and detailed timing analysis
  - report_timing
- Filtering capability via options
  - Select start point and end point
  - Change the format (path type) reported
  - Control the number of paths



113169!*note

- Filtering capability tunes your timing analysis for what you want
  - Type of analysis - delay_type default=min_max
  - Specify start, thru, end points - Choose based on types: cell instance, clocks, pins, ports, nets
  - Sort of path
    - Slack (default in Tcl console)
    - Group (default in IDE)
  - Specify a range of slack values
  - Reporting of unconstrained paths

Note: Unconstrained paths always show after constrained paths, assuming enough paths are displayed.

© Copyright 2019 Xilinx

Format	Displays
short	Source and destination clock insertion delay startpoint and endpoint *without* datapath details
full	Source and destination clock insertion delay startpoint and point *with* datapath details
full_clock	Same as full plus details on source and destination clock paths
full_clock_expanded (default)	Same as full_clock plus complete details on source / destination clock paths
end	Shows endpoint, datapath delay, path requirement skew and slack (one line per path)
summary	Shows startpoint, endpoint, slack (one line per path)

- Controllable amount of detail
  - Would you like to show ***input pins*** in the detailed path
  - Default is output and clock pins only in detailed path report

- Controllable number of paths
  - Total number of paths (-max_paths)
  - Number of paths per path group if *-sort_by group* is used, group ~ clock
  - Maximum number of paths per endpoint (-nworst), Default=1

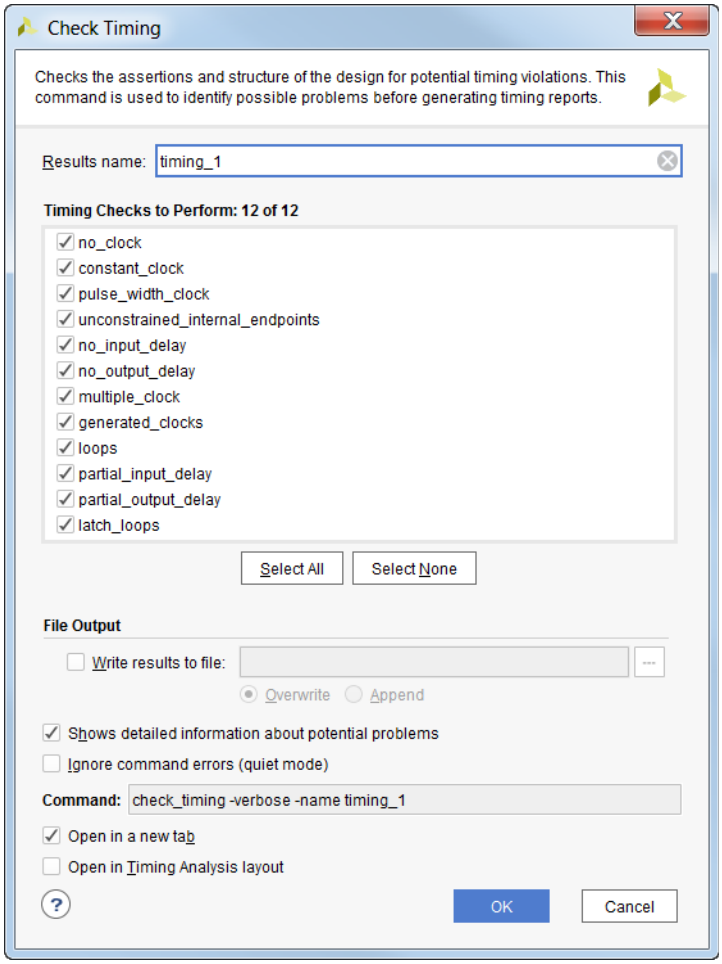
© Copyright 2019 Xilinx

Slide 12-5:

Check Timing

113171**slide

- Is my design constrained properly? Are there missing constraints in the design?
  - `check_timing report`: Checks missing constraints for
    - No clock
    - Constant clock
    - Unconstrained endpoints
    - No input delay
    - No output delay
    - Multiple clocks on same pin
    - Generated clocks
    - Combination feedback loops



113171!*note

Description

Checks the design elements of ports, pins, and paths against the current timing constraints. Use this command to identify possible problems with design data and timing constraints before running the `report_timing` command. The `check_timing` command runs a series of default timing checks and reports a summary of any violations found. To get detailed information about violations, use the `-verbose` option.

© Copyright 2019 Xilinx

Default Timing Checks

- `generated_clocks`: Checks that derived clocks defined by the `create_generated_clock` command reference primary clocks, defined by the `create_clock` command, rather than referencing another generated clock.
- `loops`: Checks for and warns of combinational feedback loops in the design.
- `multiple_clock`: Warns if multiple clocks reach a register clock pin. If more than one clock signal reaches a register clock pin, it is unclear which clock will be used for analysis. In this case, use the `set_case_analysis` command so that only one clock will propagate to the register clock pin.

© Copyright 2019 Xilinx

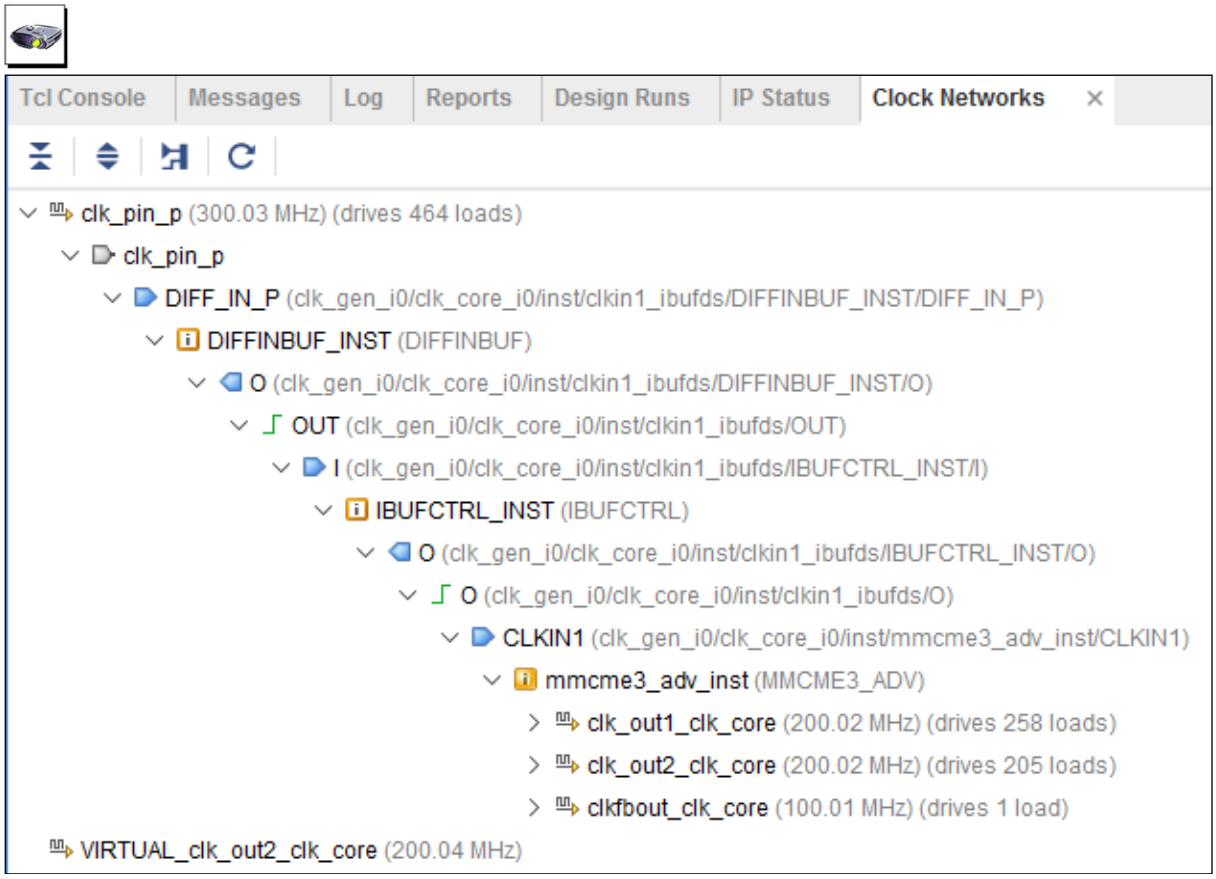
- `no_clock`: Reports unclocked registers. In this case, no setup or hold checks are performed on data pins related to the register clock pin.
- `no_input_delay`: Reports the input ports without an input delay constraint. Input delays can be assigned with the `set_input_delay` command. Input ports that are unclocked will not be checked for input delays.
- `no_output_delay`: Reports the output ports without an output delay constraint. Output delays can be assigned with the `set_output_delay` command. Output ports that are unclocked will not be checked for output delays.
- `partial_input_delay`: Reports the input ports having partially defined input delay constraints. Assigning `set_input_delay -max` or `set_input_delay -min` to an input port without assigning the other creates a partially defined input delay. In such cases, paths starting from the input port may become unconstrained and no timing checks will be done against the port. Assigning `set_input_delay` without specifying either `-min` or `-max` allows the tool to assume both min and max delays and so does not result in a partial input delay. Note that unclocked input ports are not checked for partial input delays.
- `partial_output_delay`: Reports the output ports having partially defined output delay constraints. Assigning `set_output_delay -max` or `set_output_delay -min` to an output port without assigning the other creates a partially defined output delay. In such cases, paths reaching the port may become unconstrained and no timing checks will be done against the port. Assigning `set_output_delay` without specifying either `-min` or `-max` allows the tool to assume both min and max delays, and so does not result in a partial output delay. Note that unclocked output ports are not checked for partial output delays.
- `unconstrained_endpoints`: This warning identifies timing path endpoints that are not constrained. Endpoints at register data pins are constrained by clock assignment with the `create_clock` command. Endpoints at output ports are constrained by output delays with the `set_output_delay` or `set_max_delay` commands.
- `unexpandable_clocks`: Reports clock sets in which the period is not expandable with respect to each other, when there is at least one path between the clock sets. A clock is unexpandable if no common multiples are found within 1000 cycles between the source and destination clocks.

© Copyright 2019 Xilinx

Slide 12-6:

Report Clock Networks

135124**slide



135124!*note

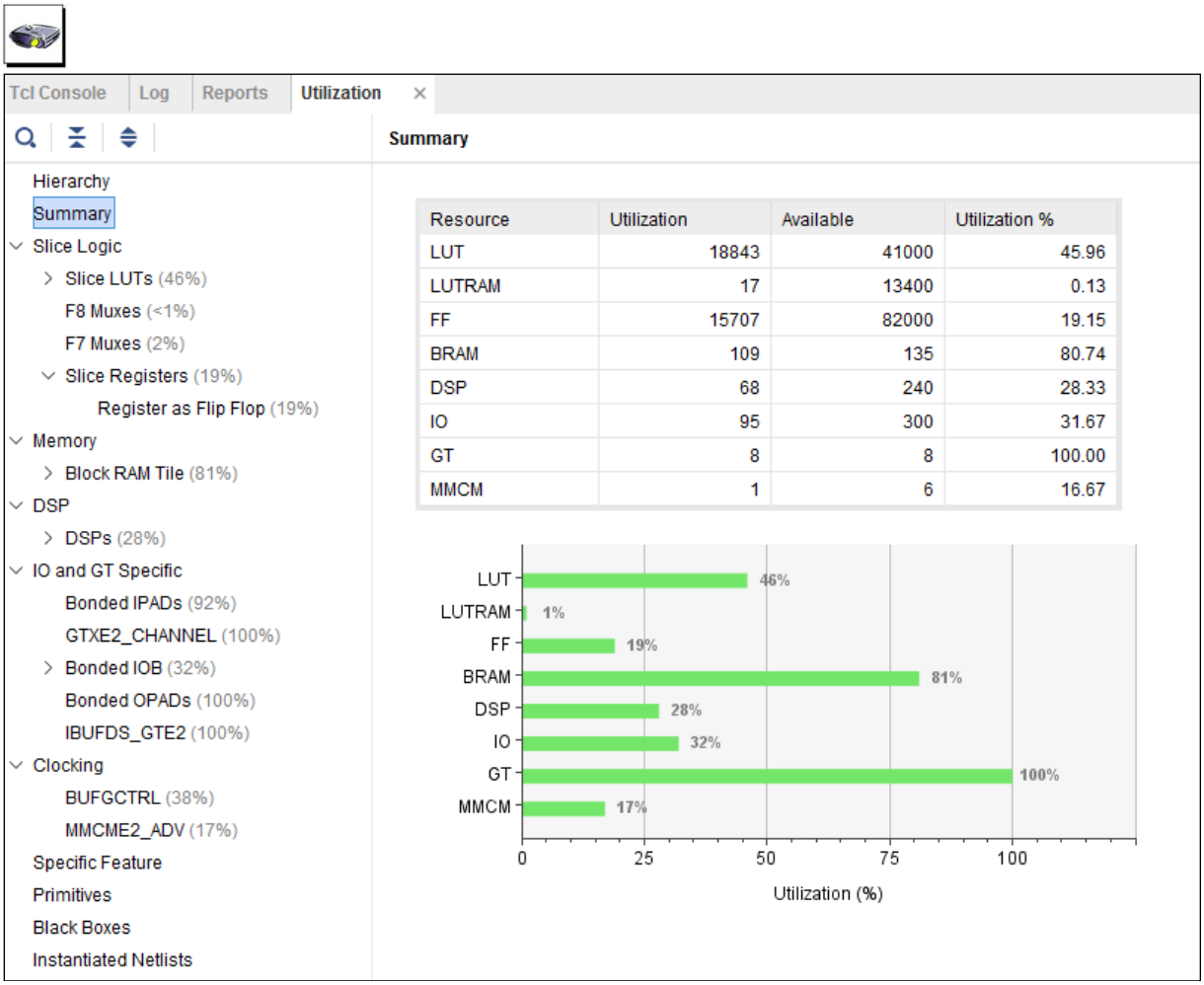
The `report_clock_networks` command provides a tree view of the clock trees in the design. Each clock tree shows the clock network from source to endpoint with the endpoints sorted by type.

© Copyright 2019 Xilinx

Slide 12-7:

Report Utilization

129124**slide



Resource	Utilization	Available	Utilization %
LUT	18843	41000	45.96
LUTRAM	17	13400	0.13
FF	15707	82000	19.15
BRAM	109	135	80.74
DSP	68	240	28.33
IO	95	300	31.67
GT	8	8	100.00
MMCM	1	6	16.67

129124!*note

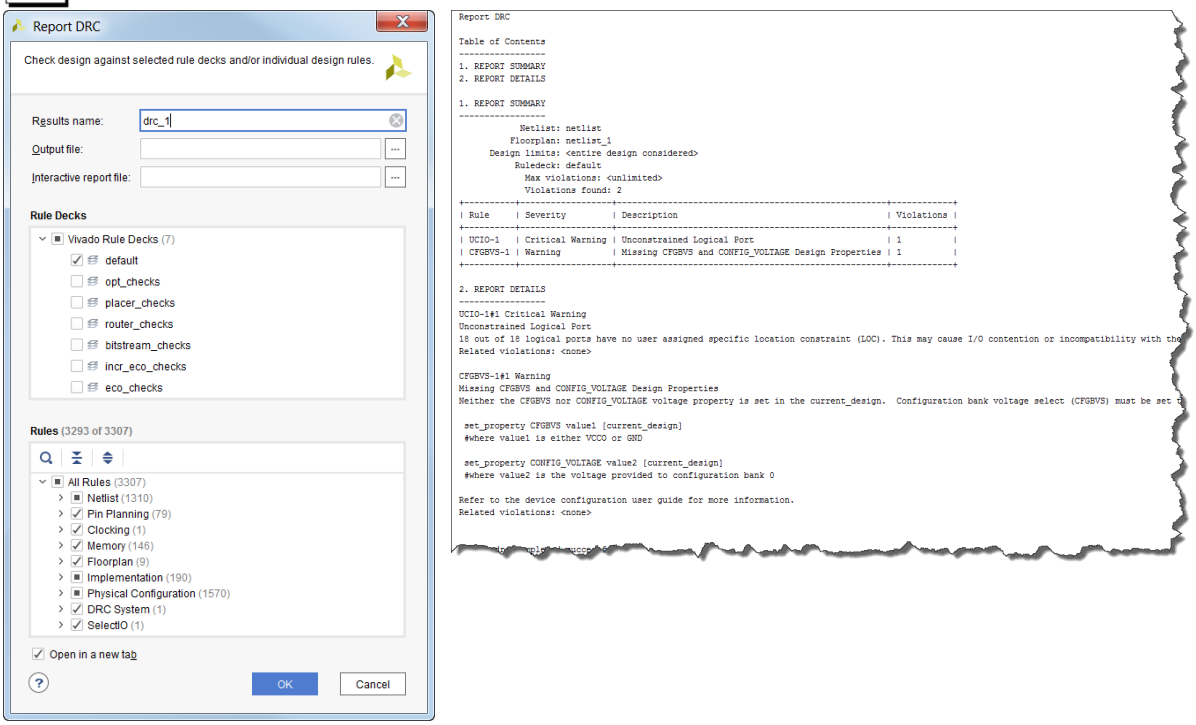
The `report_utilization` command reports the utilization of resources such as LUTs, registers, FIFO, clocking, and I/O resources on the target part by the current synthesized or implemented design. The report is returned to the standard output, unless the -file, -return_string, or -name arguments are specified.

© Copyright 2019 Xilinx

Slide 12-8:

Report DRC

129123**slide



129123!*note

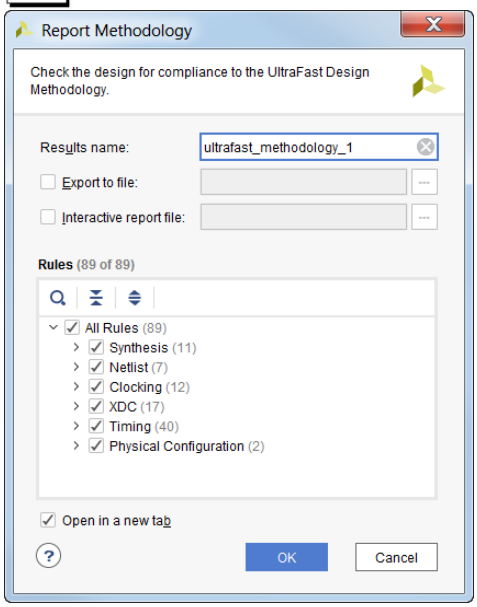
The `report_drc` command checks the design against the specified set of design rules and reports any errors or violations that are found. This runs a default rule deck when the `-checks` or `-ruledeck` options are not specified. Creating a user-defined DRC automatically adds the new design rule check to the default rule deck. DRC rules can be enabled or disabled using the IS_ENABLED property on the rule check object.

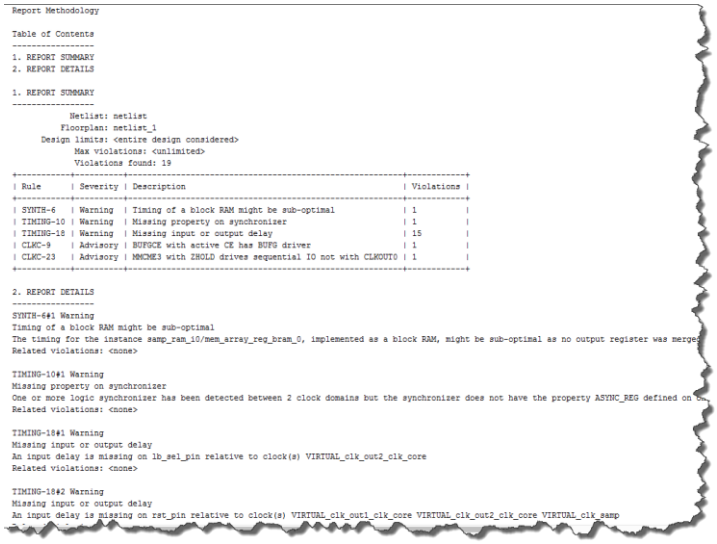
© Copyright 2019 Xilinx

Slide 12-9:

Report Methodology

135119**slide





```
Report Methodology

Table of Contents
-----------------
1. REPORT SUMMARY
2. REPORT DETAILS

1. REPORT SUMMARY
-----------------
            Netlist: netlist
          Floorplan: netlist_1
      Design limits: <entire design considered>
            Max violations: <unlimited>
            Violations found: 19
+-----------+----------+-----------------------------------------------------+------------+
| Rule      | Severity | Description                                         | Violations |
+-----------+----------+-----------------------------------------------------+------------+
| SYNTH-6   | Warning  | Timing of a block RAM might be sub-optimal          | 1          |
| TIMING-10 | Warning  | Missing property on synchronizer                    | 1          |
| TIMING-18 | Warning  | Missing input or output delay                       | 15         |
| CLKC-9    | Advisory | BUFGCE with active CE has BUFG driver               | 1          |
| CLKC-23   | Advisory | MMCME3 with ZHOLD drives sequential IO not with CLKOUT0 | 1          |
+-----------+----------+-----------------------------------------------------+------------+

2. REPORT DETAILS
-----------------
SYNTH-6#1 Warning
Timing of a block RAM might be sub-optimal
The timing for the instance samp_ram_10/mem_array_reg_bram_0, implemented as a block RAM, might be sub-optimal as no output register was merge
Related violations: <none>

TIMING-10#1 Warning
Missing property on synchronizer
One or more logic synchronizer has been detected between 2 clock domains but the synchronizer does not have the property ASYNC_REG defined on
Related violations: <none>

TIMING-18#1 Warning
Missing input or output delay
An input delay is missing on lb_sel_pin relative to clock(s) VIRTUAL_clk_out2_clk_core
Related violations: <none>

TIMING-18#2 Warning
Missing input or output delay
An input delay is missing on rst_pin relative to clock(s) VIRTUAL_clk_out1_clk_core VIRTUAL_clk_out2_clk_core VIRTUAL_clk_samp
```

135119!*note

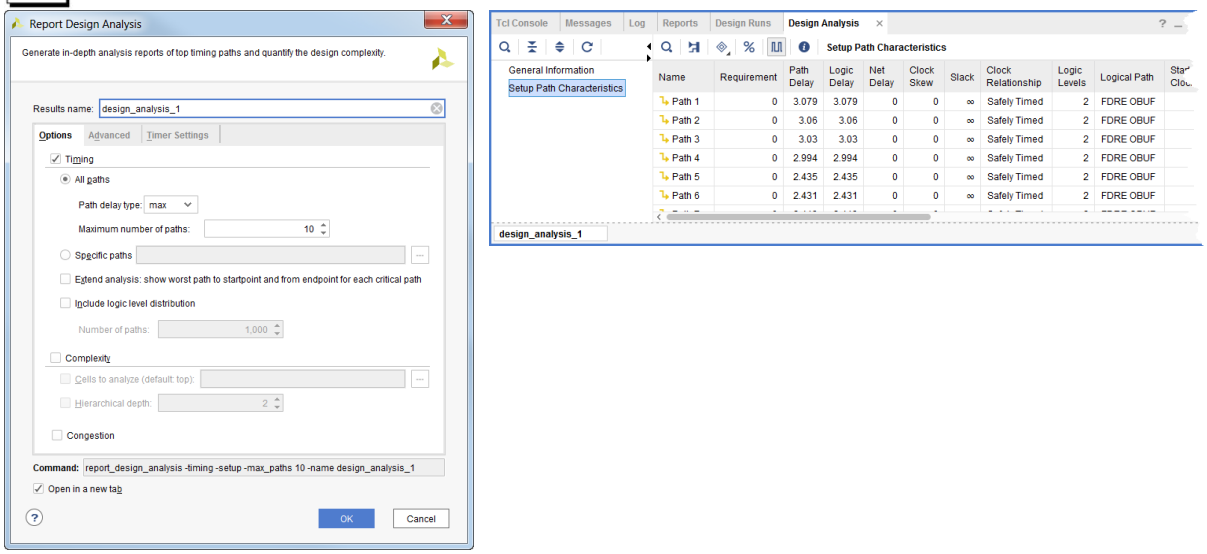
The `report_methodology` command specifically checks for compliance with methodology DRCs, due to importance of methodology. It checks the current design against a specified set of methodology checks and report any errors or violations that are found. Methodology checks are a special class of design rule checks (DRC) that are accessible through this separate Tcl command. The methodology checks are a necessary part of the design flow, and should be considered mandatory after implementation and prior to generating the bitstream.

© Copyright 2019 Xilinx

Slide 12-10:

Report Design Analysis

135127**slide



135127!*note

The `report_design_analysis` command provides information on timing path characteristics, design interconnect complexity, and congestion. You can use this information to make design or constraint changes that improve QoR and possibly alleviate routing congestion.

Report Design Analysis generates three categories of reports:

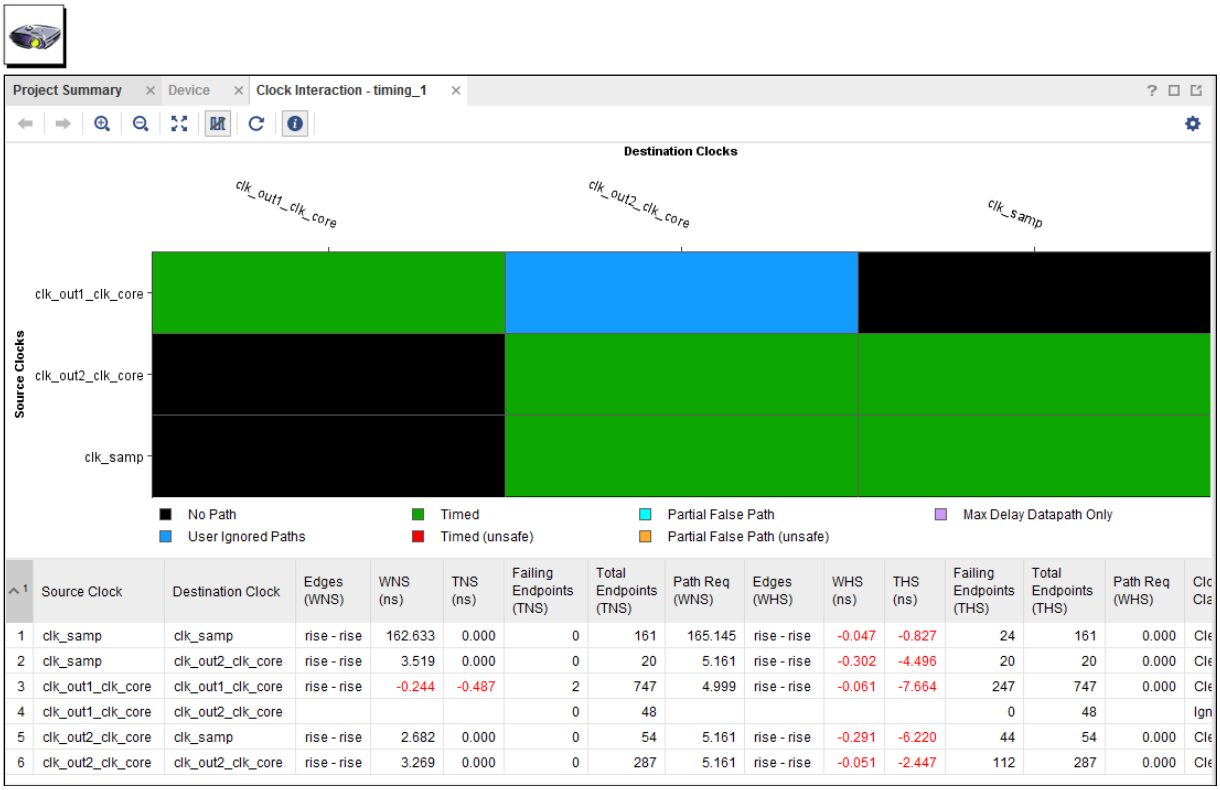
- Timing: reports timing and physical characteristics of timing paths
- Complexity: analyzes the design for routing complexity and LUT distribution
- Congestion: analyzes the design for routing congestion

© Copyright 2019 Xilinx

Slide 12-11:

Report Clock Interaction

129127**slide



	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints (THS)	Total Endpoints (THS)	Path Req (WHS)	Clo Cla
1	clk_samp	clk_samp	rise - rise	162.633	0.000	0	161	165.145	rise - rise	-0.047	-0.827	24	161	0.000	Cle
2	clk_samp	clk_out2_clk_core	rise - rise	3.519	0.000	0	20	5.161	rise - rise	-0.302	-4.496	20	20	0.000	Cle
3	clk_out1_clk_core	clk_out1_clk_core	rise - rise	-0.244	-0.487	2	747	4.999	rise - rise	-0.061	-7.664	247	747	0.000	Cle
4	clk_out1_clk_core	clk_out2_clk_core				0	48					0	48		Ign
5	clk_out2_clk_core	clk_samp	rise - rise	2.682	0.000	0	54	5.161	rise - rise	-0.291	-6.220	44	54	0.000	Cle
6	clk_out2_clk_core	clk_out2_clk_core	rise - rise	3.269	0.000	0	287	5.161	rise - rise	-0.051	-2.447	112	287	0.000	Cle

129127!*note

The `report_clock_interaction` command analyzes timing paths that cross from one clock domain into another clock domain (source to destination). It helps to identify cases in which there may be data loss or metastability issues. As shown in the figure, the Clock Interaction Report displays as a matrix of clock domains with the source clocks in the vertical axis and the destination clocks in the horizontal axis. The tiles of the matrix are color coded.

- Black – No Path: No paths exist between the two clocks; there is no path from a flip-flop on the source domain to a flip-flop on the destination domain.
- Green – Timed: Paths between the domains have no exceptions; the requirements for these paths are determined normally.
- Dark Blue – User Ignored Paths: All paths between the domains have `set_false_path` (either directly or from a command).

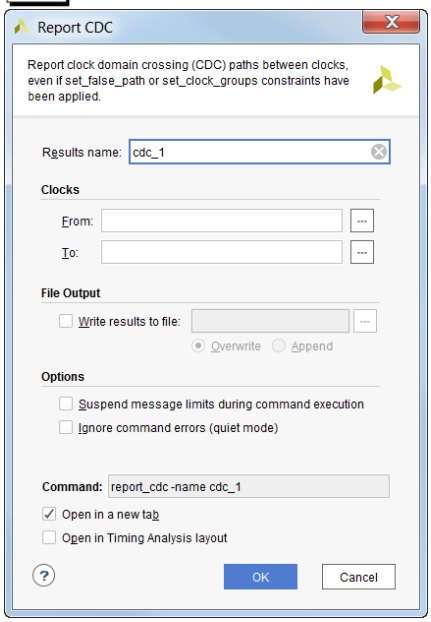
© Copyright 2019 Xilinx

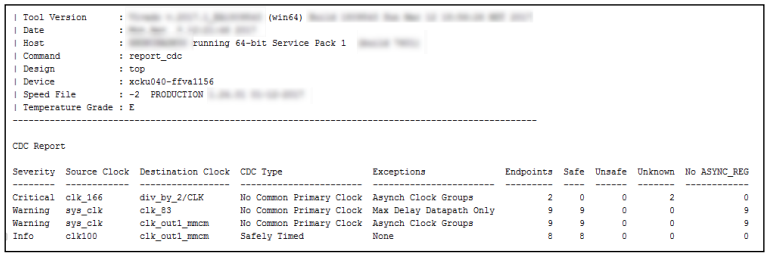
- Red – Timed (unsafe): Paths between the domains have no exceptions and the clocks are un-expandable (have no `set_clock_groups` harmonic relation in the first 1000 clocks).
- Light Blue – Partial False Path: Some paths have exceptions, others do not.
- Orange – Partial False Path (unsafe): Some paths have no exceptions, and the paths are un-expandable.
- Max Delay Datapath Only - Gray: All the paths covered by a `set_max_delay -datapath_only` constraint crossing from the source clock to the destination clock.

Slide 12-12:

Report CDC

129128**slide





```
| Tool Version     : [illegible] (win64) [illegible]
| Date             : [illegible]
| Host             : [illegible] running 64-bit Service Pack 1 [illegible]
| Command          : report_cdc
| Design           : top
| Device           : xcku040-ffva1156
| Speed File       : -2  PRODUCTION [illegible]
| Temperature Grade : E
------------------------------------------------------------------------------------------

CDC Report

Severity  Source Clock  Destination Clock  CDC Type                 Exceptions               Endpoints  Safe  Unsafe  Unknown  No ASYNC_REG
--------  -----------  -----------------  ----------------------   ------------------------ ---------  ----  ------  -------  ------------
Critical  clk_166      div_by_2/CLK       No Common Primary Clock  Asynch Clock Groups            2     0       0        2             0
Warning   sys_clk      clk_83             No Common Primary Clock  Max Delay Datapath Only        9     9       0        0             9
Warning   sys_clk      clk_out1_mmcm      No Common Primary Clock  Asynch Clock Groups            9     9       0        0             9
Info      clk100       clk_out1_mmcm      Safely Timed             None                           8     8       0        0             0
```

129128!*note

© Copyright 2019 Xilinx

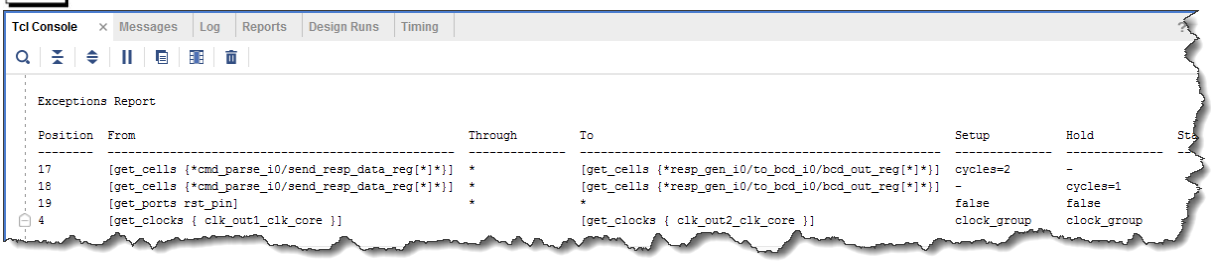
The `report_cdc` command performs a structural analysis of the clock domain crossings in your design. It gives you the information to identify potentially unsafe CDCs, which will lead to metastability or data coherency issues. It focuses on structures and their timing constraints, but does not provide information related to timing slack.

© Copyright 2019 Xilinx

Slide 12-13:

Report Exception

135133**slide



135133!*note

The Exceptions report is available via the GUI and the Tcl command `report_exceptions`.

The `report_exceptions` command has several modes of operation:

- Reporting the timing exceptions affecting the timing analysis
- Reporting the timing exceptions being ignored
- Reporting the timing exceptions coverage
- Reporting the invalid objects specified for the -from/-through/-to command line options
- Writing out the timing exceptions with only the valid objects
- Writing out the timing exceptions merged by the timing engine

The Exceptions report contains the following information:

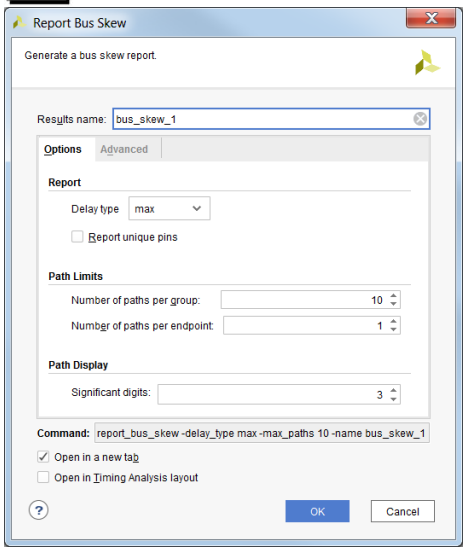
- The Position column indicates the constraint position number. This is the same position number reported by the Timing Constraint Window
- The From/Through/To columns indicate the patterns or objects specified with `-*from`/`-*through`/`-*to` command line options (including all the rise/fall versions of those options). An asterisk is displayed when the associated option was not specified.
- The Setup/Hold columns indicate whether the constraint applies to setup check, hold check, or both.

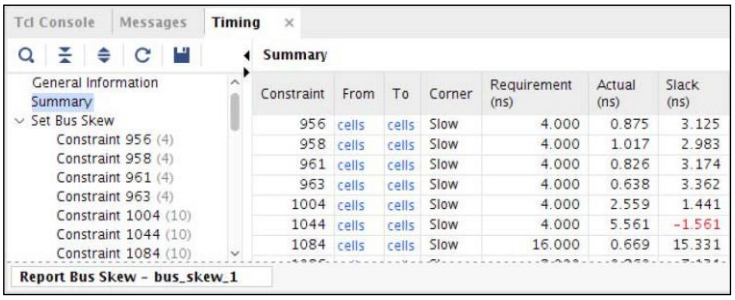
© Copyright 2019 Xilinx

Slide 12-14:

Report Bus Skew

135130**slide





135130!*note

The `report_bus_skew` command is used to generate the Bus Skew Report.

The bus skew report includes two sections:

19) A bus skew report summary.

20) A bus skew report by constraint.

© Copyright 2019 Xilinx

The Report Summary section reports all the set_bus_skew constraints defined in the design. The following information is reported for each constraint:

- Id: constraint Id referred later in the report (makes it easier to search for that constraint inside the report).
- From: pattern provided for the `set_bus_skew -from` option.
- To: pattern provided for the `set_bus_skew -to` option.
- Corner: corner (Slow or Fast) in which the worst bus skew was computed.
- Requirement: bus skew target value.
- Actual: worst bus skew computed across all the paths covered by the constraint.
- Slack: difference between the worst bus skew and the constraint requirement.

The Report Per Constraint section provides more details for each of the `set_bus_skew` constraints. Each reported constraint includes two parts:

21) Detailed summary of the paths covered by the constraint.

22) Detailed timing paths for the paths reported in the Per Constraint summary.

© Copyright 2019 Xilinx

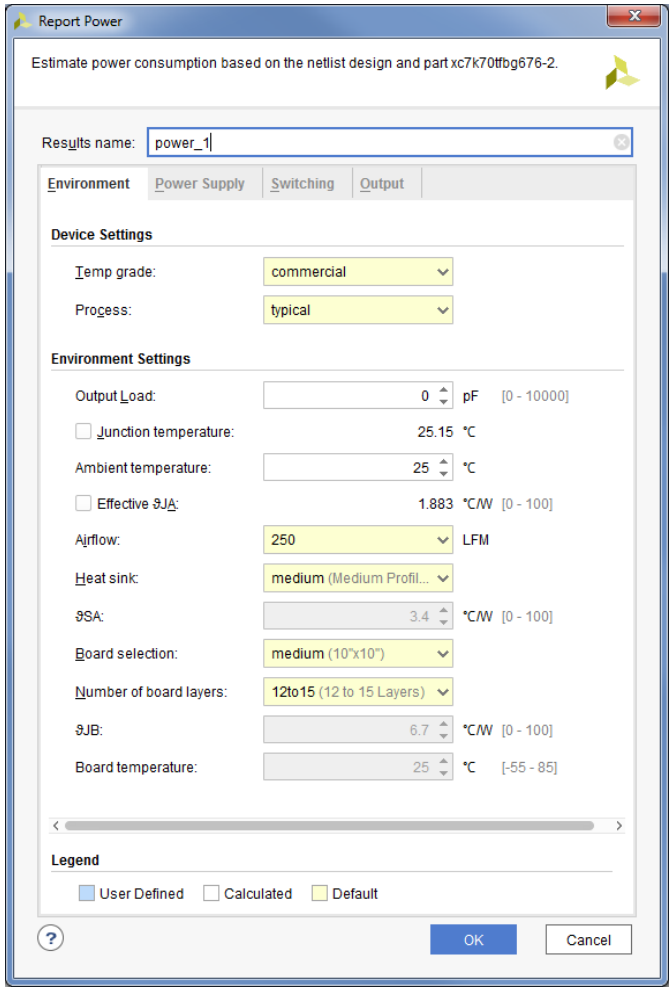
על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 12-15:

Report Power

129125**slide

- The `report_power` command runs power estimation and displays the report
- Generally used after routing stage
- Gives an idea about the power consumption based on current operating conditions of the device and switching rate of the design
  - Power consumption is calculated based on the junction and the ambient temperature
- Allows detailed power distribution analysis to guide power saving strategies and to reduce dynamic, thermal, or off-chip power
- Use the `set_operating_conditions` command to set operating conditions
- Use the `set_switching_activity` command to define switching activity

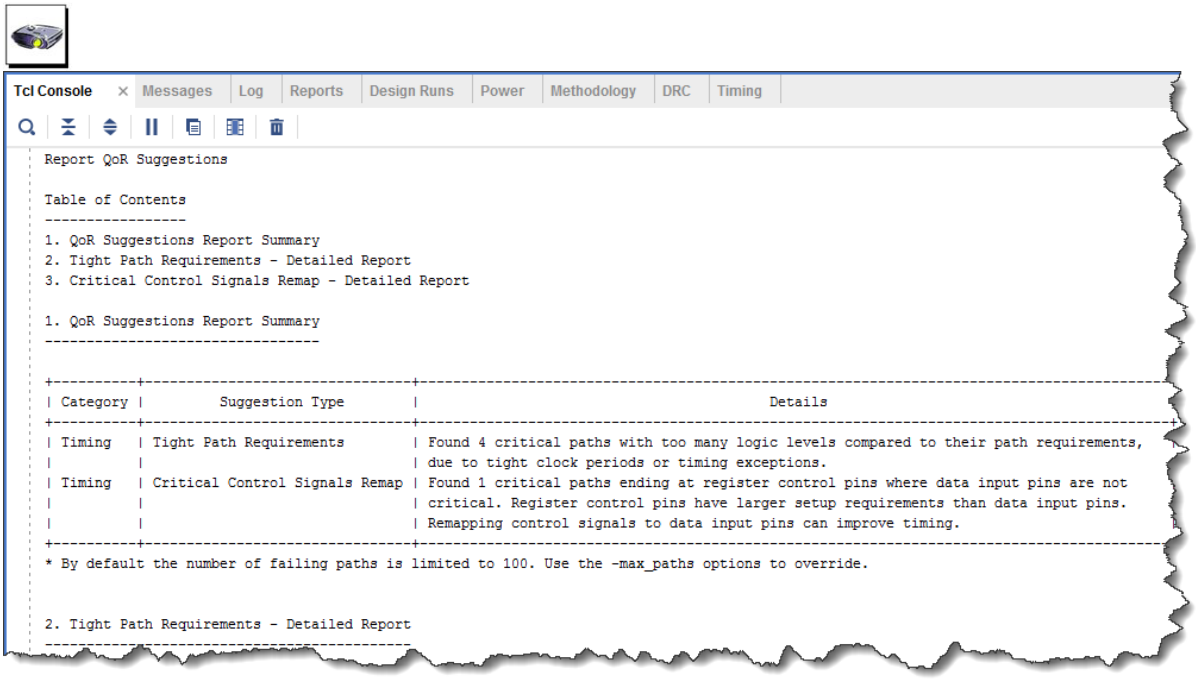


© Copyright 2019 Xilinx

Slide 12-16:

Report QoR Suggestions

135135**slide



135135!*note

The Report QoR Suggestions command is only available through the Tcl command line, using the Tcl command `report_qor_suggestions`.

By default, Report QoR Suggestions generates the QoR Suggestions Report Summary table, which is organized around three main categories; Congestion, Generic, and Timing.

By default, the QoR Suggestions report limits the analysis to the worst 100 violating timing paths.

RQA: `report_qor_assessment`

- Predicts the likelihood of meeting design goals
- Simple scoring with range of 1 to 5: 1 (will not meet), 5 (will meet)
- `report_qor_suggestions` provides suggestions to improve scores below 5

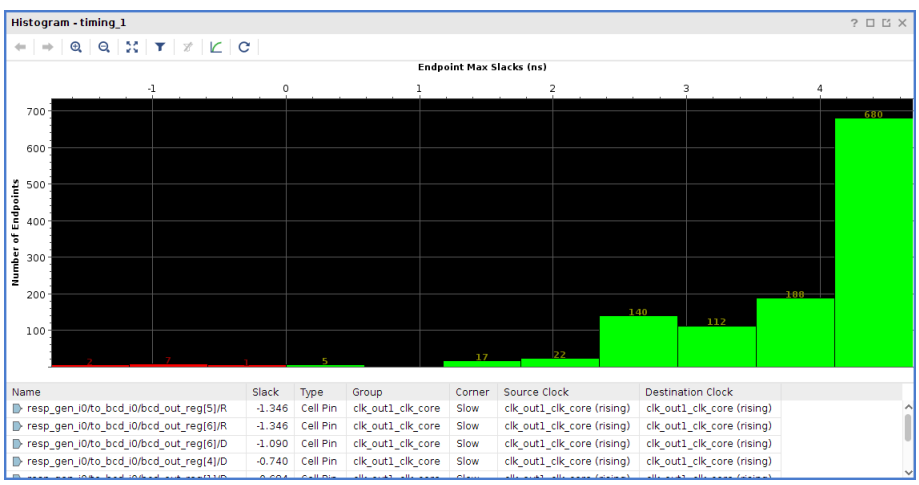
© Copyright 2019 Xilinx

Slide 12-17:

Slack Histogram

155183**slide

- The path slack histograms view the overall performance of timing paths in the logic design
- Vivado Design Suite allows users to create a slack histogram by selecting **Reports** > **Timing** > **Create Slack Histogram**
- Slack distribution provides a picture that helps in locating parts of the design that have the worst timing problems
- A histogram view consists of a bar graph on the left and an object table on the right
  - In the bar graph, the number at the top of the tallest bin indicates the number of paths in the bin
- When the histogram is generated, user can select a maximum or minimum delay (setup or hold) option
- **Note:** This command provides a graphical slack histogram that requires the tool to be running in GUI mode



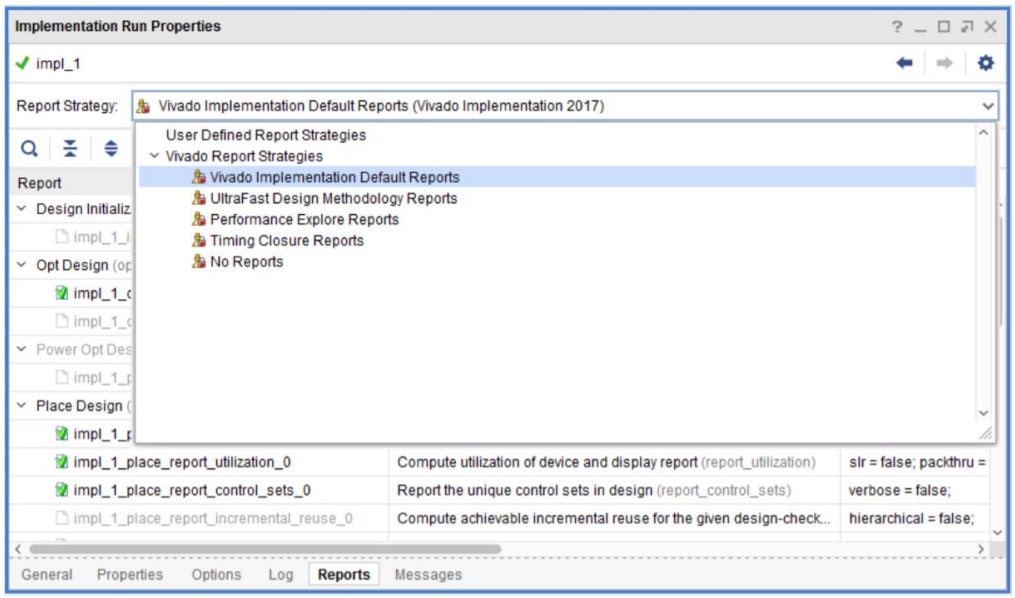
© Copyright 2019 Xilinx

Slide 12-18:

Configurable Report Strategies

136280**slide

- The Vivado IDE allows Configurable Report Strategies in project mode
- Configurable Report Strategies provide the ability to select which reporting commands are run after each step of the synthesis and implementation runs
- A different set of reports can be automatically generated, depending on the design stage, the design complexity and the user preferences
- The Vivado IDE also allows the creation of new report strategies using the settings
- Select the **Report Strategy** tab under **Design Runs** to change the report strategies



Slide 12-19:

Summary

113172**slide

© Copyright 2019 Xilinx

- The Vivado Design Suite creates different reports at different design stages
- Timing-related reports used for interactive and detailed timing analysis after synthesis or implementation
- Reports such as `report_drc` and `report_methodology` check the design against the rules and show violations

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Introduction to FPGA Configuration

Slide 13-1:

108859**slide

2019.1
108859!*note

This module provides meaningful context on how FPGAs can be configured. 15 minutes.

Slide 13-2:

Objectives

108860**slide

After completing this module, you will be able to:

- Describe the basic FPGA configuration process
- Describe the purpose of each of the FPGA configuration pins
- Create FPGA configuration files

© Copyright 2019 Xilinx

Slide 13-3:

Introduction: When Does Configuration Happen?

108862**slide

- When does configuration happen?
  - On power up
  - On demand
- Why do FPGAs need to be configured?
  - FPGA configuration memory is volatile
  - Configuration data is stored in a PROM or other external data source
- What do you need to know about FPGA configuration?
  - It is a process of loading application specific data into the internal memory
- What happens during configuration?
  - The mode is set via configuration pins and the desired device is selected for configuration

108862!*note

Configuration must occur after power up in order to achieve the desired application-specific functionality. Reconfiguration can also occur on-demand to completely reload a different bitstream or to reload the same bitstream again. Xilinx® FPGAs also support partial reconfiguration, which is similar in nature but operates on a region of the programmable logic rather than all of it. Some users take advantage of the reconfigurable nature of Xilinx FPGAs to the extent that they have a variety of full and partial bitstreams stored in their system. This is effectively time-multiplexing of hardware, which can provide many advantages including cost and power reduction.

As noted, the program storage technology is SRAM-based, so the programmable logic must be configured at least once every time power is cycled. There is a need to store the application-specific data in some form of non-volatile memory, and a number of solutions exist for doing so. A common solution is to store the data in flash memory, but it is also possible to store it on disk, or anywhere else.

© Copyright 2019 Xilinx

What do you need to know about configuration? The most important thing to know is that it is documented in the configuration user guide of each product, so there is no reason to try and commit all the details to memory. What you should focus on is a basic understanding of what takes place during configuration, as well as the available modes and their merits. It is also useful to be aware of what advanced features are offered, including their function and value.

Slide 13-4:

Configuration Pins – Basics

108863**slide

- Specific pins on the FPGA are used during configuration
- Xilinx FPGAs can load themselves through configuration pins
- Some pins act differently depending on the configuration mode
  - For example, CCLK is an output in some modes and an input in others
- Some pins are only used in specific configuration modes
- Some configuration pins are dual purpose
  - Dual purpose pins become user I/O after configuration is complete

108863!*note

Specific pins on the device are used during configuration. While the complete pin list and individual pin function varies by mode, there are certain pins which are universal. The interfaces used to configure an FPGA are the JTAG interface or the Synchronous serial interface.

© Copyright 2019 Xilinx

Slide 13-5:

Configuration Pins – Key Pins To Know (1)

108865**slide

- DONE
  - Open-drain, bidirectional pin
  - Indicates the completion of the configuration process
- DIN
  - Serial input for configuration data
- DOUT
  - Output to the next device in a daisy chain
  - Used in daisy chains only
- CFGBVS
  - Configuration bank voltage select; tells the device if the configuration bank I/O power supply requires the use of high voltage or low voltage I/O buffers

108865!*note

The DONE pin has several functions. Prior to the completion of the configuration sequence, DONE will be held low by the device to signal that configuration is not done. Its other function is for the synchronization of the transition from the end of the configuration sequence into user mode. This pin can be used by another device to align the start of user mode to an event, or can be used with multiple programmable logic devices by connecting their DONE pins, which coordinates them to enter user mode only after all of them have reached the end of the configuration sequence.

The CFGBVS pin is used to provide information to the device about what I/O bank voltage is used for the configuration pins. Xilinx devices are flexible, but to provide optimal switching thresholds, some device families require that users indicate if the I/O bank voltage for the configuration pins is high or low, within the range of allowed voltages.

© Copyright 2019 Xilinx

The POR_OVERRIDE pin is used to disable the default power-on-reset delay. Normally, as the power supplies are ramping, the device detects when they cross specific thresholds and then starts a counter. This counter is used as a power-on reset delay to allow additional time for the final portion of the power supply ramp before the configuration sequence continues.

However, if you are concerned about the time from power-on to configuration done, and you can guarantee the ramping behavior of your power supplies, you can disable the power-on reset delay generated by the counter. Using POR_OVERRIDE can save considerable time, but it is not a “casual” choice as it requires intimate understanding of the power supply design.

The most likely need for POR_OVERRIDE is to satisfy a design requirement covering the allowed time from power-up to configuration done, as is the case when the application includes PCIe® or CAN interfaces.

The PUDC_B pin selects the enable or disable of pull-ups during configuration on the User I/O pins. In some families, the pin with this functionality may be named some variation of HSWAP. This naming convention originated from a major use case of disabling pull-ups during configuration, which is hot swap.

In hot swap applications, you need ultra-low leakage current on I/O pins before you begin operation, and to achieve this you cannot have pull-ups or pull-downs. The pins need to be floating (or are sometimes even required to be pre-charged halfway between VCCO and GND) to avoid generating transients on the system signals to which the I/O pins are being connected while the system is powered, or “hot.”

Slide 13-6:

Configuration Pins – Key Pins To Know (2)

108864**slide

- Mode pins
  - Input pins that select which configuration mode is being used
- PROGRAM_B
  - Input that initiates configuration
  - Active Low

© Copyright 2019 Xilinx

- CCLK (configuration clock)
  - Input or output (depending on configuration mode)
  - Frequency up to 100 MHz (dependent on the FPGA; see the configuration user guide)

- INIT_B
  - Open-drain, bidirectional pin
  - Error and power stabilization flag
  - Active Low

108864!*note

The MODE pins (typically three of them) are used to tell the programmable logic which configuration mode is desired, based on the applied pattern. The MODE pins are sampled early in the configuration sequence.

The PROGRAM_B pin is used to initiate configuration. Although it is not called a reset pin, this is essentially what it is. When you pulse the PROGRAM_B pin, any currently loaded configuration is cleared and the programmable logic prepares to load a new configuration.

The CCLK pin runs the synchronous FPGA configuration sequence by default. The FPGA sources the configuration clock and drives CCLK as output.

The INIT_B pin has several functions, depending on where the programmable logic is with respect to the configuration sequence. Initially after power-on or subsequent PROGRAM_B pulse, INIT_B is held low by the device until it is ready to begin configuration. INIT_B can be held low by the user to further delay the start of the configuration sequence.

At the end of the configuration sequence, INIT_B can be held low by the device to signal a configuration error, such as a failure of the bitstream CRC check. After the configuration sequence is done, if the user has elected to include Readback CRC or the SEM IP core for SEU detection, INIT_B can be held low by the device – again, to signal a configuration error.

Slide 13-7:

© Copyright 2019 Xilinx

BIT File Generation

108866**slide

- write_bitstream Tcl command
  - Used to generate a bitstream (.bit) for configuration
  - Requires a fully routed DCP
  - The routed DCP defines the internal logic and interconnections for the FPGA design

- The Vivado device programmer
  - Used to configure FPGAs in the system directly from a host computer using a Xilinx download cable
  - Supports indirect programming

108866!*note

© Copyright 2019 Xilinx

The `write_bitstream` Tcl command or the Generate Bitstream button in the Flow Navigator generates a bitstream for Xilinx device configuration. After the design is completely routed, the device can be configured with a BIT file. A BIT file is a binary file with a *.bit* extension. The BIT file contains the configuration information from the fully routed DCP file.

The fully routed DCP file defines the internal logic and interconnections of the FPGA device together with device-specific information from other files associated with the target device. The binary data in the BIT file is then downloaded into the memory cells of the FPGA device or used to create a PROM file.

The Vivado device programmer allows you to program your device directly through the host computer. It also supports indirect programming. Indirect programming means programming devices such as flash devices.

Slide 13-8:

PROM File Generation

108867**slide

- In the Vivado Design Suite, PROM files are generated from bitstream files with the Tcl command `write_cfgmem`
- The `write_cfgmem` command is used to create the MCS programming file
  - This file will be used in programming the configuration memory device
- The `write_cfgmem` command automatically sizes the configuration memory file to the size of the bitstream
- The Vivado IDE supports the ability to chain multiple BIT files together via the `write_cfgmem` command

108867!*note

© Copyright 2019 Xilinx

The BIT file contains the configuration information from the NCD file. The NCD file defines the internal logic and interconnections of the FPGA device, together with device-specific information from other files associated with the target device.

The binary data in the BIT file is then downloaded into the memory cells of the FPGA device or used to create a PROM file. For more information, see Chapter 16 of the *Command Line Tools User Guide*, *PROMGen*.

Starting with Vivado Design Suite 2015.3, the configuration memory checksum verify capability is also available and Serial Vector Format (SVF) support is enabled.

© Copyright 2019 Xilinx

Slide 13-9:

Creating a Configuration Memory File

108868**slide

- Steps to generate the PROM file from the Vivado hardware manager
  1. Right-click the hardware target and select **Add Configuration Memory Device**
     - The Add Configuration Memory Device dialog box opens
  2. Select the appropriate configuration memory and click **OK**
     - The configuration memory will be added to the hardware target
     - After the configuration memory is added, the Vivado device programmer will prompt you program the memory
  3. Click **OK** in the dialog box
     - The Program Configuration Memory Device pop-up window opens and enables you to select the options required
  4. Click **OK** in the window

Slide 13-10:

Summary

111576**slide

- FPGAs need to be configured due to their volatile nature
- The bitstream is loaded each time through specific configuration pins

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Introduction to the Vivado Logic Analyzer

Slide 14-1:

```
108969**slide

2019.1
108969!*note
```

This module introduces the powerful and easy-to-use debug tool available within the Vivado® Design Suite. 10 minutes.

Slide 14-2:

Objectives

```
121336**slide
```

After completing this module, you will be able to:

- Describe what the Vivado logic analyzer (VLA) is, as well as the fundamental components that comprise the Vivado debug tool
- Enumerate the benefits of the Vivado logic analyzer
- Explain the basic probing flows to debug your design

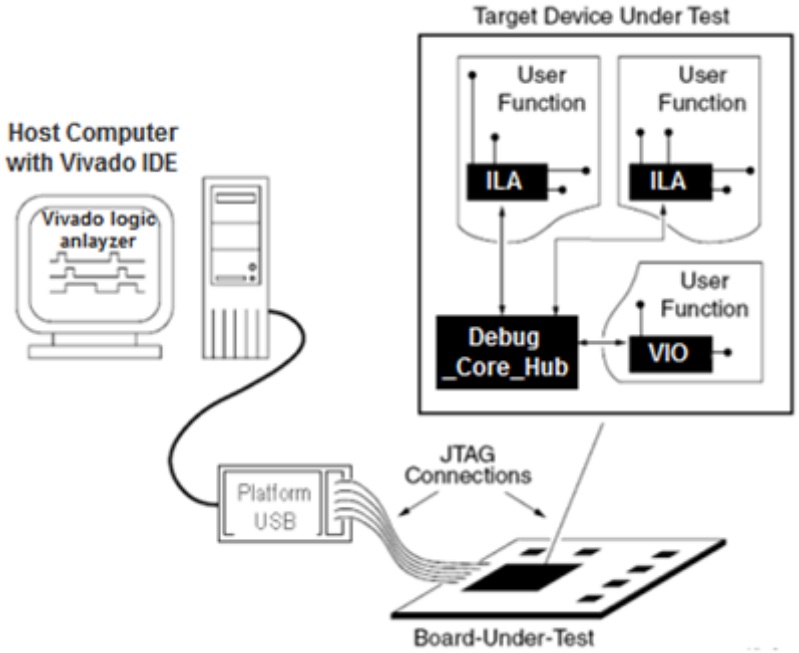
© Copyright 2019 Xilinx

Slide 14-3:

Logic Analyzer

121342**slide

- Designers can maximize productivity with a solid understanding of the underlying function of the Vivado debug tool
- Accelerate your success of design by
  - Reducing risk of issues
  - Rapidly locating problems
  - Fixing the bug quickly



121342!*note

Designers spends more than 40% of their time in debugging. The Vivado Logic Analyzer debug solution reduces the time required for verification and debugging.

© Copyright 2019 Xilinx

Slide 14-4:

Motivation for the Vivado Logic Analyzer Tool

121344**slide

- Functionality of a conventional logic analyzer can be built into any Xilinx FPGA
- Can implement the triggering and data collection capability of a logic analyzer using Vivado debug cores
- The Vivado debug cores do not consume any I/O pins
  - Captured data is sent to a PC via the FPGA's JTAG pins
- The Vivado logic analyzer allows different triggering scenarios and displays data in a variety of formats best suited to solving the problem at hand
- A single JTAG connection to the PC can be used for
  - Programming the programmable logic
  - Hardware debugging

Slide 14-5:

Vivado Logic Analyzer

121345**slide

- The Vivado logic analyzer is integrated into the Vivado Design Suite
- The Vivado logic analyzer is powerful yet easy to use
  - Extremely effective for both the verification and debugging of FPGA designs
  - Veteran FPGA designers appreciate the flexibility and power
  - Novices can quickly master the basics of this tool

© Copyright 2019 Xilinx

- The Vivado logic analyzer plays a synergistic role with simulation

- The Vivado logic analyzer can be used in three contexts
  - Verification: The process of determining the compliance of a given design to a (written) specification
  - Debugging: The process of finding and correcting the design for any discrepancy
  - Data capture: Used to collect real-world data for input into a simulation

121345!*note

The standalone VLA tools is a small download (1GB) and footprint size, and supports programming and logic and serial I/O debug. There is support for 64-bit and 32-bit OS platforms and a FlexNet license is not required for this standalone.

Both simulation and debugging play a fundamental role in the design process—it is not one or the other.

The co-play between the Vivado logic analyzer and a simulator can be an iterative design process where a specific behavior is being observed and verified, or used to provide simulation stimulus for succeeding stages in a complex design.

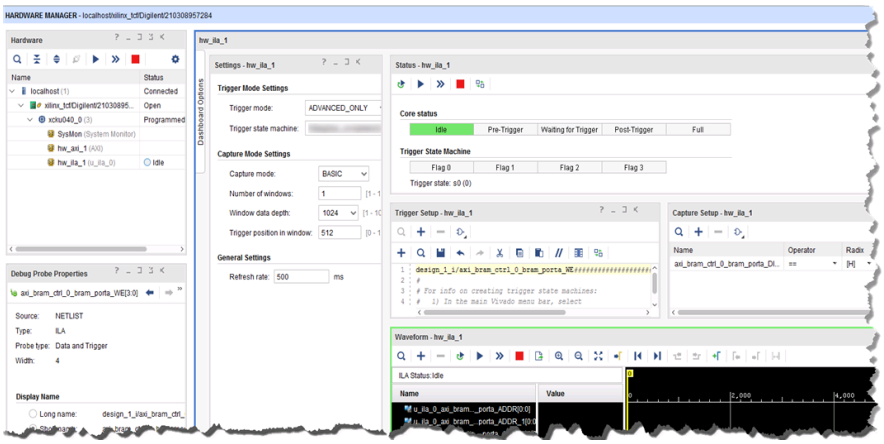
Slide 14-6:

Benefits of Logic Debug

108977**slide

© Copyright 2019 Xilinx

- The Vivado logic analyzer is comprised of two basic items
  - Debug cores that sample and inject signals into the programmable logic



  - Flows for generating and inserting these debug cores into a design and for controlling and viewing these cores once downloaded
- The Vivado logic analyzer captures data within an FPGA
  - These samples can be stored in a text file
  - This text file can be used as test vectors in a testbench
- Single trigger comparator type per PROBE
  - All comparison types, bit values
- Most debug parameters are set during run time
- Flexible, targeted probing of HDL design using MARK_DEBUG property
  - ILA probe information is extracted from the design and is stored in a data file that typically has an LTX file extension

© Copyright 2019 Xilinx

108977!*note

The Vivado logic analyzer is integrated directly into the Vivado Design Suite. The tool is opened when the Vivado Hardware Manager is opened and a debug core (ILA, VIO, JTAG-AXI) is present in the design.

There are four window panes that make up the basic GUI:

- Hardware: selecting the hardware component to configure
- Properties: setting configurable properties of selected logic analyzer core
- Dashboard: selecting ILA triggering and capturing options or VIO features
- Waveform window: graphic representation of captured signals

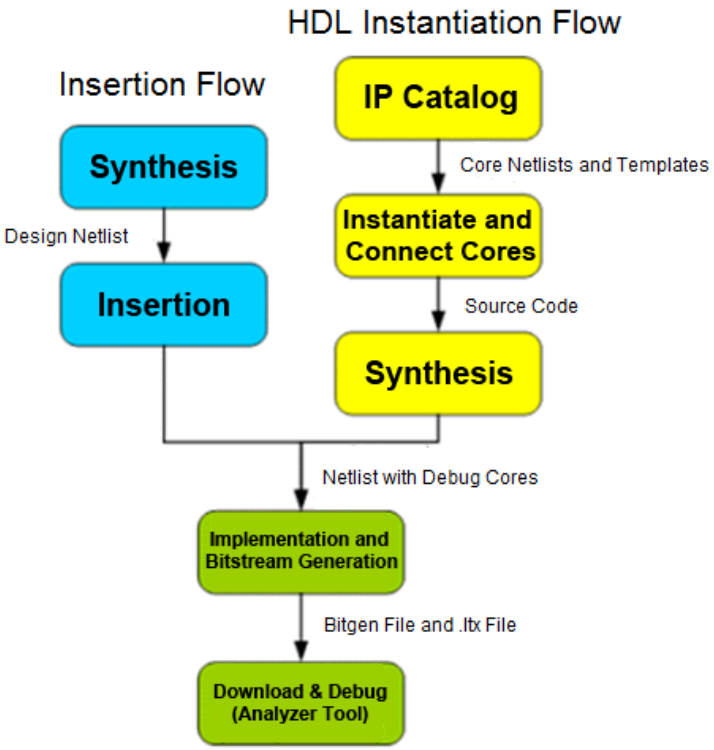
Slide 14-7:

Vivado Logic Analyzer Probing Flows

108978**slide

© Copyright 2019 Xilinx

- HDL instantiation flow
  - Generates any of the debug cores as netlists and templates via the IP catalog
  - Manual instantiation and connection required
- Netlist insertion flow
  - Limited to inserting ILA core
  - Inserts directly into the netlist; no re-synthesis required
- HDL instantiation flow and Netlist insertion flows can be mixed



108978!*note

Some Vivado logic analyzer users have an initial apprehension about inserting the debug core in their design, but once they are over it they get up to speed very quickly and become logic analyzer addicts who will put the cores in all their designs as a matter of course.

Statistics from Xilinx Support show that a majority of cases (~60%) are closed within a week and a further breakdown shows that ~40% are closed within three days. Many of the questions are from customers who are in the early stages of learning the tool and have simpler questions.

First, the design is created as usual. Then, with the design complete, the debug cores can be added in.

Netlist Insertion flow an excellent flow for situations when only the final netlist is available or only ILA cores need to probe signals at multiple levels of hierarchy. This is a fast and relatively painless way to view information inside the FPGA.

© Copyright 2019 Xilinx

The right (yellow) path represents the flow using the HDL instantiation approach. This flow is a bit more involved, but enables the designer to use the VIO core in addition to the ILA and other cores. More pre-planning is generally required to execute this approach.

Regardless of how the cores are inserted, after the final netlist is generated, the flow is identical to a non-logic analyzer flow through the implementation tools and bitstream generation.

The Vivado hardware manager can be used to download the bitstream to the FPGA. Scripts or the logic analyzer can then be used to interact with the debug cores in the FPGA.

© Copyright 2019 Xilinx

Slide 14-8:

HDL Instantiation or Insertion Flow?

108980**slide

- The HDL instantiation flow gives the user the flexibility to connect to any signal in the design
  - Involves manual customization, instantiation, and connection of debug core components
  - Debug cores supported are ILA, VIO and JTAG-to-AXI
- The user has to modify the HDL code in order to instantiate the cores in the design
  - You need to rerun synthesis, implementation, and generate bitstream after core insertion
  - Allows probing at the HDL design level only
- Netlist insertion flow is easier to add and remove the debug core, and is the more highly recommended flow
  - Allows probing at different design levels (HDL, synthesized design, system design)
- The Vivado tool automatically inserts the debug core into the post-synthesis netlist
  - Allows you to pick the nodes which you want to observe from the synthesized design
  - Requires only implementation and bitstream generation after core insertion

© Copyright 2019 Xilinx

Slide 14-9:

Summary

111593**slide

- The Vivado logic analyzer is extremely effective in the verification and debugging of FPGA designs
- Debug cores can implement triggering and data collection capability of the logic analyzer
- The logic analyzer is comprised of two main items
  - Debug cores
  - Flows for generating and inserting debug cores
- Insertion and HDL instantiation flows are the Vivado IDE debug probing flows

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Introduction to Triggering

Slide 15-1:

109178**slide

2019.1
109178!*note

The Vivado® logic analyzer is the primary way that designers will interact with the cores once they are inside the FPGA. This module illustrates various aspects of the ILA's trigger setup and conditions. 10 minutes.

Slide 15-2:

Objectives

109179**slide

After completing this module, you will be able to:

- Describe the trigger mechanism of the Vivado logic analyzer
- Explain how to use the *Run trigger* option
- Display captured data using the Waveform view

© Copyright 2019 Xilinx

Slide 15-3:

Triggering

109182**slide

- Triggering can be as simple as "capture data when this signal goes high"
- Triggers can also be set on
  - If the value on a bus is above or below
  - An appropriate signal occurs or fails
  - A group of signals matching one set of conditions and another group of signals failing to match another set of conditions
- Data can be displayed
  - Prior to the occurrence of the trigger
  - After the trigger
  - Both before and after the trigger
  - Showing multiple trigger events

Slide 15-4:

Triggering is the Key

111624**slide

- Triggering enables the ILA cores to begin capturing data at the proper time
  - Otherwise the designer would have to "get lucky" to see a specific event
- Proper triggering provides for
  - Finding a very specific event or series of events
  - Avoiding uninteresting data
- Triggers are set on an ILA-by-ILA basis
  - Each ILA has its own trigger sets

© Copyright 2019 Xilinx

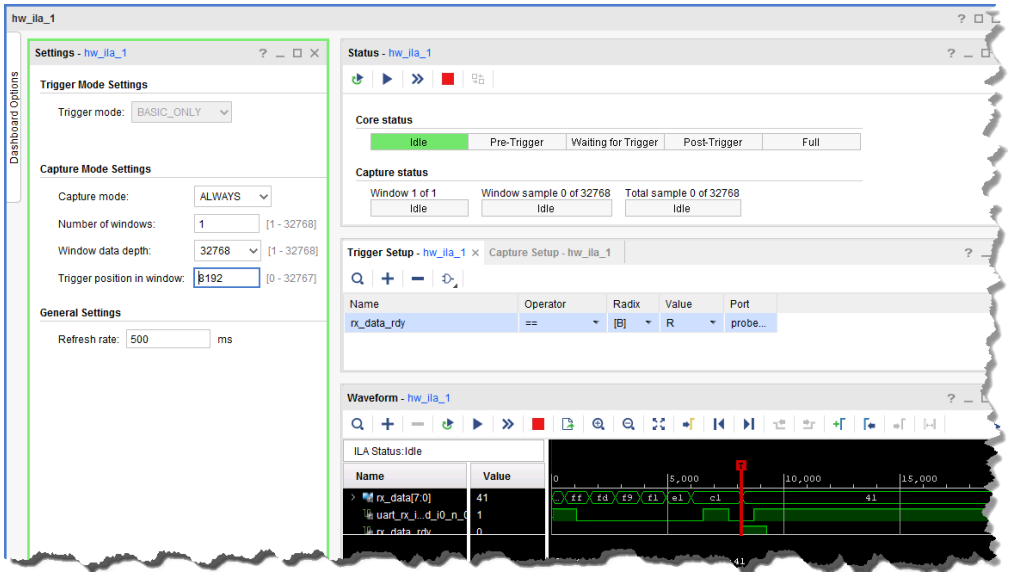
על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 15-5:

Setting Up Triggering

109181**slide

- Configuring the ILA cores
  - HDL instantiation flow
  - Netlist insertion flow
- Analyzer tool with Debug Probes window
- ILA dashboard
  - Setting the trigger mode
  - Setting the capture mode
  - Shows capture status
  - Trigger setup
  - Trigger state machine
  - Waveform
- One Trigger Setup window per ILA



109181!*note

© Copyright 2019 Xilinx

Once the settings have been selected, the hardware is generated and many aspects of the triggering are now fixed and immutable.

The analyzer enables the user to combine the capabilities defined during the core creation process (regardless of the flow) to tailor the trigger to the specific need.

This module deals with setting up the triggering from the perspective of the analyzer tool.

The figure provides an example of how a basic trigger can be constructed in the analyzer's ILA dashboard.

© Copyright 2019 Xilinx

Slide 15-6:

ILA Terminology

109180**slide

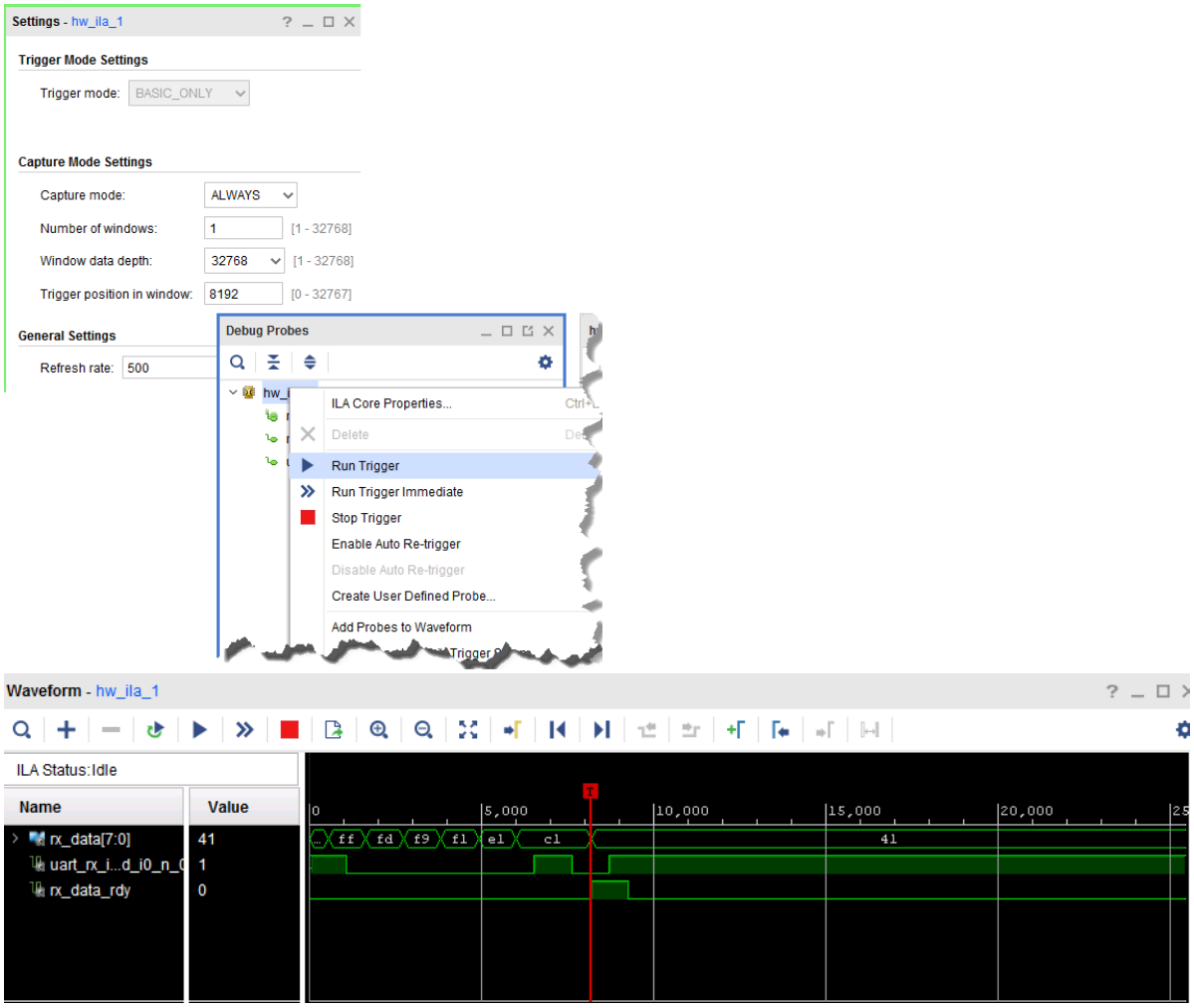
- Trigger port
  - Up to 1024 unique trigger ports
  - Up to 4096 signals per trigger port
  - Includes trigger width
- Match type and conditions
  - Specifies the sensitivity to a signal
  - Comparator used to detect events specified by the match type
- When **Run Trigger** is issued, the ILA core
  - Arms the trigger
  - Waits for the trigger
  - Displays in the waveform window

© Copyright 2019 Xilinx

Slide 15-7:

Displaying Captured Data

109183**slide



109183!*note

Setting up triggers is only half the effort. Sometimes just knowing that the trigger condition was met is sufficient, but more often than not you will need to see the data as well.

- Data can be presented graphically
  - Common Waveform view – logic analyzer

© Copyright 2019 Xilinx

- Graphical representation of the JTAG chain
  - All the debug tool cores in that chain
  - Only valid target devices and cores can be operated on
    - Devices must be configured before the cores with probes are seen

© Copyright 2019 Xilinx

Slide 15-8:

Summary

109184**slide

- The ILA core is capable of triggering on a wide variety of events, with tremendous flexibility in combining triggering events
- The Vivado logic analyzer leverages the ILA core's sampling capability
- Triggering
  - Enables tailoring for specific sensitivity
  - Supports the combining of triggering conditions
- Data can be displayed as waveform
  - Each ILA needs a separate waveform window

109184!*note

Where Can I Learn More?

- User guide
  - *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
- Tutorial
  - *Vivado Design Suite Tutorial: Programming and Debugging* (UG936)
- Further documentation
  - www.xilinx.com/products/design-tools/vivado/debug

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Debug Cores

Slide 16-1:

```
111604**slide

2019.1
111604!*note
```

This module introduces the debug cores such as the Integrated Logic Analyzer (ILA), Virtual Input Output (VIO), IBERT, and the debug core hub used in the Vivado® logic analyzer. 15 minutes.

Slide 16-2:

Objectives

```
111605**slide
```

After completing this module, you will be able to:

- Describe the ILA core and its properties
- Describe VIO and IBERT core usage
- Describe what the debug core hub is and why it is used
- Describe debug core hub insertion and customization

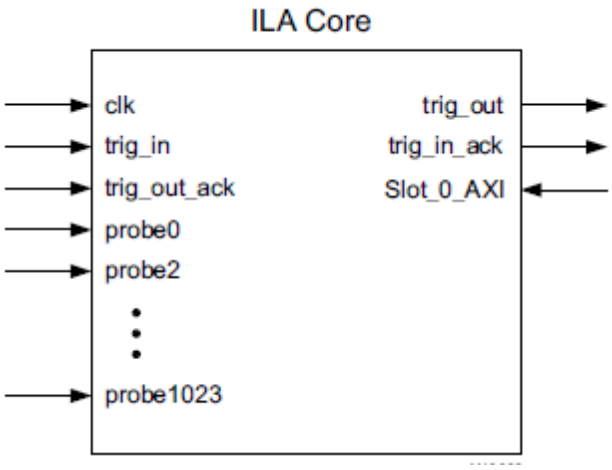
© Copyright 2019 Xilinx

Slide 16-3:

ILA Core

121373**slide

- The ILA debug core is used to monitor the design
  - Nets to debug are connected to the ILA core clock and probe inputs
- User configurable up to 1024 probe ports
- Up to 4096 signals per probe port



121373!*note

These signals, attached to the probe inputs, are sampled at design speeds and stored using on-chip block RAM. The core parameters specify the number of probes, trace sample depth, and the width for each probe input.

Communication with the ILA core is conducted using an auto-instantiated debug core hub that connects to the JTAG interface of the FPGA.

There are a maximum of 1024 probe ports on each ILA. Each probe port can consist of between 1 and 4096 signals.

The monitor type specifies which type of interface the ILA should be debugging. Currently the values for this parameter are native and AXI.

Slot_0_AXI of the ILA core is to be connected to the AXI interface. Currently, only one slot is supported for the ILA.

The input trigger port used in a process-based system such as the Zynq®-7000 SoC for embedded cross triggering. The port can also be connected to another ILA to create a cascading trigger.

© Copyright 2019 Xilinx

The output trigger can be generated either from a trigger condition or from an external trig_in port. There is run-time control from the logic analyzer to switch between the trigger condition and trig_in to drive trig_out.

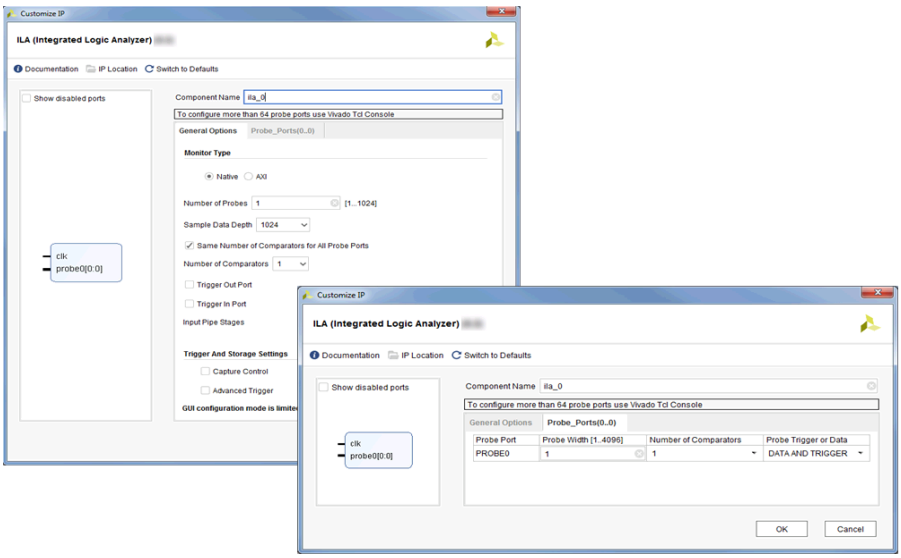
The trig_in_ack signal indicates to another ILA core, design, or processor that trig_in is properly received.

The trig_out_ack signal indicates to another ILA core, design, or processor that trig_out is properly received and causes the ILA to lower the trig_out signal on receiving trig_out_ack.

Slide 16-4:

ILA Core Wizard

121383**slide



- Steps the designer through the entering of all the parameters for the ILA
- Available in both flows
  - HDL instantiation flow
  - Netlist insertion flow
- User-selectable parameters specify
  - The component name
  - The number of probe ports
  - The sample data depth
  - The trigger and storage settings
  - The width for each probe input

121383!*note

© Copyright 2019 Xilinx

The ILA core includes many advanced feature of modern logic analyzers, including Boolean trigger equations and edge transition triggers. Because the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to the design are also applied to the components of the ILA core.

ILA Ports and Parameters:

- Clk - Design clock that clocks all trigger and storage logic.
- probe<n>[m-1 :0] - Probe port input. The probe port number <n> will be in the range from 0 to 1023. The probe port width (denoted by <m>) is in the range of 1 to 4096.
- trig_out - The trig_out can be generated either from a trigger condition or from an external trig_in port. There is run-time control from the logic analyzer to switch between the trigger condition and trig_in to drive trig_out.
- trig_in - Input trigger port used in process-based systems such as the Zynq-7000 SoC for embedded cross triggering. Can be connected to another ILA to create a cascading trigger.
- trig_out_ack - An acknowledgment to trig_out.
- trig_in_ack - An acknowledgment to trig_in.
- C_NUM_OF_PROBES - Number of ILA probe ports.
- C_DATA_DEPTH - Probe storage buffer depth. This number represents the maximum number of samples that can be stored at run time for each probe input.
- C_PROBE<n>_WIDTH - Width of probe port <n>, where <n> is the probe port having a value from 0 to 1023.
- C_TRIGOUT_EN - Enables the trig out functionality. Ports trig_out and trig_out_ack are used.
- C_TRIGIN_EN - Enables the trig in functionality. Ports trig_in and trig_in_ack are used.
- C_INPUT_PIPE_STAGES - Add extra flip-flops to the probe ports. One parameter applies to all of the probe ports.
- C_EN_STRG_QUAL - Enable the Capture (Storage) Qualifier. By enabling this you can specify the capture condition in the Vivado logic analyzer and thus capture the probes selectively.
- C_ADV_TRIGGER - Enables the advance trigger option. This enables the trigger state machine and allows you to write your own trigger sequence in the Vivado logic analyzer.

© Copyright 2019 Xilinx

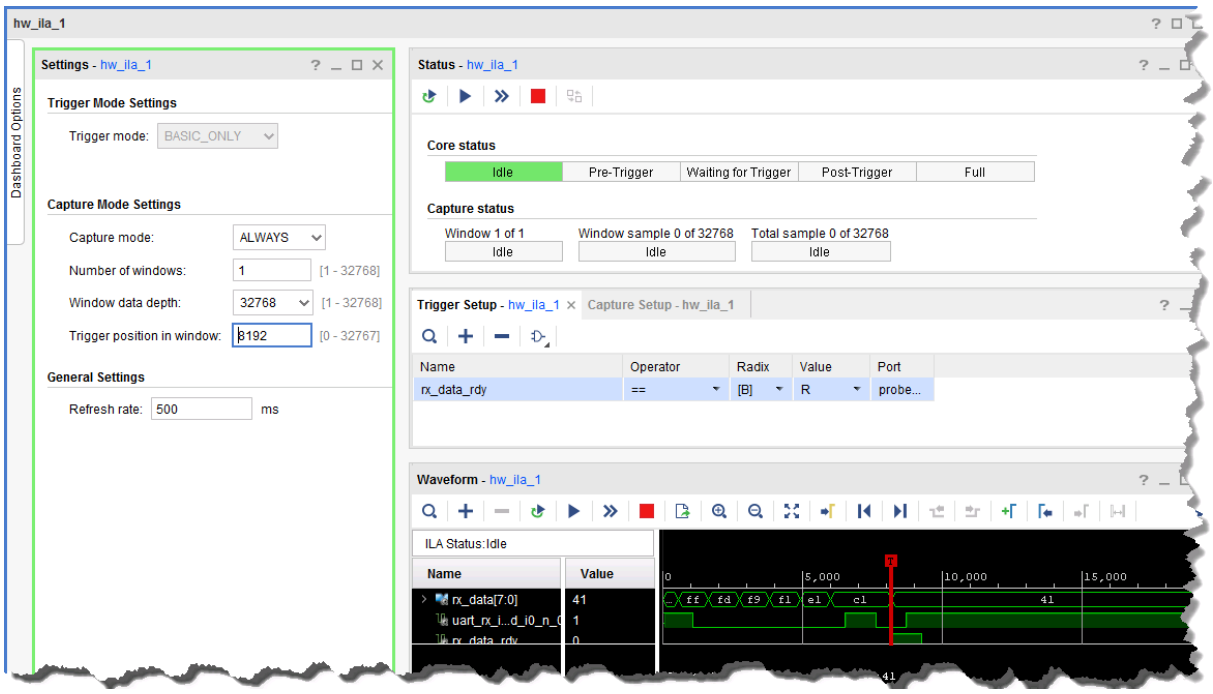
על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 16-5:

ILA Dashboard

113813**slide

- The ILA dashboard shows all status and control information pertaining to a given ILA core
  - Trigger mode settings
  - Capture mode settings
  - Status
  - Capture setup
  - Trigger setup
  - Waveform



113813!*note

Set the trigger mode to trigger on various events in hardware:

- BASIC_ONLY: The ILA Basic Trigger mode can be used to trigger the ILA core when a basic AND/OR functionality of debug probe comparison result is satisfied.
- ADVANCED_ONLY: The ILA Advanced Trigger mode can be used to trigger the ILA core as specified by a user-defined state machine.

© Copyright 2019 Xilinx

- TRIG_IN_ONLY: The ILA TRIG_IN Trigger mode can be used to trigger the ILA core when the TRIG_IN pin of the ILA core transitions from low to high.
- BASIC_OR_TRIG_IN: The ILA BASIC_OR_TRIG_IN Trigger mode can be used to trigger the ILA core when a logical OR-ing of the TRIG_IN pin of the ILA core and BASIC_ONLY trigger mode is desired.
- ADVANCED_OR_TRIG_IN: The ILA ADVANCED_OR_TRIG_IN Trigger mode can be used to trigger the ILA core when a logical OR-ing of the TRIG_IN pin of the ILA core and ADVANCED_ONLY trigger mode is desired.
- Set the trigger output mode.
- Use ALWAYS and BASIC capture modes to control filtering of data to be captured.
- Set the number of ILA capture windows.
- Set the data depth of the ILA capture window.
- Set the trigger position to any sample within the capture window.
- Monitor the trigger and capture status of the ILA debug core.

Programming an FPGA device prior to debugging with the .bit file that contains the debug cores involves several settings.

- In ILA Properties settings
  - Trigger position: Used to set the position of the trigger mark in the captured data buffer.
  - Data depth: Used to set the data depth of the ILA core captured data buffer.
- You can also change ILA core settings in the ILA Cores tab of the Basic Trigger Setup window.
  - Trigger condition: Use the condition to select "AND" and "OR" settings which causes a trigger event.

The Debug Probes window shows the probes associated with each ILA core.

Basic trigger setup comparators are used to detect equality and inequality conditions on probe inputs.

The Compare Value dialog box allows you to configure:

- Operator
- Radix
- Value

An interface for interacting with the ILA and VIO is available. The Vivado logic analyzer supports simultaneous multiple ILA waveform viewers.

The ILA core has several possible states:

© Copyright 2019 Xilinx

- Idle: The ILA core is idle and waiting for its trigger to be armed. The ILA core is idle and waiting for its trigger to be run. If the trigger position is 0, then the ILA core will transition to Waiting for Trigger status once the trigger is run, otherwise the ILA core will transition to Pre-Trigger status.
- Pre-Trigger: The ILA core is capturing pre-trigger data into its data capture window. Once the pre-trigger data has been captured, the ILA core will transition to Waiting for Trigger status.
- Waiting for Trigger: The ILA core trigger is armed and is waiting for the trigger event to occur as described by the basic or advanced trigger settings. Once the trigger occurs, the ILA core will transition to Full status if the trigger position is set to the last data sample in the capture window; otherwise it will transition to Post-Trigger status.
- Post-Trigger: The ILA core is capturing post-trigger data into its data capture window. Once the post-trigger data has been captured, the ILA core will transition to Full status.
- Full: The ILA core capture buffer is full and is being uploaded to the host for display. The ILA core will transition to Idle status once the data has been uploaded and displayed.

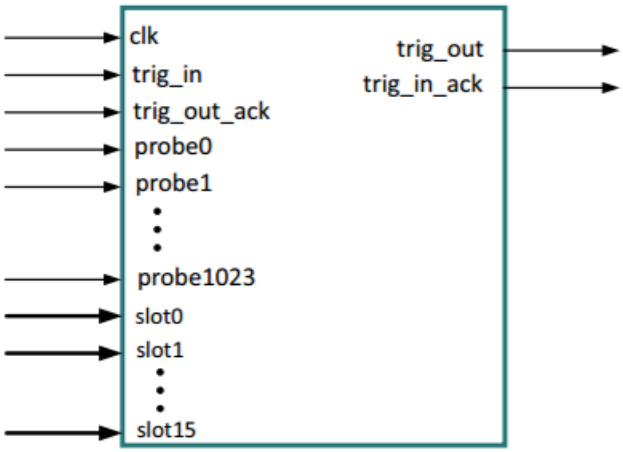
Slide 16-6:

System ILA Core

121443**slide

© Copyright 2019 Xilinx

- The System ILA debug core is used to monitor
  - The internal signals and interfaces of a design
- Allows in-system debugging of post-implemented designs
- User configurable up to 1024 probe ports
- Up to 4096 signals per probe port
- Up to 16 interface slots



121443!*note

Signals and interfaces in the FPGA design are connected to the System ILA probe and slot inputs. These signals and interfaces, attached to the probe and slot inputs respectively, are sampled at design speeds and stored using on-chip block RAM (BRAM).

There are a maximum of 1024 probe ports on each System ILA debug core. Each probe port can consist of between 1 and 4096 signals.

The monitor type specifies which type of interface the System ILA debug core should be debugging. Currently the values for this parameter are native and interface.

Slot_0_AXI of the System ILA core is to be connected to the AXI interface. The System ILA core supports up to 16 interface slots.

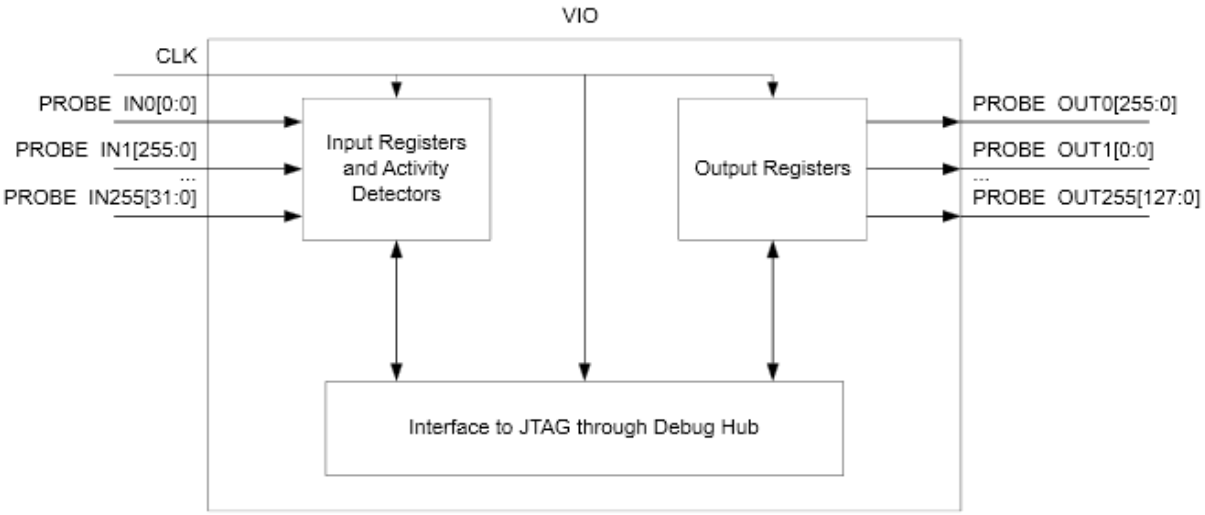
Slide 16-7:

Virtual Input/Output (VIO) Core

113814**slide

© Copyright 2019 Xilinx

- Presents and captures data on the design clock attached to the CLK input port to the sampled data
- Insert "virtual" pins into your design
- Can produce up to 65 input and output probes
  - Each port can be 256 bits wide



113814!*note

The VIO core enables you to place virtual input/output cores within an FPGA design.

Inputs and outputs can be visualized in different ways, not only as DIP switches and LEDs. These cores can be used as virtual LEDs, pushbutton, or toggle switches—added at the last minute to an FPGA design.

These cores can also stimulate a design, mimicking the activity of logic that is not yet completed.

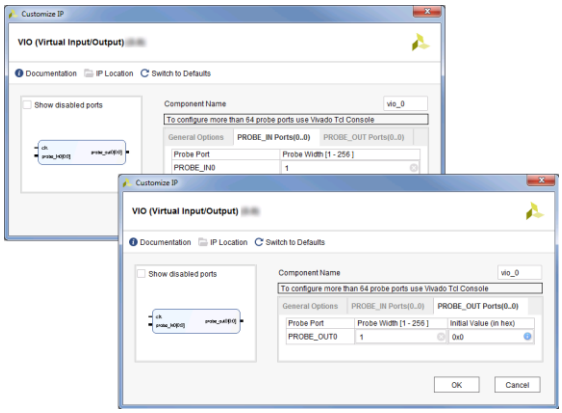
© Copyright 2019 Xilinx

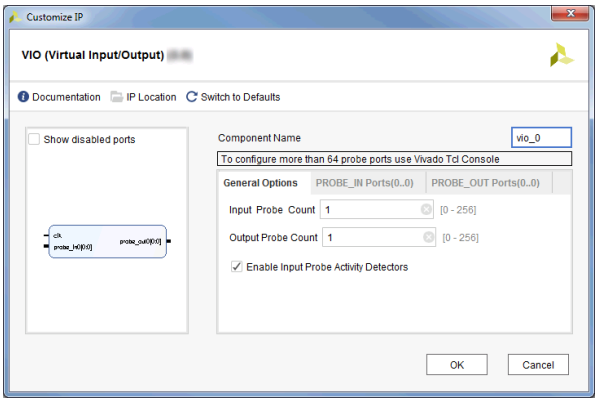
Slide 16-8:

VIO Core Wizard

121388**slide

- Steps the designer through entering all the parameters for the VIO
- Available in
  - HDL instantiation flow
- User-selectable parameters specify
  - The component name
  - The number of input and output probe on VIO core
  - The width for each probe port





121388!*note

- Inputs are virtual LEDs and other status indicators
  - Activity detectors (if multiple transitions between displays) on input ports to detect rising and falling transitions between samples
- Outputs are virtual DIP switches and other controls
  - Force individual signals
  - Custom output initialization

The VIO core can be added via the IP catalog in the HDL instantiation flow. In the GUI, the valid range for input and output probe count is 0 to 64. If you want to add more probe ports, you will need to use the Tcl command flow to generate the VIO core. Run time resets the VIO core to its initial values.

© Copyright 2019 Xilinx

Slide 16-9:

VIO Ports and Parameters

113816**slide

- VIO core I/O port signals

Signal Name	I/O	Description
CLK	I	Design clock used to register input ports and output ports. This is required
PROBE_IN<m>[<n> - 1:0]	I	Input probe port number <m> (where <m> can be 0 to 256) of width <n> (where <n> can be 1 to 256). For a 1-bit wide port, use PROBE_IN<m>[0:0]
PROBE_OUT<m>[<n> - 1:0]	O	Output probe port number <m> (where <m> can be 0 to 256) of width <n> (where <n> can be 1 to 256). For a 1-bit wide port, use PROBE_OUT<m>[0:0]

113816!*note

More information on the General Options Panel of VIO core generation is provided below:

Input Probe Count – Use this text field to select the number of input probe ports on the VIO core. The valid range used in the GUI is 0 to 64. If you need more than 64 input probe ports, you need to use the Tcl command flow to generate the VIO core. At least one input or output probe port needs to be specified.

Output Probe Count – Use this text field to select the number of output probe ports on the VIO core. The valid range used in the GUI is 0 to 64. If you need more than 64 output probe ports, you need to use the Tcl command flow to generate the VIO core. At least one input or output probe port needs to be specified.

PROBE_IN Ports panels – Each PROBE_IN Ports panel has up to 16 ports.

PROBE_OUT Ports panels – Each PROBE_OUT Ports panel has up to 16 ports.

Probe Width– Use the Probe Width field to set the width of each probe port. The valid width range is 1 to 256 bits wide.

Initial Value – Use the Initial Value field to set the initial value of the output probe port to a specific value. The value is to be specified in hexadecimal format with a “0x” prefix.

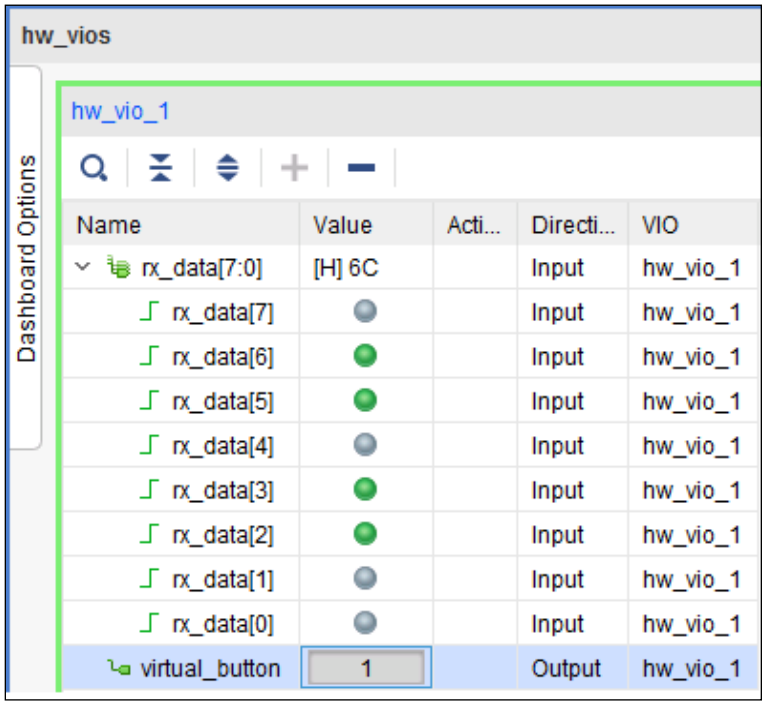
© Copyright 2019 Xilinx

Slide 16-10:

VIO Dashboard

113818**slide

- The VIO dashboard shows all status and control information pertaining to a given ILA core
  - Drag-and-drop VIO probes from the Debug Probes window
  - Monitor "virtual" pins in your design
- Inputs appear as
  - LEDs
  - Text
- Outputs appear as
  - Toggle buttons



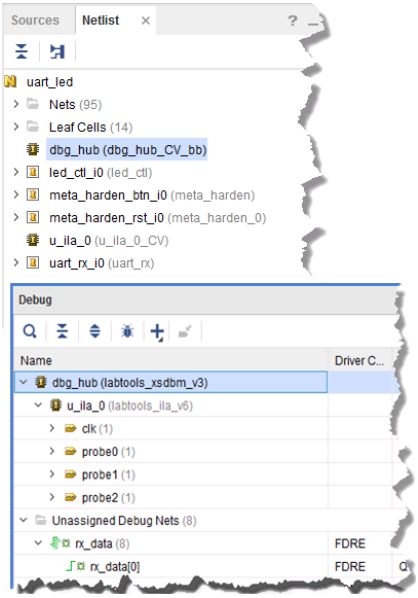
© Copyright 2019 Xilinx

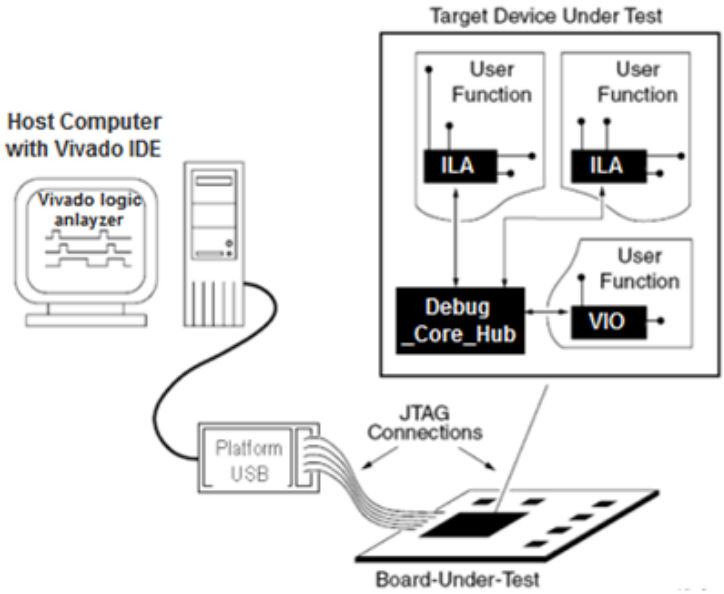
Slide 16-11:

Debug Core Hub

111603**slide

- The debug core hub is used to connect debug cores to the JTAG scan chain
  - ICON core is no longer required for new debug cores; i.e., ILA v2.x, VIO v2.x
- Only one debug_core_hub is required in a design
  - Automatically generated and connected into the synthesized design netlist
- Debug cores are connected to provide connectivity
- User can view and change the BSCAN user scan chain of the debug core hub





111603!*note

Ensure that the clock going to the dbg_hub is a free running clock. Failure to do so could result in an inability to communicate with the debug core when the design is loaded onto the device. You can use the connect_debug_port Tcl command to connect the clk pin of the debug hub to a free-running clock.

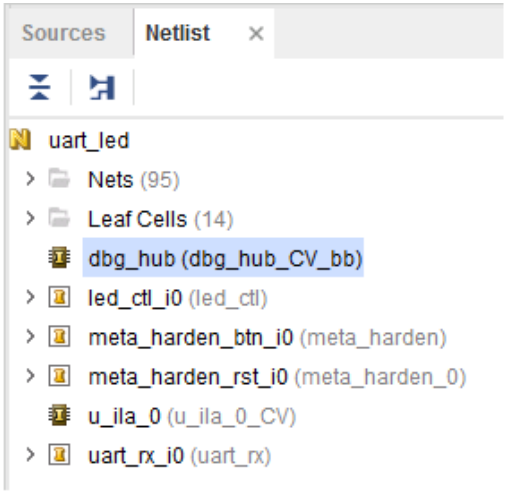
© Copyright 2019 Xilinx

Slide 16-12:

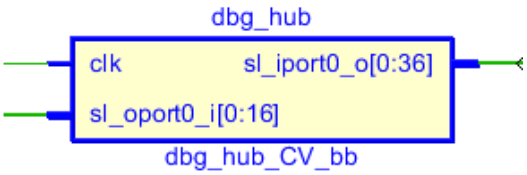
Debug Core Hub Insertion

111606**slide

- Netlist updated to include core interfaces



- Netlist and Schematic views display new cores
  - Black-box module **debug_core_hub** displayed in Netlist view (empty instance)
    - Not implemented yet
  - Cores are configured



- All probed signals are connected up to the top level of the design
  - Ports are added to logic hierarchy

111606!*note

© Copyright 2019 Xilinx

The debug core hub is automatically inserted to provide connectivity between the ILA core and the JTAG scan chain.

Debug cores are inserted into the netlist initially as black boxes because they have not been "implemented" yet. All views in the Vivado IDE will reflect this by having the debug core instances displayed with the black box icon, which indicates the netlist is still empty below the ILA and debug core hub cores.

The debug cores are instantiated at the top level of the design hierarchy, and ports are created with signal wires inserted into the netlist to connect up to the debug cores.

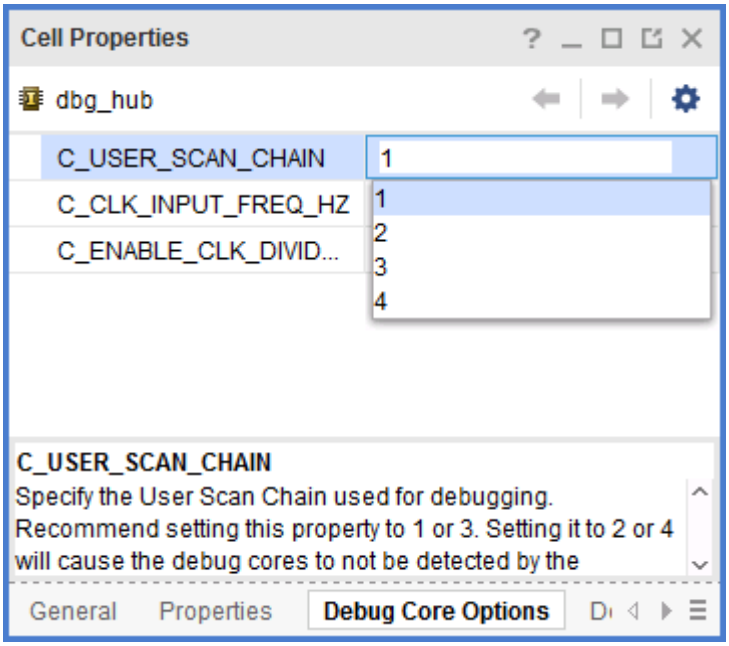
© Copyright 2019 Xilinx

Slide 16-13:

Debug Core Hub Customization

111609**slide

- View or change the BSCAN user chain index of debug_core_hub
  - Select **debug_core_hub** to open the Cell Properties window and click **Debug Core Options**
  - View or change the C_USER_SCAN_CHAIN property



111609!*note

To mix legacy ICON, ILA, and/or VIO v1.x cores with ILA v2.x cores, you will need to set the C_USER_SCAN_CHAIN property of the debug_core_hub to a user scan chain that does not conflict with the ICON v1.x core's Boundary Scan Chain setting. Failure to do so will result in errors later in the implementation flow.

Slide 16-14:

Debug Bridge

129369**slide

- The debug bridge IP core is a controller
  - Provides connectivity to debug hub and debug cores

© Copyright 2019 Xilinx

- Enables remote debugging over a Xilinx Virtual Cable (XVC) server running on a hardware device
- Enables tandem debug with field update designs in a Partial Reconfiguration flow

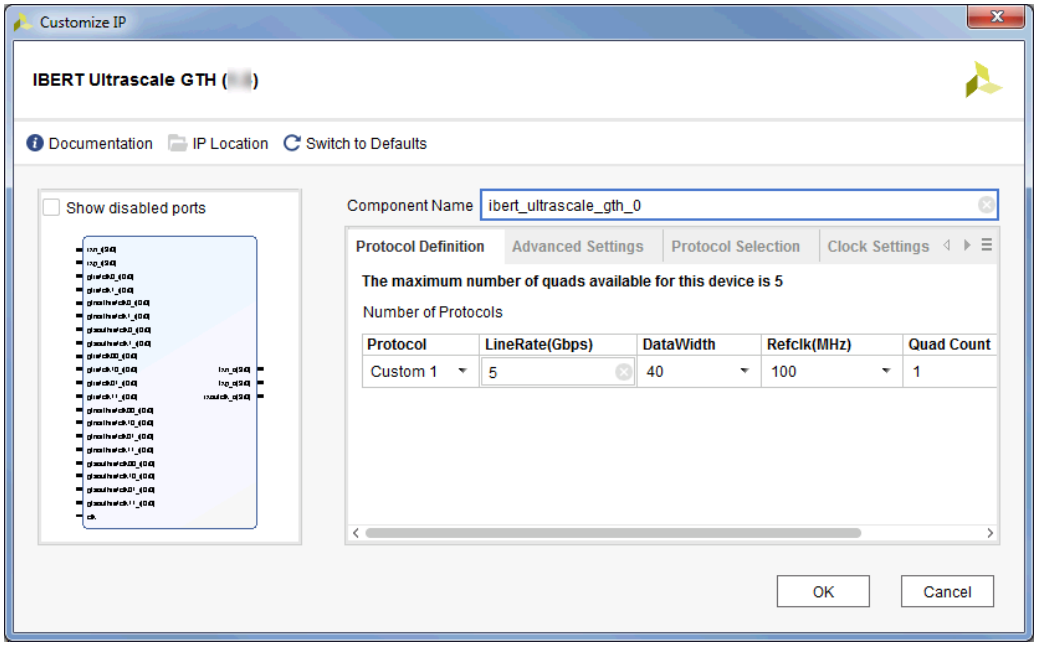
© Copyright 2019 Xilinx

Slide 16-15:

Integrated Bit Error Ratio Tester (IBERT) Core

113819**slide

- Debugs, verifies, and optimizes high-speed serial communication channels
- Provides access to the multi-gigabit transceivers
- Performs bit-error ratio analysis on channels composed of MGT/GTP/GTX/GTH components
- Is a "wrapper" for transceivers
  - User design cannot operate simultaneously with the IBERT
- Analyzer tool provides user-adjustable parameters to test various configurations



113819!*note

Another debug core IP In-System IBERT, is used to debug transceivers in the design. This IP performs 2D eye scans of UltraScale and UltraScale+ transceivers in the design.

© Copyright 2019 Xilinx

Slide 16-16:

IBERT Sweep Function

113820**slide

- IBERT parameter sweep function for 7 series FPGA GTP, GTX, and GTH transceivers
  - Enables you to specify TX and RX parameters to sweep through to find optimal transceiver settings
  - Provides a means to determine phase margin by automatically sweeping through all RX phase sampling points
  - Saves all parameter sweep results to a file for post processing
  - Saves results in graphical eye scan display

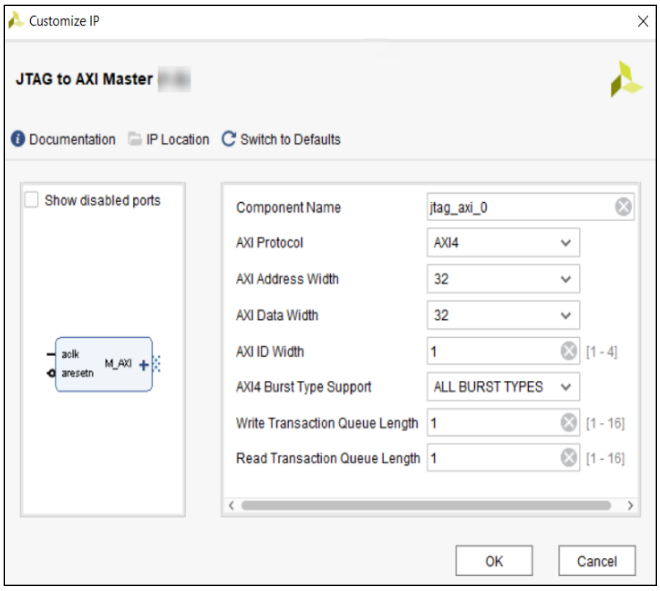
Slide 16-17:

JTAG to AXI Master

155202**slide

© Copyright 2019 Xilinx

- The JTAG to AXI Master debug feature is used to generate AXI transactions
- AXI transactions interact with various AXI Full and AXI Lite slave cores in a system that is running in hardware
- Core supports all memory-mapped AXI and AXI-Lite interfaces and can support 32- or 64-bit wide data interfaces
- Xilinx recommends using this core to generate AXI transactions and debug/drive AXI signals internal to an FPGA at run time



© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Netlist Insertion Debug Probing Flow

Slide 17-1:

```
109093**slide

2019.1
109093!*note
```

This module introduces the netlist insertion debug probing flow of the Vivado® Logic Analyzer. Core insertion is the process of selecting signals from within a netlist for monitoring and producing an updated netlist that contains the appropriate ILA cores. 15 minutes.

Slide 17-2:

Objectives

```
109094**slide
```

After completing this module, you will be able to:

- Describe the process of inserting the ILA core into a synthesized design netlist in the Vivado® IDE
- Identify the strengths and weaknesses of the netlist insertion flow
- Use Tcl commands to insert debug cores and analyze the design

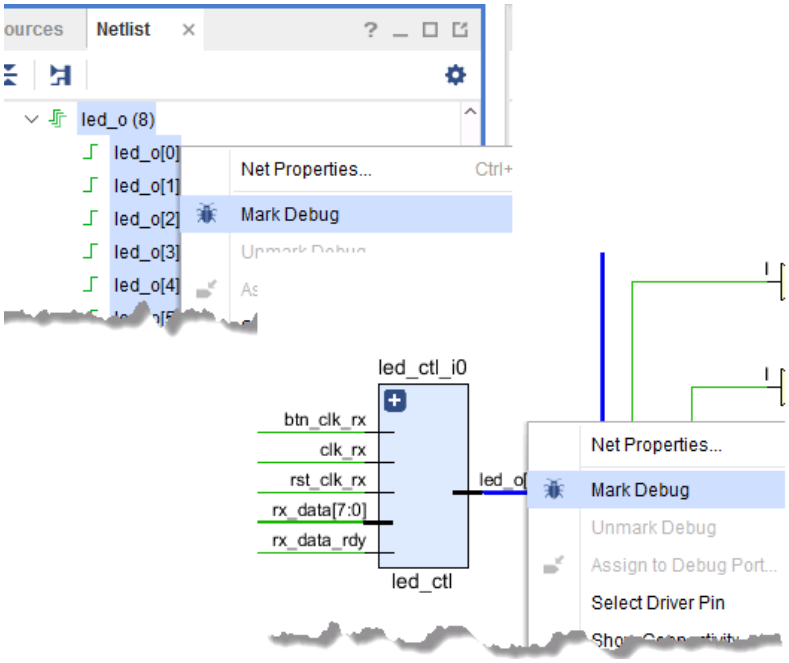
© Copyright 2019 Xilinx

Slide 17-3:

Netlist Insertion Flow

121350**slide

- Netlist insertion flow inserts the debug cores to the synthesized netlist
- Insertion of debug cores into the Vivado Design Suite involves
  - A wizard that creates and configures ILA cores
  - The Debug window to control debug cores, ports, and properties



- Select the nets in the Vivado Design Suite from
  - The netlist view
    - Each level of logic hierarchy
  - The schematic view
  - The Find results
- Right-click the nets and select **Mark Debug**
- Netlist insertion flow is the most flexible flow
  - Allows probing at different design levels

121350!*note

The Vivado IDE offers cross-selection from any view and a GUI operation that allows you to select a net object. You can select a net and connect it to debug cores for probing with the debug tools. This includes the Netlist and Schematic views.

© Copyright 2019 Xilinx

When a net is selected and Mark Debug is used, this will add the net name to a temporary folder which will be looked at by the Set Up Debug Wizard. You can use this folder to select nets of interest on-the-fly as you are analyzing your design connectivity.

The Set Up Debug Wizard will give you the option of using the unassigned nets folder nets if there are any. There is also a find capability to let you search for nets of interest within the dialog box.

The nets are added into *Unassigned Debug Nets* in the Debug tab view

Slide 17-4:

Using the mark_debug Attribute

109077**slide

- Mark nets to debug in the HDL source level
  - Identify signals prior to synthesis by using the `mark_debug` constraint
- Vivado synthesis `mark_debug` examples
  - VHDL syntax

```
attribute mark_debug : string;
attribute mark_debug of char_fifo_dout: signal is "true";
```

  - Verilog syntax

```
(* mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

- Add the `mark_debug` attribute to the target XDC file

```
set_property mark_debug true [get_nets sine*]
```

  - Use the `mark_debug` attribute in XDC in a synthesized design only

109077!*note

© Copyright 2019 Xilinx

The procedure for marking nets for debug depends on whether you are working with an RTL source-based project or a synthesized netlist-based project.

For an RTL netlist-based project, you can use the Vivado synthesis feature to optionally mark HDL signals for debug by using the `mark_debug` constraint in VHDL and Verilog source files. The valid values for the `mark_debug` constraint are “TRUE” or “FALSE.” The Vivado synthesis feature does not support the “SOFT” value.

For a synthesized netlist based project, you can use the `mark_debug` attribute without editing the HDL; that is, by manually adding the attribute on the nets for debug in the XDC file. Alternatively, you can add `mark_debug` on the nets to probe via the synthesized netlist view and save the attributes in the XDC automatically.

© Copyright 2019 Xilinx

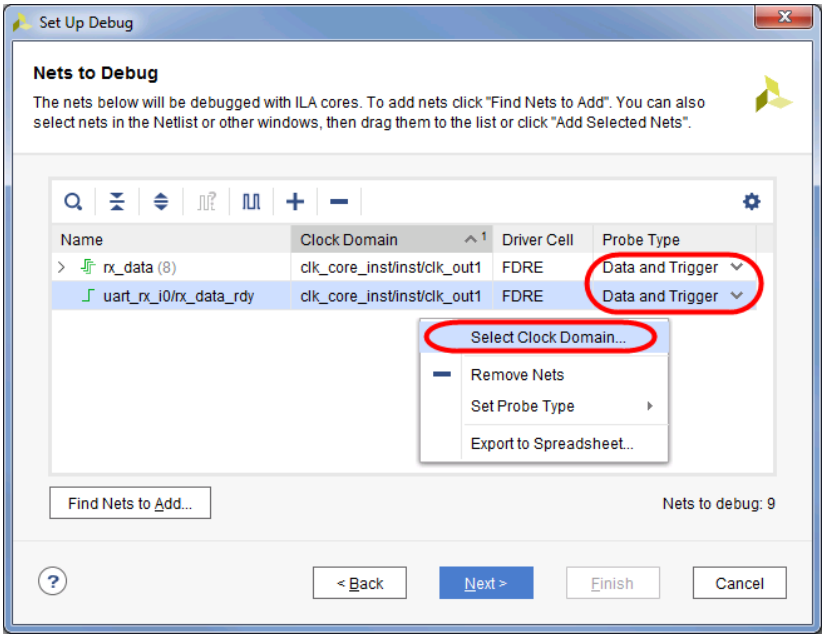
Slide 17-5:

Set Up Debug Wizard to Insert Debug Cores

109078**slide

- Select **Tools** > **Set Up Debug**

- All unassigned nets appear in the Specify Nets to Debug pane
  - Use the **Add/Remove Nets** button to modify the selected nets list
    - Net filtering
    - Add new nets based on name search
  - Display and select the clock domain
  - Displays TRIG, DATA, or both



- Confirmation summary
  - Number of cores, clocks

109078!*note

The Vivado tool core insertion process is made easier by the Set Up Debug wizard. This wizard guides you through the options, including pre-populating the list of debug signals with any pre-selected unassigned nets.

© Copyright 2019 Xilinx

There is a button to add or remove any nets from this list of nets for a final choice of signals to debug. The dialog box shows the clock domain that will clock the debug cores, so if a signal crosses multiple clock domains, you can manually override the chosen clock and select the specific one to capture the data with. You are also shown whether a signal is for triggering, data, or both.

Probes can be used as Data or Trigger. Probes that are needed in triggering and capturing compare values are customized as "trigger"-only probes, while probes whose data needs to be captured are customized as "data"-only probes. The above signals in the graphic are used for both data and triggering, hence Data and Trigger is selected from the Probe Type menu.

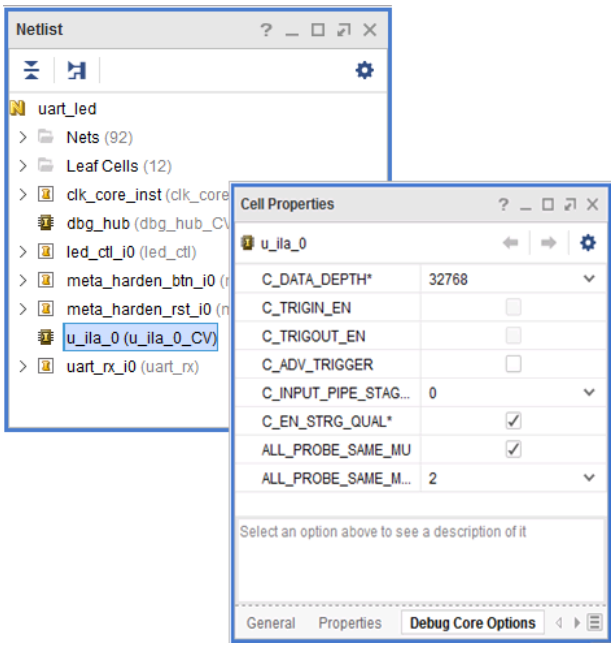
At the end of the series of dialog boxes, a summary dialog box tells you how many cores will be inserted, or removed (if appropriate), along with the number of clock domains that will be attached to the debug cores.

Slide 17-6:

Debug Core Customization

109084**slide

- The debug window in the Vivado Design Suite displays core content and configuration
  - CLK, PROBE
  - Signal count
- Set options for cores and signals in the Properties view
- View and change debug core properties
  - Select debug core **u_ila_0** > **Debug Core Options** in the instance properties window
  - C_DATA_DEPTH determines the number of samples to be captured
  - C_TRIGIN_EN enables the TRIG_IN and TRIG_IN_ACK ports
  - C_TRIGOUT_EN enables the TRIG_OUT and TRIG_OUT_ACK ports



© Copyright 2019 Xilinx

- C_ADV_TRIGGER enables advance triggering

© Copyright 2019 Xilinx

Slide 17-7:

Implementing Debug Cores in the Vivado IDE

109086**slide

- The Vivado IDE generates and synthesizes a black box debug core
- Netlists for cores are added to the project
- Cores are implemented before the tool launches
- Cores are used in all implementation runs
- Core connectivity is preserved through netlist updates

109086!*note

Debug core implementation is done automatically when implementing the design; however, you can also force debug core implementation manually for floorplanning or timing analysis.

When the debug core implementation is complete, the debug core black boxes are resolved and you can access the generated instances.

© Copyright 2019 Xilinx

Slide 17-8:

Configuring the Number of Comparators and Input Pipe Stages

155195**slide

- Number of comparators
  - ILA probe trigger comparators are used to detect specific equality or inequality conditions on the probe inputs to the ILA core
  - Trigger condition is the result of a Boolean "AND", "OR", "NAND", or "NOR" calculation of each of the ILA probe trigger comparator results
  - ILA allows you to set the exact number of comparators to be use per probe by selecting the Probe_Ports tab and setting the desired number
    - Range allowed is 1 to 16
  - It is not possible to set a different number of comparators for different probes in the ILA using the insertion flow

- Input pipe stage (C_INPUT_PIPE_STAGES)
  - Allows users to select the number of registers to be added for the probe; applies to all of the probes
  - Enables extra levels of pipe stages (for example, flip-flop registers) on the PROBE inputs
  - Improves timing performance of the design by allowing the Vivado tools to place the ILA core away from critical sections of the design
  - Valid values from 0 (default) - 6

Slide 17-9:

Apply Your Knowledge

109090**slide

23) What does the netlist insertion require?

24) Which cores can and cannot be added to a design using the inserter tools?

© Copyright 2019 Xilinx

109090!*note

25) What does the netlist insertion require?

The inserter tools require the netlist to both identify signals and to insert the cores into.

26) Which cores can and cannot be added to a design using the inserter tools?

Regardless of the flow, you can only insert the ILA cores. The VIO core cannot be added with this process.

Slide 17-10:

Summary

109091**slide

- In the netlist insertion flow, the debug cores are adding to the synthesized netlist
- The Set Up Debug Wizard is used to create and configure a debug core
- The Vivado IDE generates and synthesizes a black box debug core

109091!*note

Where Can I Learn More?

- User guide
  - *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
- Tutorial
  - *Vivado Design Suite Tutorial: Programming and Debugging* (UG936)
- Further documentation
  - www.xilinx.com/products/design-tools/vivado/debug

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Trigger Using the Trigger State Machine in the Vivado Logic Analyzer

Slide 18-1:

113822**slide

2019.1
113822!*note

This module introduces the trigger state machine (TSM), the ILA advanced trigger capability. 15 minutes.

Slide 18-2:

Objectives

137259**slide

After completing this module, you will be able to:

- Learn how to use the ILA advanced triggering option
- Describe the steps involved in using the trigger state machine (TSM)
- Code for complex trigger conditions using different states in the TSM

Slide 18-3:

ILA Advanced Triggering

113841**slide

© Copyright 2019 Xilinx

- The ILA can be configured in **Advanced mode** to use advanced capabilities, called the trigger state machine
  - When enabled, the ILA **basic mode** changes to **Advanced_only** or **Advanced_or_Trig_in** mode

- The trigger state machine enables designers to trigger the ILA with complex conditions

113841!*note

© Copyright 2019 Xilinx

The ILA core in the design that is running in the hardware device has advanced trigger capabilities. The advanced trigger mode features can be enabled by setting the Trigger mode control in the ILA Properties window of the ILA dashboard to ADVANCED_ONLY or ADVANCED_OR_TRIG_IN.

When you set the Trigger mode to ADVANCED_ONLY or ADVANCED_OR_TRIG_IN, two changes happen in the ILA dashboard.

27) A new control called Trigger State Machine appears in the ILA Properties window.

28) The Basic Trigger Setup window is replaced by a Trigger State Machine code editor window.

Slide 18-4:

What is the Trigger State Machine?

113823**slide

- A language is used to describe complex trigger conditions that map to the advanced trigger logic of the ILA debug core
  - State machine consisting of up to 16 states
  - Each state consist of one-, two-, or three-way conditional branching
  - Four built-in 16-bit counters used to count events, implement timers, etc.
  - Four built-in flags used for monitoring trigger state machine execution status
  - Trigger ILA

113823!*note

© Copyright 2019 Xilinx

The counters can be used in a trigger state machine program to keep track of multiple events.

The flags can be used in a trigger state machine program to indicate when certain branches are taken.

The state machine can execute "goto," "trigger," and various counter- and flag-related actions.

Slide 18-5:

States in the TSM

121281**slide

- Each state machine program can have up to 16 states declared
  - Each state is composed of a state declaration and a body

```
state <state_name>:
<state_body>
```

- The default trigger state machine program is valid for any ILA core configuration

Slide 18-6:

Goto Action in the TSM

113838**slide

- The `goto` action is used to transition between states
- For example, using the `goto` action to transition from one state to another before triggering

```
state my_state_0:
goto my_state_1;
state my_state_1:
trigger;
```

Slide 18-7:

© Copyright 2019 Xilinx

Conditional Branching

113837**slide

- The TSM supports one-, two-, and three-way conditional branching per state
- One-way branching involves using `goto` actions without any if/elseif/else/endif constructs

```
state my_state_0:
goto my_state_1;
```

- Two-way conditional branching uses `goto` actions with if/else/endif constructs

```
state my_state_0:
if (<condition1>) then
goto my_state_1;
else
goto my_state_0;
endif
```

113837!*note

The three-way conditional branching uses goto actions with if/else/elseif/endif constructs:

```
state my_state_0:
if (<condition1>) then
goto my_state_1;
elseif (<condition2>) then
goto my_state_2;
else
goto my_state_0;
endif
```


© Copyright 2019 Xilinx

Slide 18-8:

Counters in the TSM

113831**slide

- There are four built-in 16-bit counters which have fixed names
  - `$counter0`, `$counter1`, `$counter2`, and `$counter3`
  - Can be reset, incremented, and used in conditional statements

- To reset a counter, use the reset_counter action

```
state my_state_0:
reset_counter $counter0;
goto my_state_1;
```

- To increment a counter, use the increment_counter action

```
state my_state_0:
increment_counter $counter3;
goto my_state_1;
```

Slide 18-9:

Flags in the TSM

113825**slide

- Flags can be used to monitor the progress of the TSM program as it executes
  - Four built-in flags have fixed names, `$flag0`, `$flag1`, `$flag2`, and `$flag3`

- To set a flag, use the set_flag action

```
state my_state_0:
set_flag $flag0;
goto my_state_1;
```

- To clear a flag, use the clear_flag action

```
state my_state_0:
clear_flag $flag2;
```


© Copyright 2019 Xilinx

```
goto my_state_1;
```


© Copyright 2019 Xilinx

Slide 18-10:

Counter Conditions

113832**slide

- Counter conditions can be used in two-way and three-way branching conditional statements
  - Each condition consumes one counter comparator
- The probe port conditions consist of a comparison operator and a value
- The valid probe condition comparison operators are
  - == (equal to)
  - != (not equal to)

113832!*note

Each counter is always 16 bits wide.

Each counter has only one counter comparator. This means that you can only use a particular counter in a counter condition once in the entire TSM program.

Slide 18-11:

Conditional Statements

113834**slide

- Debug probe conditions can be used in two-way and three-way branching conditional statements
  - Consumes one trigger comparator on the PROBE port of the ILA to which the debug probe is attached
- The debug probe conditions consist of a comparison operator and a value
  - == (equal to)

© Copyright 2019 Xilinx

- != (not equal to)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

- For example
  - A single-bit debug probe called abc equals 0

```
if (abc == 1'b0) then
```


© Copyright 2019 Xilinx

Slide 18-12:

Combined Debug Probe and Counter Conditions

113836**slide

- Debug probe conditions and counter conditions can be combined together to form a single condition
  - All debug probe comparisons must be combined together using the "||" (OR) or "&&" (AND) operators
  - The combined debug probe condition can be combined with the counter condition using either the "||" (OR) or "&&" (AND) operators, regardless of the operator used to combine them

- For example
  - Two debug probe comparisons combined with an "OR" function, then combined with counter conditional using "AND" function

    ```
    if (((xyz >= 23'u456) || (abc == 1'b0)) && ($counter0 ==
    16'u0023)) then
    ```
  - Two debug probe comparisons combined with an "AND" function, then combined with counter conditional using "OR" function

    ```
    if (((xyz >= 23'u456) && (abc == 1'b0)) || ($counter0 ==
    16'u0023)) then
    ```

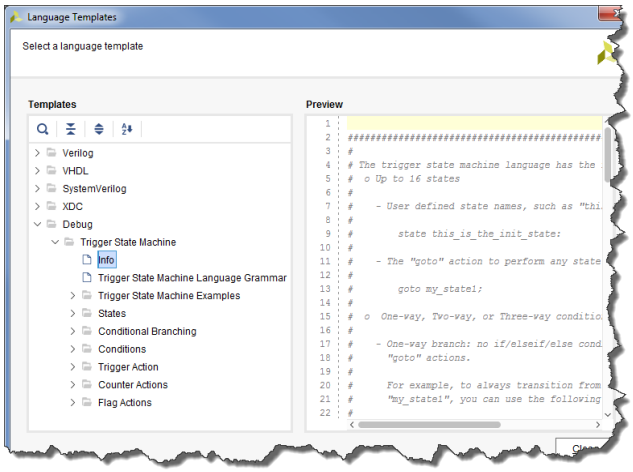
Slide 18-13:

TSM Code in Language Templates

121282**slide

© Copyright 2019 Xilinx

- Use the Language Templates window to access the template for TSM code



121282!*note

In the *Trigger using the Trigger State Machine in the Vivado Logic Analyzer* lab of this topic cluster, you will use a trigger state machine to trigger the ILA core.

© Copyright 2019 Xilinx

Slide 18-14:

TSM Code Example

129160**slide

```
#
# Trigger on the 4th rising edge of debug probe called "abc"
#
state wait_for_4th_rising_edge_of_abc:
  if ((abc == 1'bR) && ($counter0 == 16'h0003)) then
    reset_counter $counter0;
    trigger;
  elseif (abc == 1'bR) then
    increment_counter $counter0;
    goto wait_for_4th_rising_edge_of_abc;
  else
    goto wait_for_4th_rising_edge_of_abc;
  endif
```

129160!*note

In this example, the state is triggered when the fourth rising edge comes. Depending upon the condition, the triggering action happens and the counter is incremented accordingly.

Slide 18-15:

Summary

137258**slide

© Copyright 2019 Xilinx

- The trigger state machine enables designers to trigger the ILA with complex conditions
- Each state machine program can have up to 16 states declared
- Four built-in 16-bit counters are present
- Four flags can be used to monitor the progress of the TSM program as it executes

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Sampling and Capturing Data in Multiple Clock Domains

Slide 19-1:

```
109205**slide

2019.1
109205!*note
```

This module highlights sampling in multiple clock domains, an important technique that is frequently used with the Vivado® debug tool. 5 minutes.

Slide 19-2:

Objectives

```
109206**slide
```

After completing this module, you will be able to:

- Examine sampling data in multiple clock domains

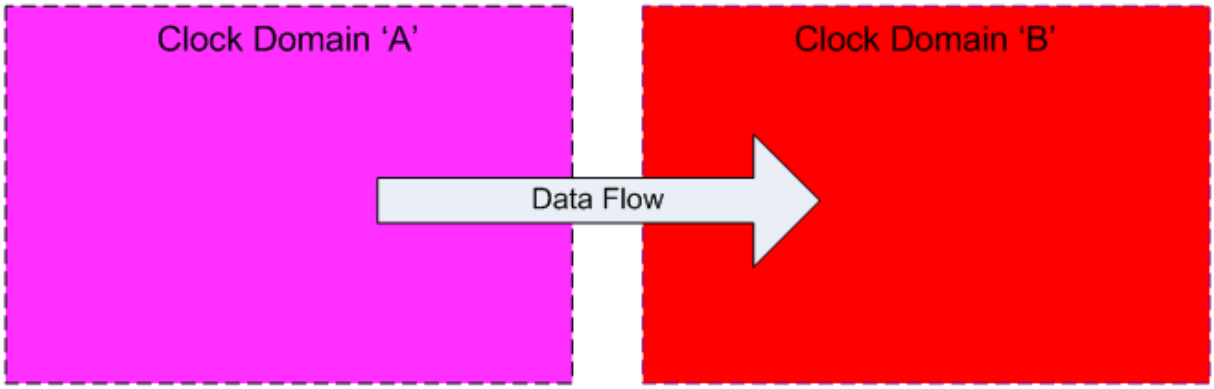
© Copyright 2019 Xilinx

Slide 19-3:

Multiple Clock Domains

109207**slide

- Many FPGA designs contain multiple clock domains
  - Signals will be in different clock domains
- Oversampling technique can be used to sample the signals
- Oversampling technique can increase the risk of metastable conditions
  - Possible reduction of performance due to driving the Vivado debug tools too fast
  - Cost of logic and block RAM space



109207!*note

Oversampling is a technique by which each signal is sampled at a greater frequency than the sample changes. This is occasionally used to measure specific, asynchronous timing and is not recommended for general use.

The easiest technique to use is to employ an asynchronous FIFO.

© Copyright 2019 Xilinx

Slide 19-4:

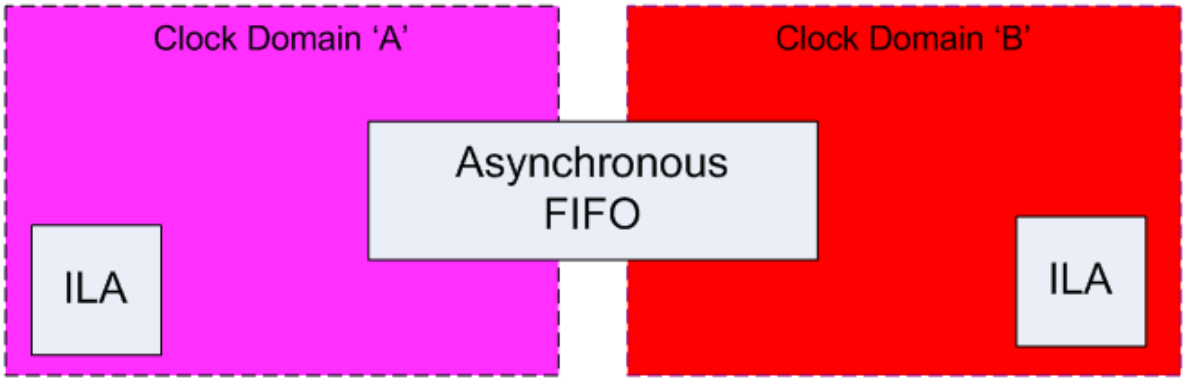
Debugging in Multiple Clock Domains

109208**slide

- Good design practices call for clock domain-crossing circuits
  - Like asynchronous FIFO

- Recall the following facts about ILAs
  - Can be independently clocked
  - Can be externally triggered
  - Can share the same signal
    - Which is sampled according to the ILA clock

- These factors enable sampling in separate domains



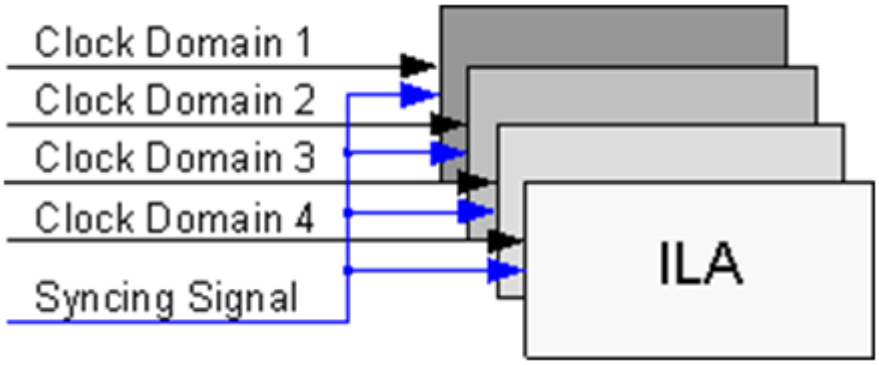
© Copyright 2019 Xilinx

Slide 19-5:

Multiple ILAs

109209**slide

- Each ILA must be independently armed by using the Vivado logic analyzer
- In order to avoid any of the ILA triggering as soon as the trigger condition is met, a synchronizing signal must be used to manually hold off the trigger until all ILAs are armed
- The syncing signal is subject to metastability
- With multiple ILAs, a hold-off signal should be used
  - To allow the arming of all cores prior to the first trigger occurrence



109209!*note

The designer can specifically protect the ILA against metastability, but there may be some uncertainty as to exactly how the data aligns with data from other ILAs because of latency through the metastable protection circuit and the possibility of the metastability situation itself.

Timing considerations for ILA:

- Ensure that the clocks chosen for the ILA cores are free-running clocks. Failure to do so could result in an inability to communicate with the debug core when the design is loaded onto the device.

© Copyright 2019 Xilinx

- Make sure the clock input to the ILA core is synchronous to the signals being probed. Failure to do so results in timing issues and communication failures with the debug core when the design is programmed into the device.
- Close timing on the design prior to adding the debug cores. Xilinx does not recommend using the debug cores to debug timing-related issues.
- If you still notice that timing has degraded due to adding the ILA debug core and the critical path is in dbg_hub, perform the following steps:
  - Open the synthesized design.
  - Find the dbg_hub cell in the netlist.
  - Go to the properties of dbg_hub.
  - Find the property C_CLK_INPUT_FREQ_HZ.
  - Set it to the frequency (in Hz) of the clock that is connected to dbg_hub.
  - Find the property C_ENABLE_CLK_DIVIDER and enable it.
  - Re-implement design.
- For designs with high-speed clocks, consider the following:
  - Limit the number and width of the signals being debugged.
  - Pipeline the input probes to the ILA (C_INPUT_PIPE_STAGES), which enables extra levels of pipe stages.
- For designs with limited MMCM/BUFG availability, consider clocking the debug hub with the lowest clock frequency in the design instead of using the clock divider inside the debug hub.

In the lab you will be introduced to debugging techniques such as configuring the logic analyzer to support cross-clock domains and capture capabilities.

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

UltraScale Architecture DSP Resources

Slide 20-1:

```
118422**slide

2019.1
118422!*note
```

This module describes DSP slice resources in the UltraScale™ architecture. 20 minutes.

Slide 20-2:

Objectives

```
118424**slide
```

After completing this module, you will be able to:

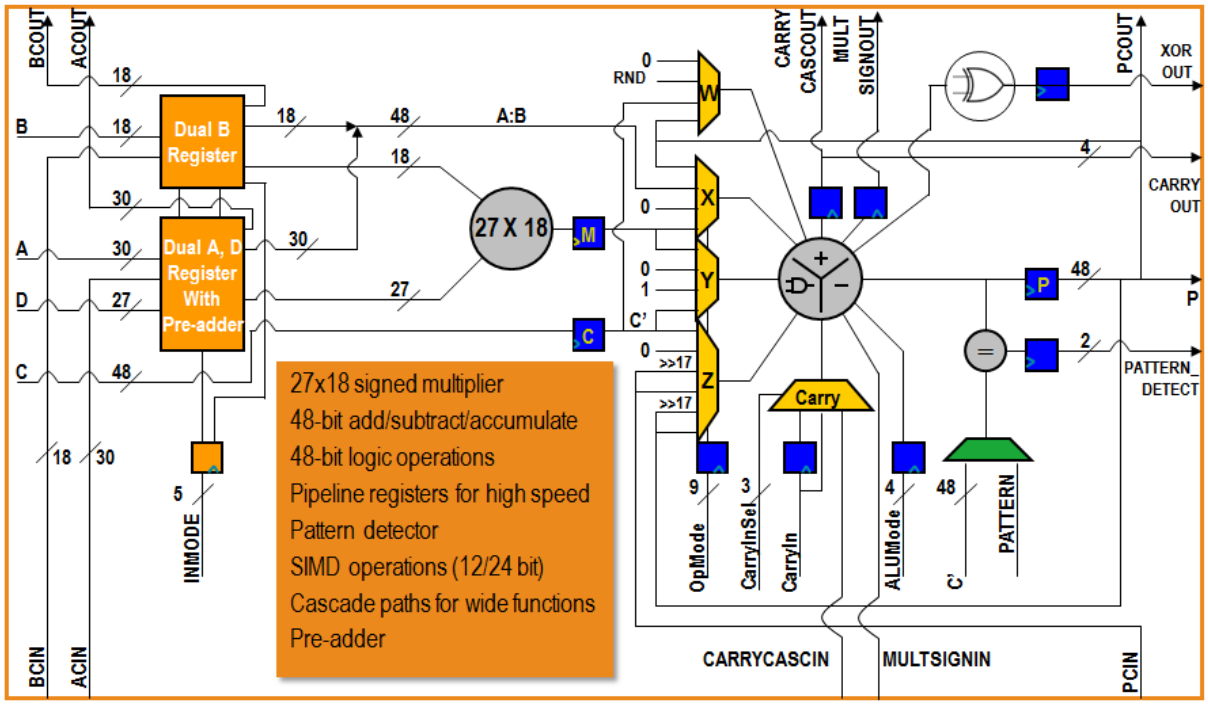
- Describe the DSP slice architecture in the UltraScale architecture
- Describe the capabilities of DSP slice resources
- Instantiate and infer DSP resources

© Copyright 2019 Xilinx

Slide 20-3:

UltraScale Architecture DSP48E2 Slice

118427**slide



118427!*note

© Copyright 2019 Xilinx

All UltraScale™ FPGAs contain the same DSP48E2 slice. Only the number of slices and maximum frequency vary from device to device.

C' denotes the output of the optional C pipeline register. This bus is one input of the pattern detect multiplexer; the other input is the 48-bit PATTERN attribute.

The DSP48E2 has additional features compared to the DSP48E1 of previous generations:

- Wider multiplier (27x18 versus 25x18)
- Wider pre-adder, D, and AD registers (27 bits versus 25 bits)
- More flexibility in the pre-adder
- Ability to use the multiplier to square some inputs as well as the pre-adder result
- Additional operand to the ALU controlled by the W multiplexer, including a 48-bit RND constant
- Wide XOR function

© Copyright 2019 Xilinx

Slide 20-4:

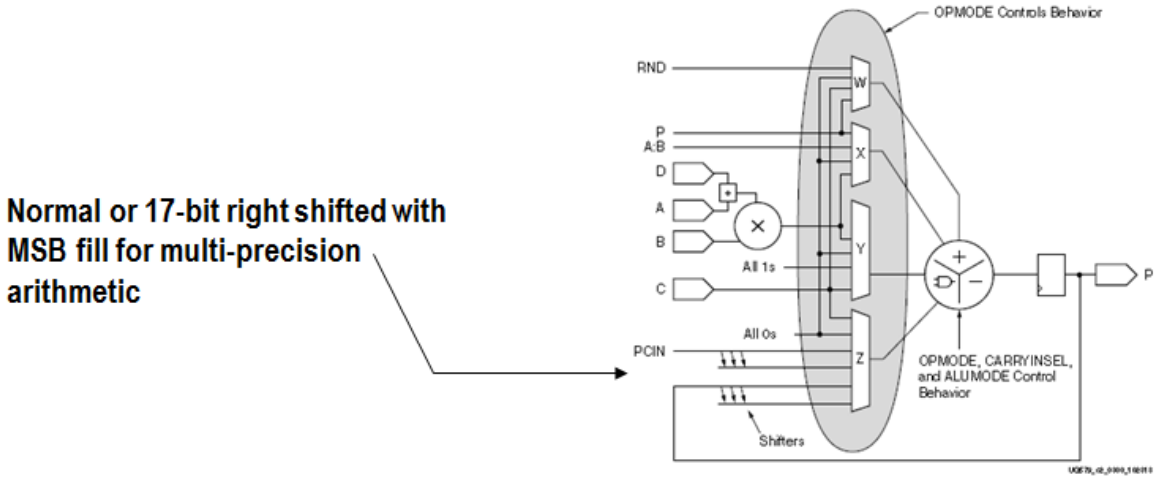
X, Y, Z, and W Multiplexers

118429**slide

- The adder/subtractor operates on X, Y, Z, W, and CIN operands
  - The table shows basic operations

ALUMODE	Operation
0000	Z + W + X + Y + CIN
0001	-Z + (W + X + Y + CIN) – 1
0010	-Z – W – X – Y – CIN – 1
0011	Z – (W + X + Y + CIN)
Others	Logic Operations

- X, Y, Z, and W multiplexers allow for dynamic OPMODEs
- Multiplier output requires both X and Y multiplexers



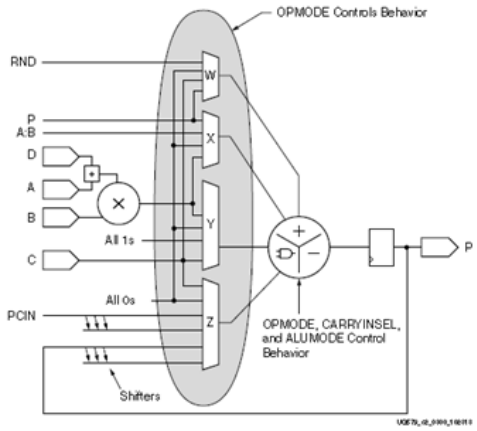
© Copyright 2019 Xilinx

Slide 20-5:

Apply Your Knowledge

118430**slide

- What is the OPMODE for the following functions?
  - C + A:B
  - A*B + C
  - P + C + PCIN
  - A*B + C + P



W		Z			Y		X		Select	Notes
8	7	6	5	4	3	2	1	0		
0	0	x	x	x	x	x	x	x	*Zero*	*Default*
0	1	x	x	x	x	x	x	x	P	Requires PREG=1
1	0	x	x	x	x	x	x	x	RND	
1	1	x	x	x	x	x	x	x	C	

W		Z			Y		X		Select	Notes
8	7	6	5	4	3	2	1	0		
x	x	x	x	x	x	x	0	0	*Zero*	*Default*
x	x	x	x	x	0	1	0	1	Mulitplier Output (A xB)	Must select with Y[3:2]=01
x	x	x	x	x	x	x	1	0	P	
x	x	x	x	x	x	x	1	1	A:B	48 bits wide

W		Z			Y		X		Select	Notes
8	7	6	5	4	3	2	1	0		
0	0	x	x	x	0	0	x	x	*Zero*	*Default*
0	1	x	x	x	0	1	0	1	Mulitplier Output (A x B)	Must select with X[1:0]=01
1	0	x	x	x	1	0	x	x	48'hFFFF_FFFF_FFFF	For Bitwise Logic Ops
1	1	x	x	x	1	1	x	x	C	

W		Z			Y		X		Select	Notes
8	7	6	5	4	3	2	1	0		
x	x	0	0	0	x	x	x	x	*Zero*	*Default*
x	x	0	0	1	x	x	x	x	PCIN	
x	x	0	1	0	x	x	x	x	P	
x	x	0	1	1	x	x	x	x	C	
x	x	1	0	0	x	x	x	x	P	For MACC extend
x	x	1	0	1	x	x	x	x	Shift(PCIN)	Shifted by 17 bits
x	x	1	1	0	x	x	x	x	Shift(P)	Shifted by 17 bits
x	x	1	1	1	x	x	x	x		illegal selection

118430!*note

The OPMODE of each DSP component is individually controllable.

The OPMODE can change dynamically on each cycle.

- C + A:B
  - OPMODE = 11 000 00 11 or 00 011 00 11 or 00 000 11 11
- A*B + C
  - OPMODE = 00 011 01 01 or 11 000 01 01
- P + C + PCIN
  - OPMODE = 00 001 11 10 or 11 001 00 10 or 01 001 11 00
- A*B + C + P
  - OPMODE = 01 011 01 01 or 11 010 01 01

© Copyright 2019 Xilinx

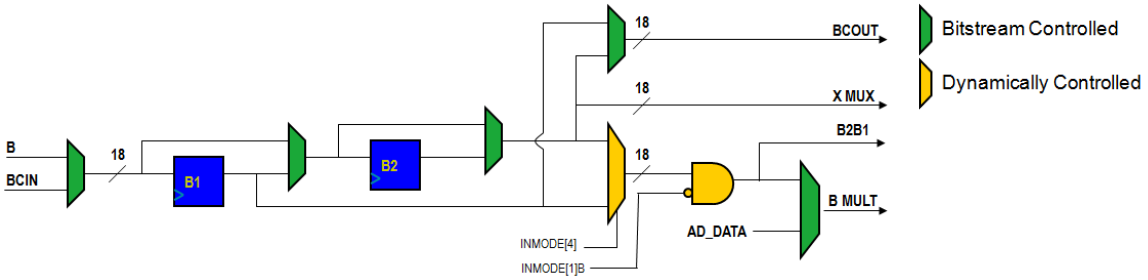
© Copyright 2019 Xilinx

Slide 20-6:

Dual B Register

118431**slide

- B input to multiplier is controlled by INMODE[4]
  - Dynamically selects B1/B2 pipeline level
- B MULT can be driven from AD_DATA via bitstream option
  - AD_DATA comes from Dual A, D Register
    - Allows squaring of the AD_DATA
- X MUX and BCOUT cascade outputs are statically controlled by bitstream options



118431!*note

New multiplexer on B MULT output the squaring of the AD_DATA.

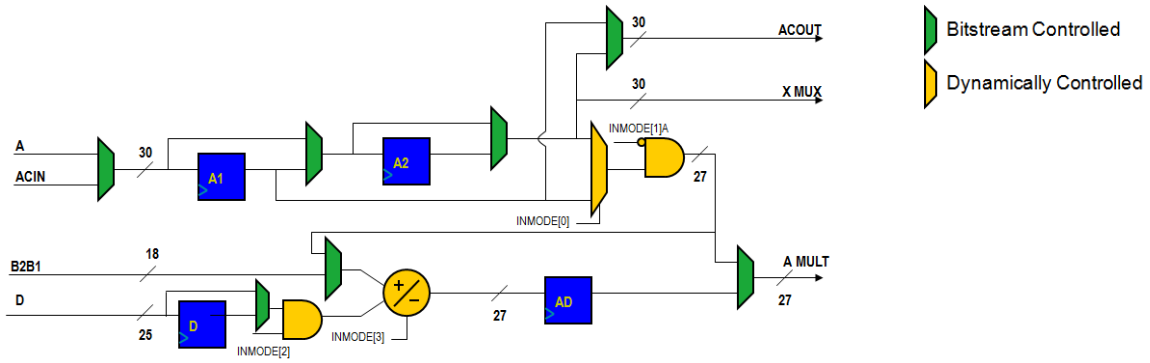
© Copyright 2019 Xilinx

Slide 20-7:

Dual A, D, Registers and Pre-Adder

118432**slide

- A input to multiplier is controlled by INMODE[3:0]
  - Dynamically selects A1/A2 pipeline level
  - Dynamically selects add/subtract
  - Dynamically selects Zero for A or D
- A MULT output can select B2B1 from B registers
  - Allows squaring of B or (D +/- B)
- ACOUT and X MUX input are statically controlled



118432!*note

Fine-grain access to the A and B pipelines optimizes the implementation of certain algorithms, like short FFTs and sequential complex multiplications.

The pre-adder doubles the efficiency of symmetrical filters and convolutions.

New multiplexer on A MULT allows the calculation of B^2, (D +/- B)*B, (D +/- A)^2, …

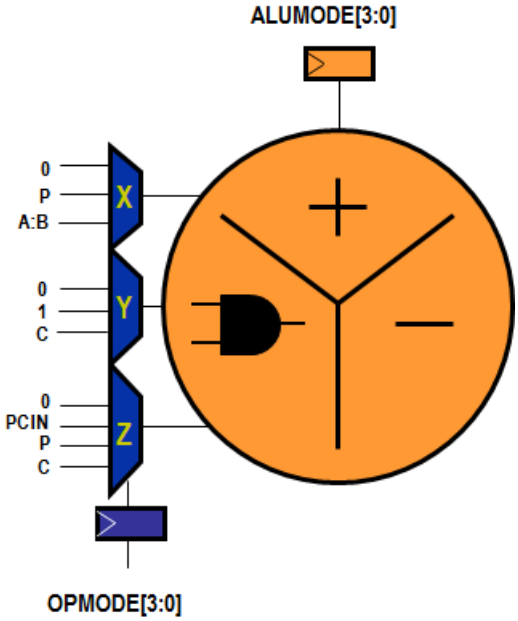
© Copyright 2019 Xilinx

Slide 20-8:

Two-Input Logic Functions

118433**slide

- 48-bit logic operations
  - XOR, XNOR, AND, NAND, OR, NOR, NOT



ALUMODEs

Logic Unit Mode	OPMODE[3:2]	ALUMODE[3:0]
X XOR Z	00	0100
X XNOR Z	00	0101
X XNOR Z	00	0110
X XOR Z	00	0111
X AND Z	00	1100
X AND (NOT Z)	00	1101
X NAND Z	00	1110
(NOT X) OR Z	00	1111
X XNOR Z	10	0100
X XOR Z	10	0101
X XOR Z	10	0110
X XNOR Z	10	0111
X OR Z	10	1100
X OR (NOT Z)	10	1101
X NOR Z	10	1110
(NOT X) AND Z	10	1111
	11	0100

118433!*note

The DSP48E2 has an additional operation over previous generations: X XOR Y XOR Z.

For this operation, the Y multiplexer must select the C input.

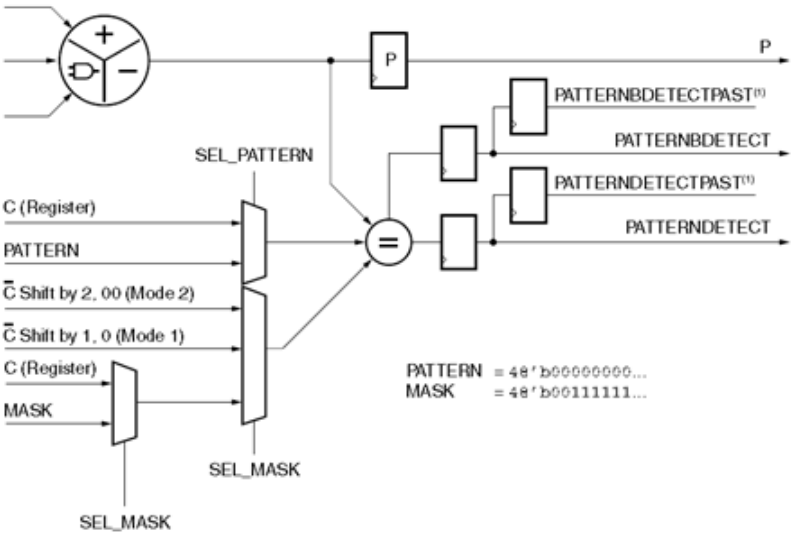
© Copyright 2019 Xilinx

Slide 20-9:

Pattern Detect

118434**slide

- Pattern and mask operation on output of adder
  - Pattern can be constant (set by attribute) or C input
  - Pattern can be constant (set by attribute) or C input
  - PATTERNB detects complement of pattern



- Enables
  - Symmetric rounding for multi-precision operations
  - Convergent rounding
  - Saturation
  - Accumulator terminal count

© Copyright 2019 Xilinx

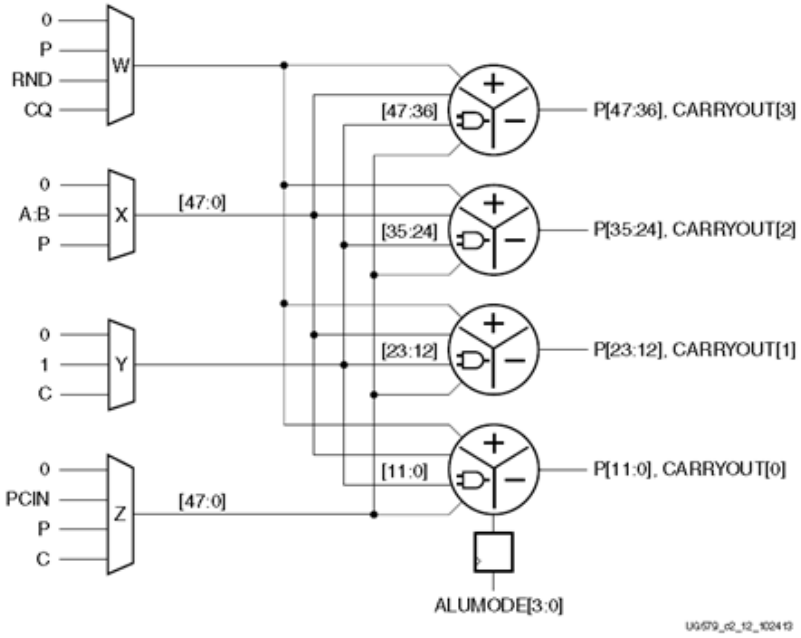
על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 20-10:

SIMD

118435**slide

- 48-bit adder broken into 2x24 bits or 4x12 bits
  - Allows two or four independent additions to be done
- Carry bits brought out independently and disabled between sections
  - Carry bits can be cascaded between DSP48E2 _slices



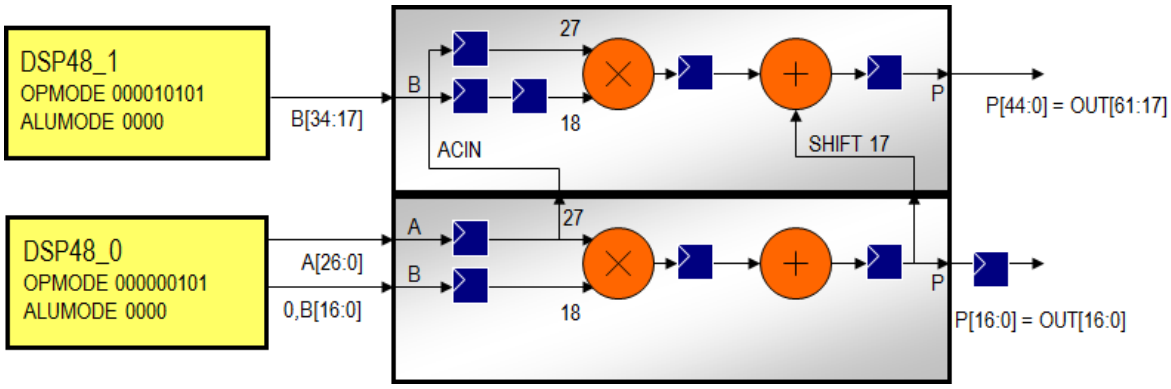
© Copyright 2019 Xilinx

Slide 20-11:

Cascade Paths

118436**slide

- Cascade paths exist from each DSP48E2 slice to the slice above it
  - A input, B input, P output, and carry out
    - P cascade path can be shifted by 17 bits by slice above
- Enables common functions with little or no additional resources
  - Wider accumulators, wider multipliers, complex multipliers, and FIR filters
    - Example: 35-bit x 27-bit multiplier with two DSP48E1s



118436!*note

The cascade connections do not cross onto the silicon interposer for devices that use stacked silicon interconnect technology. When using the cascade paths, ensure that DSP48E2 cells do not straddle an SLR boundary.

© Copyright 2019 Xilinx

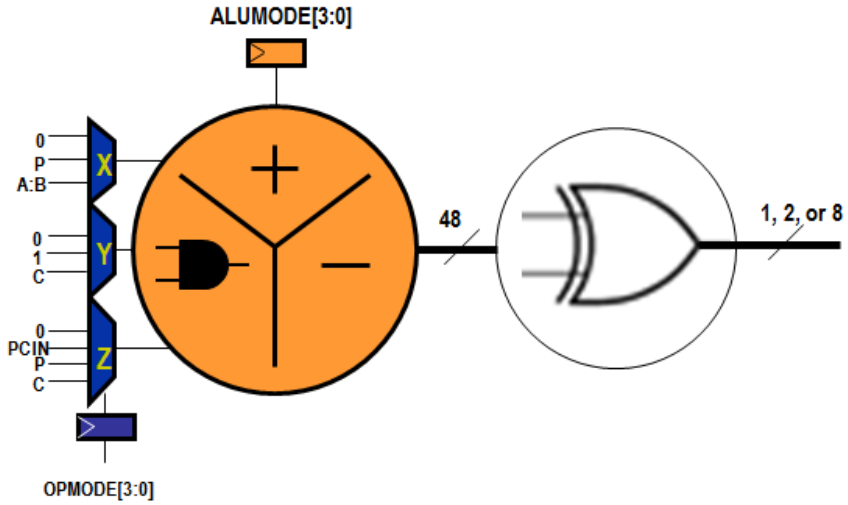
Slide 20-12:

Wide XOR

118437**slide

- ALU can do 48-bit 2-input or 3-input XORs
  - (X XOR Z)
  - (X XOR Y XOR Z)

- Wide XOR does reduction XOR of ALU result in four possible configurations
  - 1 x 48 to 1
  - 2 x 24 to 2
  - 4 x 12 to 4
  - 8 x 6 to 8



118437!*note

The wide XOR is a new feature of the DSP48E2 over previous generations.

Slide 20-13:

IP Support and Inference

118438**slide

- Some basic functions can be inferred
  - These include multiplier, multiply-accumulate, and others

© Copyright 2019 Xilinx

- Other functions are supported through the Vivado® IP catalog
  - Math and DSP functions

- Device primitives can be directly instantiated
  - DSP48E2 can be instantiated manually
    - Primitives have lots of ports and attributes

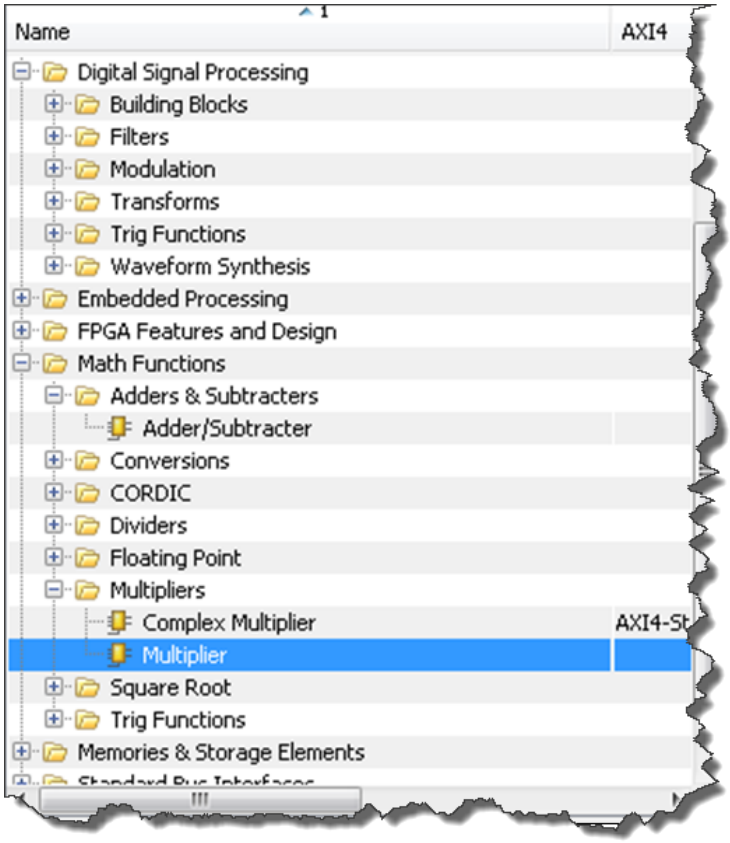
© Copyright 2019 Xilinx

Slide 20-14:

Math and DSP Operations in the IP Catalog

118442**slide

- Math and DSP functions are available from the IP catalog
  - Some can target either DSP48 cells or LUTs
    - Can build operations that target multiple DSP48 cells
  - Many can control the pipelining for performance tradeoffs
- Language templates also provide instantiation templates
  - Over 100 ports and attributes



© Copyright 2019 Xilinx

Slide 20-15:

Inferring a 16 x 16 Multiplier

118439**slide

```
-----------------------------------------------------------------
-- Example: 16x16 Multiplier, inputs registered once,
-- outputs twice
--        Matches 1 DSP48 slice
--        OpMode(W,Z,Y,X):Subtract
--          (00,000,01,01):0
-----------------------------------------------------------------
p1 <= a1*b1;

 process (clk) is
 begin
  if clk'event and clk = '1' then
   if rst = '1' then
    a1 <= (others => '0');
    b1 <= (others => '0');
    p  <= (others => '0');
   elsif ce = '1' then
    a1 <= a;
    b1 <= b;
    p  <= p1;
   end if;
  end if;
 end process;
```

```
//////////////////////////////////////////////////////////////
// Example: 16x16 Multiplier, inputs registered
// once, outputs twice
//        Matches 1 DSP48 slice
//        OpMode(W,Z,Y,X):Subtract
//          (00,000,01,01):0
//////////////////////////////////////////////////////////////

 assign p1 = a1*b1;

 always @(posedge clk)
  if (rst == 1'b1)
    begin
     a1 <= 0;
     b1 <= 0;
     p <= 0;
    end
  else if (ce == 1'b1)
    begin
     a1 <= a;
     b1 <= b;
     p  <= p1;
    end
```

Slide 20-16:

© Copyright 2019 Xilinx

Inferring a 2-Input Adder

118440**slide

```
-----------------------------------------------------------------
-- Example: 16 bit adder 2 inputs, input and output
-- registered once
--   Mapping to DSP48 should be driven  by timing as
-- DSP48 are limited resources.  The -use_dsp
-- switch must be set to YES
--   Matches 1 DSP48 slice
--         OpMode(W,Z,Y,X):Subtract
--           (00,000,11,11):0  or
--           (00,011,00,11):0
-----------------------------------------------------------------

p1 <= a1 + b1;

 process (clk) is
 begin
   if clk'event  and clk = '1' then
     if rst = '1' then
       p  <= (others => '0');
       a1 <= (others => '0');
       b1 <= (others => '0');
     elsif ce = '1' then
       a1 <= a;
       b1 <= b;
       p <= p1;
     end if;
   end if;
 end process;
```

```
//////////////////////////////////////////////////////////////////
// Example: 16 bit adder 2 inputs, input and output
// registered once
//   Mapping to DSP48 should be driven  by timing as
//   DSP48 are limited resources.  The -use_dsp
//   switch must be set to YES
//   Matches 1 DSP48 slice
//         OpMode(W,Z,Y,X):Subtract
//           (00,000,11,11):0  or
//           (00,011,00,11):0
//////////////////////////////////////////////////////////////////

assign p1 = a1 + b1;

 always @(posedge clk)
   if (rst == 1'b1)
      begin
        p  <= 0;
        a1 <= 0;
        b1 <= 0;
      end
   else if (ce == 1'b1)
      begin
        a1 <= a;
        b1 <= b;
        p <= p1;
   end
```


© Copyright 2019 Xilinx

Slide 20-17:

Inferring a Loadable MACC

118441**slide

```
-----------------------------------------------------------------------
-- Example: Loadable Multiply Accumulate with one level
--  of registers
--        Map into 1 DSP48 slice
--        Funtion: OpMode(W,Z,Y,X):Subtract
--        - load    (00,011,00,00):0
--        - mult_acc (00.010,01,01):0
-- Expected mapping:
--        AREG: no, BREG: no, CREG: no, MREG: no, PREG: yes
-----------------------------------------------------------------------

 with load select p_tmp <= signed(c) when '1',
  p_reg + a1*b1 when others;

 process(clk)
 begin
  if clk'event and clk = '1' then
    if p_rst = '1' then
      p_reg <= (others => '0');
      a1 <= (others => '0');
      b1 <= (others => '0');
    elsif p_ce = '1' then
      p_reg <= p_tmp;
      a1 <= signed(a);
      b1 <= signed(b);
    end if;
  end if;
 end process;

 p <= std_logic_vector(p_reg);
```

```
//////////////////////////////////////////////////////////////////////
// Example: Loadable  Multiply Accumulate with one level
//  of registers
//        Map into 1 DSP48 slice
//        Funtion: OpMode(W,Z,Y,X):Subtract
//        - load     (00,011,00,00):0
//        - mult_acc (00.010,01,01):0
// Expected mapping:
//    AREG: no, BREG: no, CREG: no, MREG: no, PREG: yes
//////////////////////////////////////////////////////////////////////

 assign p_tmp = load ? c:p + a1*b1;

 always @(posedge  clk)
  if (p_rst  == 1'b1)  begin
    p <= 0;
    a1 <=0;
    b1 <=0;
  end
  else if (p_ce  == 1'b1)  begin
    p <= p_tmp;
    a1 <=a;
    b1 <= b;
  end
```

Slide 20-18:

Apply Your Knowledge

118447**slide

29) Can you identify the features of the DSP slice in the UltraScale architecture?

118447!*note

30) Can you identify the features of the DSP slice in the UltraScale architecture?

- 27x18 twos complement multiplier
- 27 bit pre-adder

© Copyright 2019 Xilinx

- Dynamic input pipeline control
- 48-bit add/subtract/accumulator
- 48-bit logic unit
- Wide XOR
- SIMD operations (12/24 bit)
- Cascade paths for wide functions
- Dynamic pipeline selection
- Pipeline registers for high speed
- Pattern detector

© Copyright 2019 Xilinx

Slide 20-19:

Summary

118444**slide

- DSP functions allow extremely high-speed mathematical operations
  - Fast multiplication and multiply-accumulate operations
  - Excellent resources for high-performance digital signal processing (DSP) applications
- DSP resources can be inferred or instantiated
- DSP functions are also available from the IP catalog

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Non-Integer Numbers

Slide 21-1:

```
6277**slide
2015.3
*note
```

This module introduces the basic concepts of fixed and floating point numeric representations for synthesis. 45 minutes.

Slide 21-2:

Objectives

```
6278**slide
```

After completing this module, you will be able to:

- Explain the contents and relevance of the IEEE-proposed standard for synthesis of floating point numbers
- List alternatives to floating point numbers
- Describe the theory behind floating point representation

Slide 21-3:

© Copyright 2015 Xilinx

Overview

6279**slide

- **Overview**
- Fixed Point Numbers
- Floating Point Numbers
- Summary

© Copyright 2015 Xilinx

Slide 21-4:

Non-Integer Numbers

6280**slide

- When synthesized all data types turn into 1s and 0s
  - Not all data types are synthesizable
  - They are numbers because of how these 1s and 0s are interpreted

- Real numbers are not floating point numbers
  - Real is a defined type and does not synthesize
  - Real has a defined minimum range –1.0E+38 to +1.0E+38
    - This is the minimum range specified by the IEEE; any implementation can exceed this
  - Floating point numbers are synthesizable
    - As are fixed point numbers

Slide 21-5:

Real, Fixed Point, and Floating Point Numbers

6281**slide

- Real numbers are represented internal to the simulation tool
  - User has no access to the "raw" format
  - Tool manufacturer may extend the range of the "real"

- Fixed point numbers are effectively scaled integers
  - An imaginary radix point is specified to delineate the integer portion from the fractional portion

- Floating point numbers are specially crafted fields
  - One sign bit
  - Some number of bits to represent the mantissa or significant
  - Some number of bits to represent the exponent

© Copyright 2015 Xilinx

© Copyright 2015 Xilinx

Slide 21-6:

Fixed Point Numbers

6282**slide

- Overview
- **Fixed Point Numbers**
- Floating Point Numbers
- Summary

Slide 21-7:

Fixed Point Number

6283**slide

- A fixed point number is one in which the radix point never moves within a fixed field
- Fixed point numbers are nearly as fast as integers and can perform basic comparison, addition, and subtraction functions
  - As long as the two numbers have their radix points aligned
- A useful technique when limited precision and range is required

6283!*note

Radix point: Separates the integer from the fractional portion. When base 10 is used, this is most commonly referred to as the “decimal” point.

© Copyright 2015 Xilinx

Slide 21-8:

Representing Fixed Point Numbers

6284**slide

- Integers
  - Pros – Math is natural to apply to integers
  - Cons – Requires the user to manually track scaling factors
    - In 5.4 format, 12.125 is 192

- std_logic_vector
  - Pros – Makes shifting for alignment or multiplication/division by powers of two easy
  - Cons – Still requires the user to manually track scaling factors

- Records (integer portion, fractional portion)
  - Pros – Easy management of integer and fractional portions
  - Cons
    - Carries from fractional portion to integer portion must be managed manually
    - Still requires shifting for alignment with fractional portions of different sizes

6284!*note

Recommendation: Use integers or std_logic_vectors.

© Copyright 2015 Xilinx

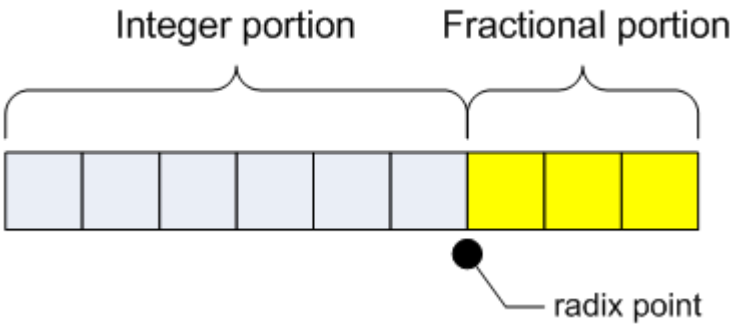
Slide 21-9:

Basics of Fixed Point Math

6285**slide

- Numbers are represented by a certain quantity of bits

- Some of the bits are used to represent the integer portion of the number
  - The remainder is used to represent the fractional portion of the number



- From the illustration, the fixed point number has the following qualities
  - Assuming unsigned
    - Range: 2^6-1 + (1-2^{-3})= 0 to 63.875
    - Resolution = .125 (1/8)

6285!*note

A radix point is the formal term used to indicate the differentiation point between the integer and fractional portion of a number.

In the decimal system, the radix point carries a specific name. The decimal point which indicates the demarcation between the integer and the tenth's place. (So in base 8, the radix point would be called the octal point; in base 2, it is the binary point.)

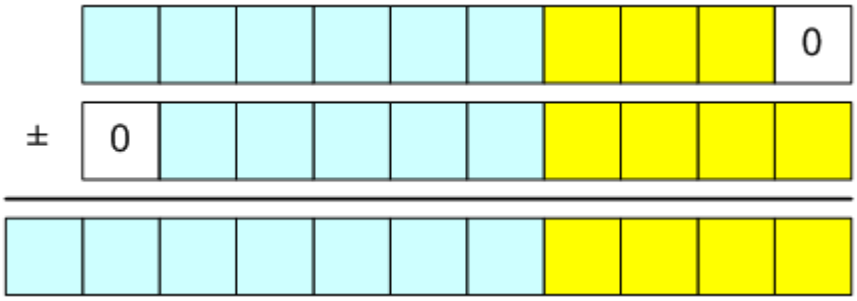
© Copyright 2015 Xilinx

Slide 21-10:

Addition and Subtraction of Fixed Point Numbers

6286**slide

- Step 1: Align the radix points by left shifting and filling with 0s as necessary
  - This allows fixed point numbers of different precisions and ranges to be added and subtracted
- Step 2: Perform the operation
  - Remember to provide an extra bit in the case of addition as it can generate a carry
- Example
  - Add (or subtract) two fixed point numbers: one in 6.3, the other in 5.4 format



6286!*note

In the example, two numbers (one in a (6.3) format and the other in (5.4) format) are to be added.

Adjust the 6.3 number by shifting one bit to the left and zero filling.

Depending on the algorithm used to add, the 5.4 number may need to be resized to 6.4 to match the other argument.

Now that the numbers are aligned, the addition or subtraction can be performed.

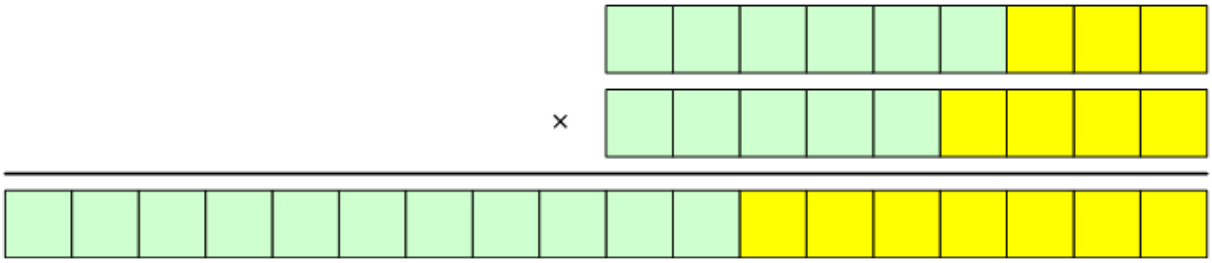
© Copyright 2015 Xilinx

Slide 21-11:

Multiplication of Fixed Point Numbers

6287**slide

- Multiplication results in the maximum size of the sum of the length of both multiplicands
  - Radix must be moved to the sum of the fractional portions
- Multiplication can be implemented as
  - Repeated summations
  - Radix adjusted integer multiplications
- Example: Multiply a 6.3 format and 5.4 format number



6287!*note

Remember that the radix point will have to be moved after the multiplication is performed.

The result can be reformatted to reside in a smaller format by losing precision in the fractional portion.

© Copyright 2015 Xilinx

Slide 21-12:

IEEE Fractional Math Libraries

6288**slide

- Officially designated IEEE 1076.3-2006
  - Designed for use with IEEE 1076-2006
    - However a compatibility version supports IEEE 1076-1993
  - Found under IEEE_PROPOSED library

- Includes
  - Fixed point package
  - Floating point package
  - Math utility package

- Provided with the Vivado® simulator for all parts

- Vivado synthesis engine supports IEEE_PROPOSED for all 7 series FPGAs
  - Older families: User must manually point to

  <xilinx install directory>\VHDL\src\IEEE_PROPOSED

© Copyright 2015 Xilinx

Slide 21-13:

IEEE Solution

6289**slide

- IEEE provides a fixed point math package
  - In library IEEE_PROPOSED
    - Has been adopted for IEEE 1076.3-2008
    - Use the proposed library as implemented by the tool vendor
  - Included in the <install directory>\vhdl\src\ieee or ieee_2008

- Special data types are defined
  - ufixed for unsigned fixed point notation
  - sfixed for signed fixed point notation

- Fixed point numbers
  - Defined as type of sfixed (an array)
  - Positive index down to 0 represents the integer portion of the fixed point number
  - –1 down to the right index represents the fractional portion
    - Subtype sfixed7 is sfixed (3 downto -3); -- 7 bit (4 bits int, 3 bits frac)
    - Subtype sfixed16 is sfixed (7 downto -8); -- 16 bit (8 bits int, 8 bits frac)

Slide 21-14:

Important Considerations for Fixed Point Use

6290**slide

© Copyright 2015 Xilinx

- If the size of the result signal or variable is not proper, then the solution will be misrepresented
- Example: If two [7,-3] numbers are added together, there is a possibility of an overflow; so the result must be an [8,-3] number—even if it is known that an overflow cannot occur
- Example: If two [7, -3] numbers are multiplied together, there is a strong possibility that the result will be much larger; therefore, the solution must be stored in a [14, -6] number that can be resized later if necessary
- Tip: Follow the sizing guide on the following slide to select the proper size for the results of the various operations

6290!*note

[7:3] represents seven bits for the integer portion and three bits for the fractional portion, or ufixed(6 downto -3).

Slide 21-15:

© Copyright 2015 Xilinx

Unsigned Fixed Point Sizing Guide

6291**slide

Operation	Result Range
A + B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A - B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A * B	A'left + B'left+1 downto A'right + B'right
A rem B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed /	A'left - B'right+1 downto A'right - B'left
Signed A mod B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed Reciprocal(A)	-A'right downto -A'left-1
Abs(A)	A'left +1 downto A'right
- A	A'left +1 downto A'right
Unsigned /	A'left - B'right downto A'right - B'left -1
Unsigned A mod B	B'left downto Min(A'right, B'right)
Unsigned Reciprocal(A)	-A'right +1 downto -A'left

6291!*note

Table taken from *Fixed Point Package User's Guide* by David Bishop.

This slide is intended as reference material.

© Copyright 2015 Xilinx

Slide 21-16:

Fixed Point Example

6292**slide

```
signal fxp_A  : sfixed (10 downto -6);
. . .
signal fxp_AB : sfixed (11 downto -6);
signal fxp_AC : sfixed (fxp_A'left+fxp_B'left+1 downto
                        fxp_A'right+fxp_B'right);
```

```
-- convert from integer directly to signed, fixed point number
fxp_A <= to_sfixed(A, 10, -6) when rising_edge(clk);
fxp_B <= to_sfixed(B, 10, -6) when rising_edge(clk);
fxp_C <= to_sfixed(C, 10, -6) when rising_edge(clk);

-- do an add and multiply
fxp_AB <= fxp_A + fxp_B when rising_edge(clk);
fxp_AC <= fxp_A * fxp_C when rising_edge(clk);

-- drive the outputs
fxp_AplusB <= to_slv(fxp_AB) when rising_edge(clk);
fxp_AdivC  <= to_slv(fxp_AC) when rising_edge(clk);
```

Slide 21-17:

Floating Point Numbers

6293**slide

- Overview
- Fixed Point Numbers
- **Floating Point Numbers**
- Summary

© Copyright 2015 Xilinx

Slide 21-18:

Floating Point Formats

6294**slide

- Numerous mechanisms exists for representing floating point numeric formats
- Most widely accepted is the IEEE 754 standard. Standard includes
  - Arithmetic formats and interchange formats
  - Rounding algorithms
  - Operations and exception handling
- IEEE 854 defines a variable width floating point specification
  - IEEE 754 covers 32-bit and 64-bit implementations
- Not all aspects of IEEE 754 are required for a successful FPGA synthesizable implementation
- Note: IEEE 754 is a specification for floating point numeric formats and was originally intended for use in computers
  - There is no FPGA-specific intention

6294!*note

Arithmetic formats: Sets of binary and decimal floating-point data, which consist of finite numbers, (including negative zero and subnormal numbers), infinities, and special 'not a number' values (NaNs).

Interchange formats: Encodings (bit strings) that can be used to exchange floating point data in an efficient and compact form.

Rounding algorithms: Methods to be used for rounding numbers during arithmetic and conversions.

Operations: Arithmetic and other operations on arithmetic formats.

Exception handling: Indications of exceptional conditions (such as division by zero or overflow, for example).

© Copyright 2015 Xilinx

Slide 21-19:

Brief Look at a Floating Point Format

6295**slide

- Representation composed of three parts
  - Sign bit (1 for negative, 0 for positive)
  - Exponent
  - Mantissa or "fraction" or "significand"
    - Because this value is normalized, the most significant bit will always be a '1'; therefore, to save 1 bit, this most significant '1' is virtual. The fractional part is really '1.' + what is shown in the mantissa

- Definitions for 3 binary formats (32, 64,128 bits), and 2 decimal (64, 128)

6295!*note

This course is not about the IEEE 754-2008 standard, but rather a look at one possible mechanism for constructing a floating point engine.

The illustration is based on the IEEE 754 standard.

Slide 21-20:

Floating Point Special Values

6296**slide

- Often special meanings must be represented
  - ± 0, ± ∞
  - NAN – Not A Number

© Copyright 2015 Xilinx

- The library will set a result to one of these special values in the case of an exception
  - Divide by zero
  - Complex result

- The library offers a collection of generics which can enable and disable various warnings and describes how special cases are to be handled
  - Disabling certain special cases can result in a smaller implementation

© Copyright 2015 Xilinx

Slide 21-21:

Adding Floating Point Numbers

6297**slide

- Like fixed point numbers, the radices must be aligned
  - This requires de-normalization of the numbers so that the exponents are the same
  - If the exponents are more than the number of bits available in the significand, then the operation should not occur as this value will disappear when the number is renormalized
    - Example: If 1e64 and 1e-12 are added together, there are not enough significant bits available in the significand (in single precision) for the 1e-12 to register
  - Example: Add 1.125 and 32.5 together
    - Convert each to single precision
    - De-normalize (aligns radix points)
    - Add
    - Re-normalize (convert back to floating point standard)

Slide 21-22:

IEEE Fractional Math Libraries

6288**slide

- Officially designated IEEE 1076.3-2006
  - Designed for use with IEEE 1076-2006
    - However a compatibility version supports IEEE 1076-1993
  - Found under IEEE_PROPOSED library
- Includes
  - Fixed point package
  - Floating point package
  - Math utility package

© Copyright 2015 Xilinx

- Provided with the Vivado® simulator for all parts

- Vivado synthesis engine supports IEEE_PROPOSED for all 7 series FPGAs
  - Older families: User must manually point to
  <xilinx install directory>\VHDL\src\IEEE_PROPOSED

Slide 21-23:

Common Functions (1)

6299**slide

- Unary functions
  - Absolute Value: abs(), "abs"
  - Negate: "-"

- Basic math
  - +, -, *, /, rem, mod – Uses the default rounding and normalization rules
  - Add, subtract, multiply, divide, remainder, modulo – Enables the designer to specify how certain conditions are handled

- Extended math
  - Reciprocal
  - Dividebyp2 – Efficient division by a power of 2
    - Directly manipulates the exponent without actually doing a division (divide by shift)
  - Mac – Multiply and accumulate
  - sqrt

6299!*note

List of functions that are common to both the fixed and floating point packages.

© Copyright 2015 Xilinx

Slide 21-24:

Common Functions (2)

10498**slide

- Logical/status and comparative
  - Is_Negative
  - and, nand, or, nor, xor, xnor
  - eq, ne, lt, gt, le, ge, =, /=, <, >, <=, >=
- Other functions
  - maximum, minimum
  - logb
- Conversions
  - Resize, normalize
  - to_float, to_float32, to_float64, to_float128
  - to_slv, to_suv
  - to_signed, to_unsigned, to_real, to_integer, to_01
  - to_ufixed, to_sfixed

Slide 21-25:

Common Functions (3)

6300**slide

- Extends capabilities of TEXTIO
  - Read/write
    - Binary, octal, hex
    - Conversions to/from string

© Copyright 2015 Xilinx

Slide 21-26:

Math Utility

6301**slide

- Package contains types that describe how the functions in other packages are to operate
  - fixed_round_style_type can be fixed_round or fixed_truncate
  - fixed_overflow_style_type can be fixed_saturate or fixed_wrap
  - round_type can be
    - round_nearest – Default, nearest LSB '0'
    - round_inf – Round toward positive infinity
    - round_neginf – Round toward negative infinity
    - round_zero – Round toward zero (truncate)

Slide 21-27:

Representing Floating Point Numbers

6302**slide

- Floating point numbers
  - Defined as type of float (an array)
  - The most significant bit represents the sign
  - The MSb-1 downto 0 represents the exponent
    - The exponent is offset by 128 to represent positive and negative powers
  - The remaining bits represent the significand – 1

6302!*note

© Copyright 2015 Xilinx

Remember that the normalization process forces the first ‘1’ in the number into the most significant place in the mantissa. Because the ‘1’ is always there, to double the range of the mantissa, the one is removed and considered implied.

So, the MSb-1 downto 0 represents the mantissa as stored in the system, but to reclaim the real value, a 1 needs to be added to it.

For example, if the number 1.25 were to be stored, it might be represented as 101. Because there is a 1 in the most significant place, the number does not need to be normalized (that is, have the exponent adjusted).

Therefore the value that is actually stored in the mantissa portion of the number is “01”, not “101”.

© Copyright 2015 Xilinx

Slide 21-28:

Deciding on Fixed or Floating Point

6303**slide

	Fixed Point	Floating Point
Ease-of-use	Requires manual index adjustment to compensate for size of the result	Once the number of bits is selected (32,64,128), no manual adjustment is required
Performance	Fairly quick – can use block RAMs to implement additional functions (sine, sqrt, …)	About 1/7th the performance for the test program
Resources	Fewer fabric resources – some may map to DSP48x's	More processing required; therefore, more hardware needed
Range/Accuracy	Depending on the number of bits applied, accuracy around zero can be very good, but range can suffer	Compromise between range and accuracy

6303!*note

A simple add and multiply was constructed to test performance and resource utilization. For the fixed point design (s10:-6), approximately 60 slice registers, 68 LUT/flip-flop pairs and 1 DSP48E was inferred. The performance was slightly in excess of 350 MHz in a Virtext-6-2 device.

The roughly equivalent floating point design ran about 50 MHz in the same part and consumed 167 slice registers, 1449 LUT/flip-flop pairs, and used 2 DSP48Es.

Block RAMs can be preprogrammed with specific functions (such as trigonometric, and square and cube roots, for example). The same cannot be said for floating point.

Bottom line: Floating point is much easier to implement, but at a considerably greater cost.

Slide 21-29:

© Copyright 2015 Xilinx

Floating Point Example

6304**slide

```
signal flp_A  : float(8 downto -23);  -- 1:8:23 for 32 bit single precision
   . . .
signal flp_AC : float(8 downto -23);
```

Slide 21-30:

Summary

6305**slide

- Overview
- Fixed Point Numbers
- Floating Point Numbers
- **Summary**

Slide 21-31:

Apply Your Knowledge

6306**slide

31) Why would one use fixed point when floating point is available?

6306!*note

32) Why would one use fixed point when floating point is available?
Faster and fewer resources are required.

Slide 21-32:

© Copyright 2015 Xilinx

Summary

6307**slide

- IEEE supports fixed and floating point synthesizable packages
- Users can build their own fixed and/or floating point packages
  - IEEE is generalized; user may have specific qualities that may result in faster, smaller code
- Real types are not synthesizable and can vary from implementation to implementation

Slide 21-33:

Where Can I Learn More?

29545**slide

- IEEE – A fully synthesizable single-precision, floating-point adder/subtractor and multiplier in VHDL for general and educational use
- IEEE 754 – 2008 floating point standard

© Copyright 2015 Xilinx

Capture Your Notes Here

© Copyright 2015 Xilinx

Synchronous Design Techniques

Slide 22-1:

112506**slide

2019.2
112506!*note

This module describes the benefits of using synchronous design techniques. 20 minutes.

Slide 22-2:

Objectives

112507**slide

After completing this module, you will be able to:

- Increase circuit reliability and performance by applying synchronous design techniques

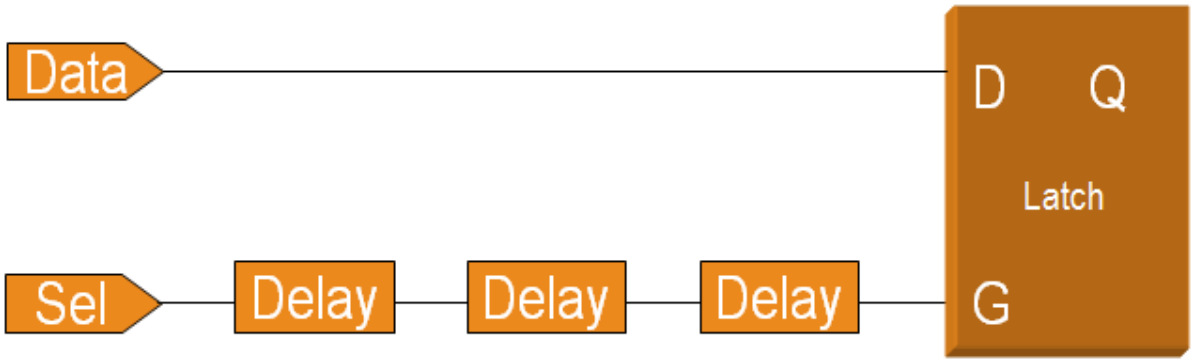
Slide 22-3:

Limitations of Asynchronous Circuits

112510**slide

© Copyright 2019 Xilinx

- Asynchronous circuits are less reliable



112510!*note

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).

For reliability reasons, Xilinx strongly recommends synchronous design. Asynchronous circuits may work correctly in one device but fail in another identical device due to normal variations of delays inside the chip.

Review of a case study: A design that was created two years ago no longer works. What did Xilinx change in its FPGAs? SRAM process improvements and geometry shrinks increase speed. Also, normal variations between wafer lots can occur, which means you may receive a faster device than is labeled on the package.

This 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.

Slide 22-4:

Synchronous Circuits

112508**slide

- A fully synchronous design uses only one clock and one clock edge

- Synchronous design methodology
  - All flip-flops use rising or falling edge (use D-type flip-flops)

© Copyright 2019 Xilinx

  - Partition hierarchy along functional lines while minimizing the interaction of blocks
  - Register the outputs of each behavioral block
  - Use clock enables in place of multiple clocks
  - Synchronize asynchronous signals to the *single* clock (synchronization circuits)
  - No problem of races or hazards

- Do *NOT*
  - Gate clocks in RTL
  - Create derived or divided clocks
  - Create ripple counters
  - Use global set / reset

112508!*note

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.

- Synchronous designs are inherently reliable.
- Static timing analysis is fast and reliable.
- Fewer clocks provide for easier design and debug.
- High performance can be achieved through pipelining.
- Interface timing is more predictable.

Note that global asynchronous set / reset is also discouraged because it hurts device utilization.

The designer of a synchronous system must perform three well-defined tasks:

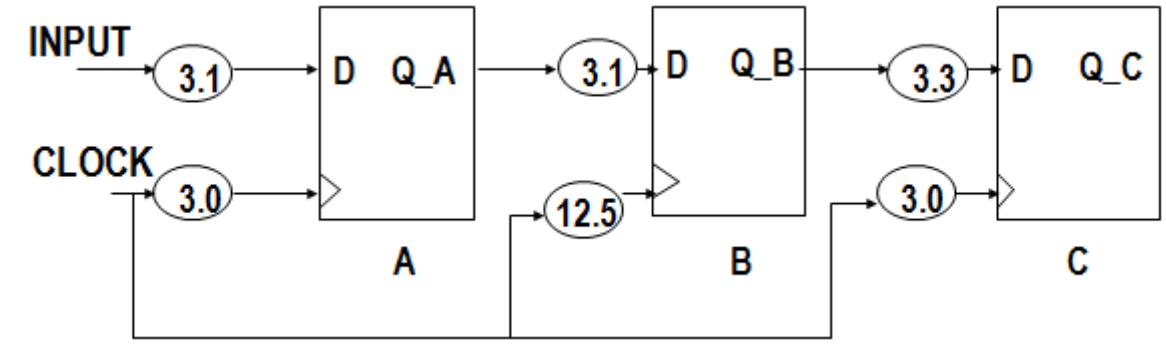
- Minimize and determine the amount of clock skew in the system
- Ensure that flip-flops have positive setup and hold time margins, including an allowance of clock skew
- Synchronize asynchronous inputs with the clock and ensure that the synchronizers have low probability of failure

Slide 22-5:

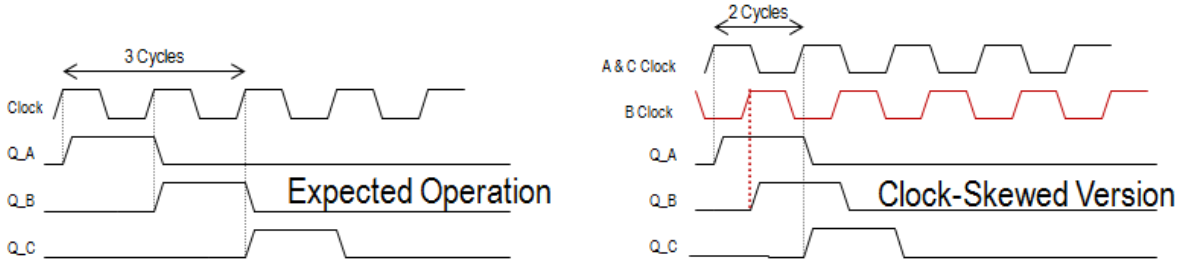
© Copyright 2019 Xilinx

Clock Signal Controls Events in Synchronous Circuits

112511**slide



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



112511!*note

In synchronous circuits, events are controlled by the clock signal. If the clock arrives at different flip-flops at different times, your circuit may not function correctly. This is an example of a clock skew problem.

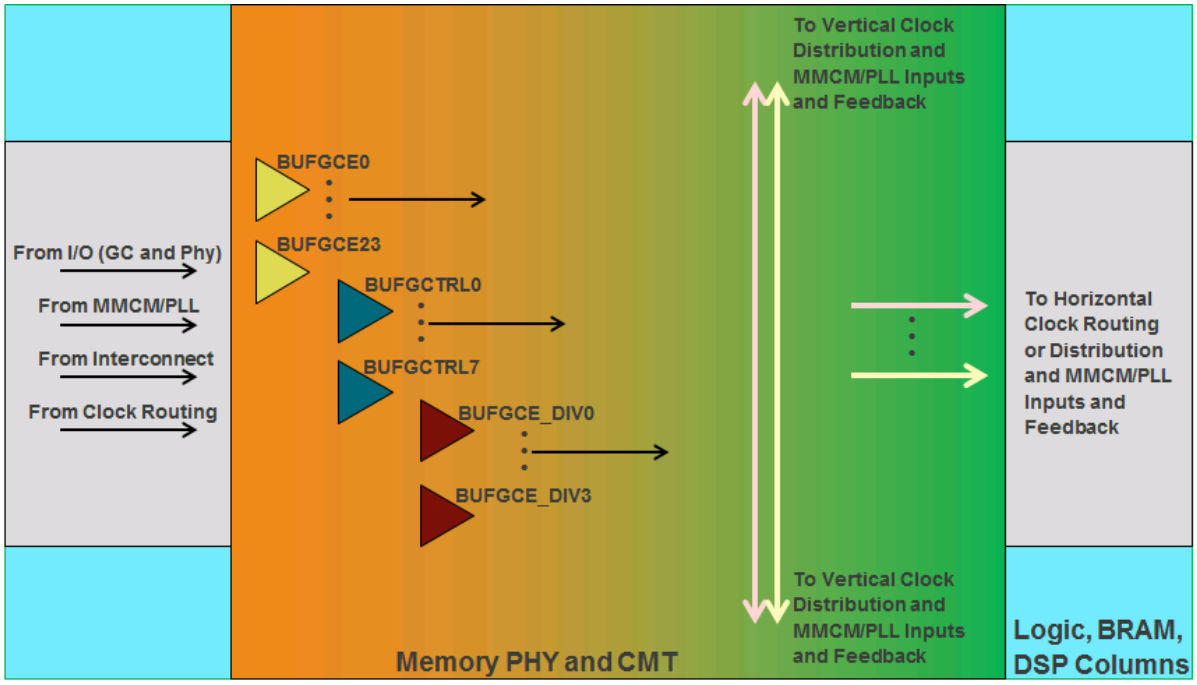
This is easily controlled by making certain that your clocks are on clock distribution nets. Use global clock buffers (BUFGs) in the design to ensure that clocks are using clock distribution net. The synthesis tool infers these BUFGs when the clock signal is connected to a clock capable pin.

© Copyright 2019 Xilinx

Slide 22-6:

Dedicated Clock Buffers to Reduce Clock Skew

112512**slide



112512!*note

Global clock buffers located in the memory PHY column.

24 BUFGCE, 8 BUFGCTRL, 4 BUFGCE_DIV per PHY.

Any 24 of the buffers can drive horizontal/vertical.

In this (and every other) junction there are a total of 36 clock buffers. All of these clock buffers have similar connectivity.

- They take their clock inputs from I/O, MMCM, PLL, interconnect or clock routing.
- They drive their clock outputs vertically to get to adjacent regions or MMCM/PLL inputs or feedback, and horizontally to get to the horizontal clock tracks and MMCM/PLL inputs or feedback in the same region.

Once the output clocks have been driven onto the clock distribution and routing tracks, the segmented clock routing determines what CRs use the clock.

- These clocks may end up clocking resources in as few as one clock region, or potentially all clock regions.

© Copyright 2019 Xilinx

The three different types of buffers (BUFGCE, BUFGCTRL, and BUFCE_DIV) offer different clock enable, multiplexing and division capabilities.

All major synthesis tools can infer global buffers onto clock signals that come from off chip. I/O and regional clock buffers cannot be inferred. Internally generated clocks are not automatically placed on a clock buffer.

The easiest way to design your clocking *core* is with the Clocking Wizard tool.

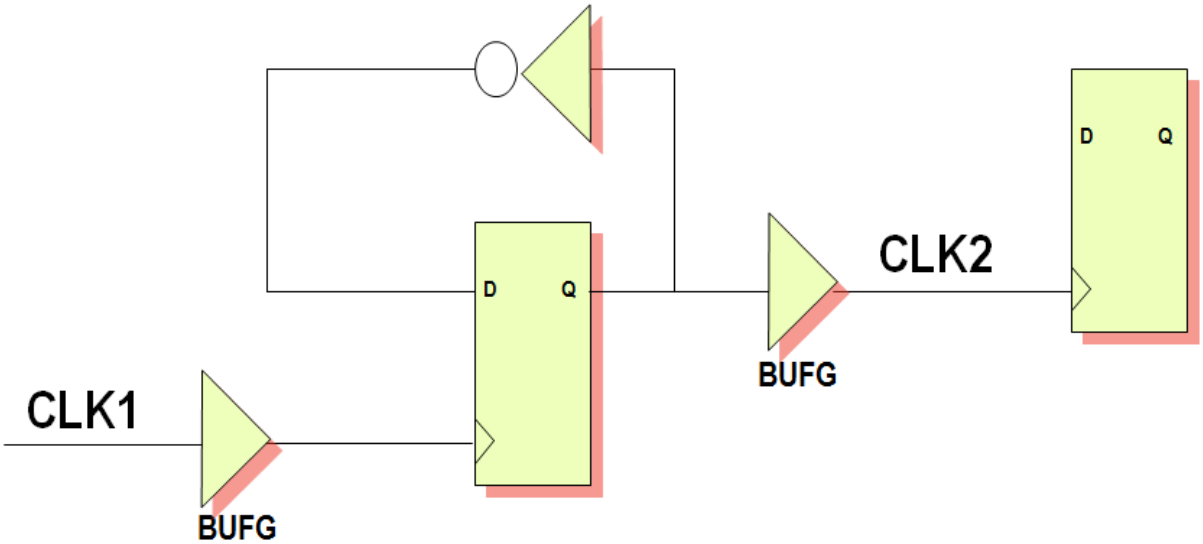
Note that instantiation templates for various clock buffer configurations are available in the language templates in the Vivado® Design Suite.

Slide 22-7:

Traditional Clock Divider

112513**slide

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



112513!*note

© Copyright 2019 Xilinx

This is a typical circuit that divides CLK1 by two. This circuit has a potential problem: skew between CLK1 and CLK2. If this skew does not cause a problem in your design, it is okay to use this circuit.

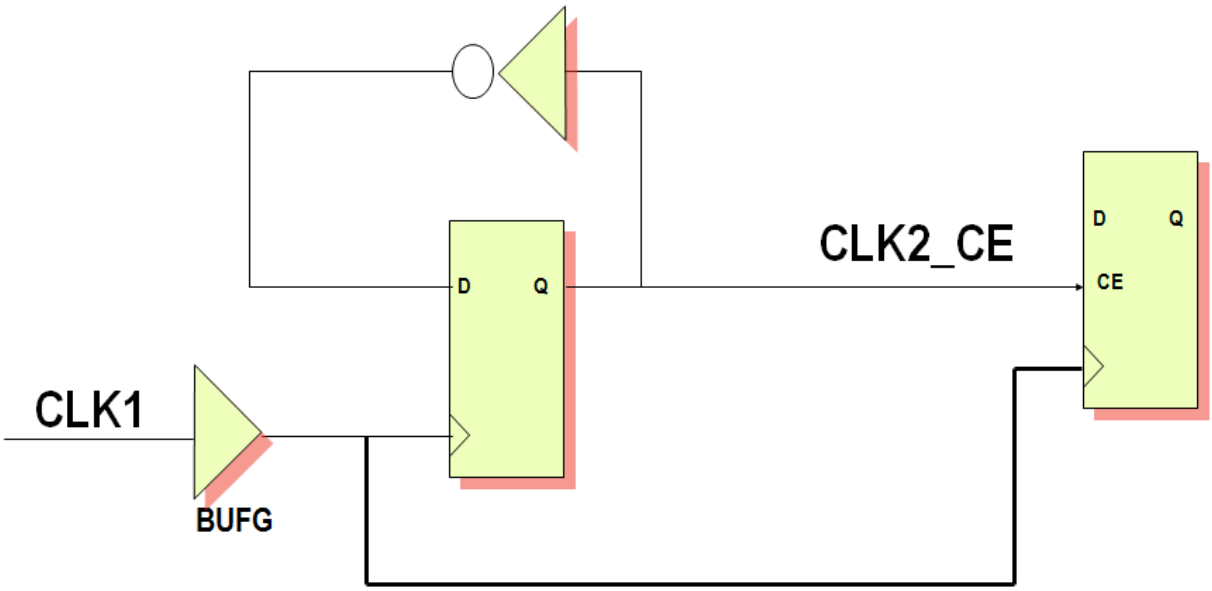
© Copyright 2019 Xilinx

Slide 22-8:

Recommended Clock Divider

112514**slide

- No clock skew between flip-flops



112514!*note

This implementation is very similar, but it eliminates the skew because all flip-flops are clocked with CLK1.

You can also use the dedicated clock management features to generate a divided clock with no skew.

© Copyright 2019 Xilinx

Slide 22-9:

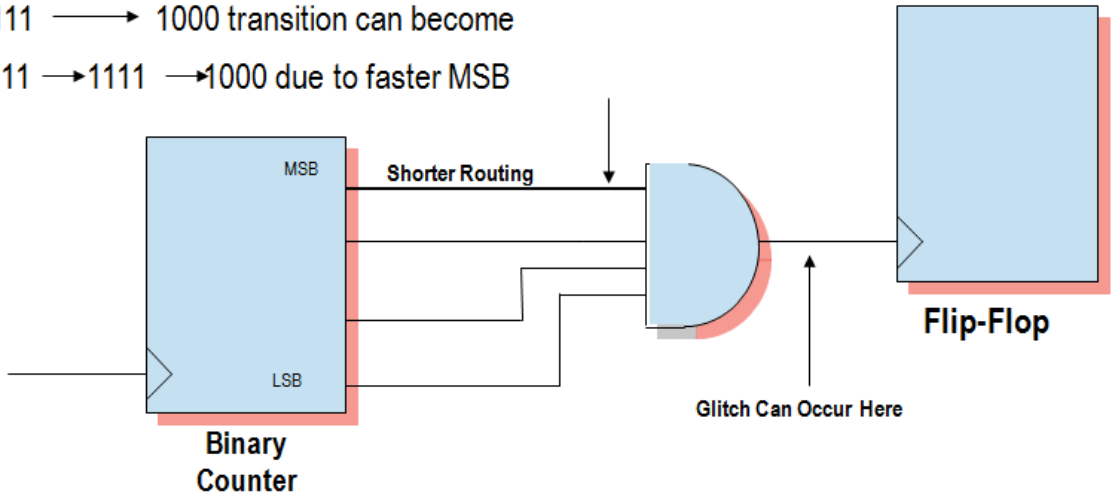
Clock Glitches

112515**slide

MSB

0111 → 1000 transition can become

0111 → 1111 → 1000 due to faster MSB



112515!*note

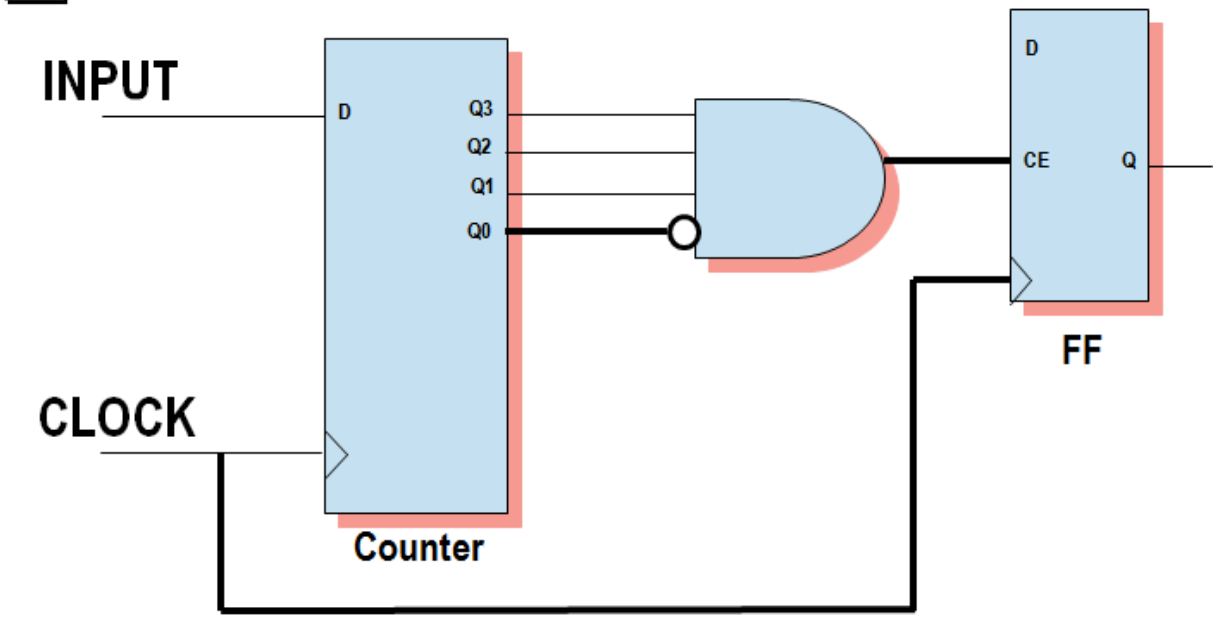
Because flip-flops in today's FPGAs are very fast, they can respond to very narrow clock pulses. Never source a clock signal from combinatorial logic, which is also known as *gating the clock*. Again, because it is such an important signal, you must avoid glitches on the clock.

Slide 22-10:

© Copyright 2019 Xilinx

Avoid Clock Glitches

112516**slide



112516!*note

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

Notice two things:

- The output of the AND gate is connected to the clock enable of the flip-flop instead of being connected to the clock pin.
- The AND gate is decoding the 1110 state of the counter.

Slide 22-11:

Coding Clock Enables

112517**slide

© Copyright 2019 Xilinx

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;
```

112517!*note

If ENABLE is not a top-level port, write the code for ENABLE in another process. This makes the code more readable and helps the synthesis tools create better netlists.

The Set / Reset signal has precedence over Clock Enable. Use this order in your code.

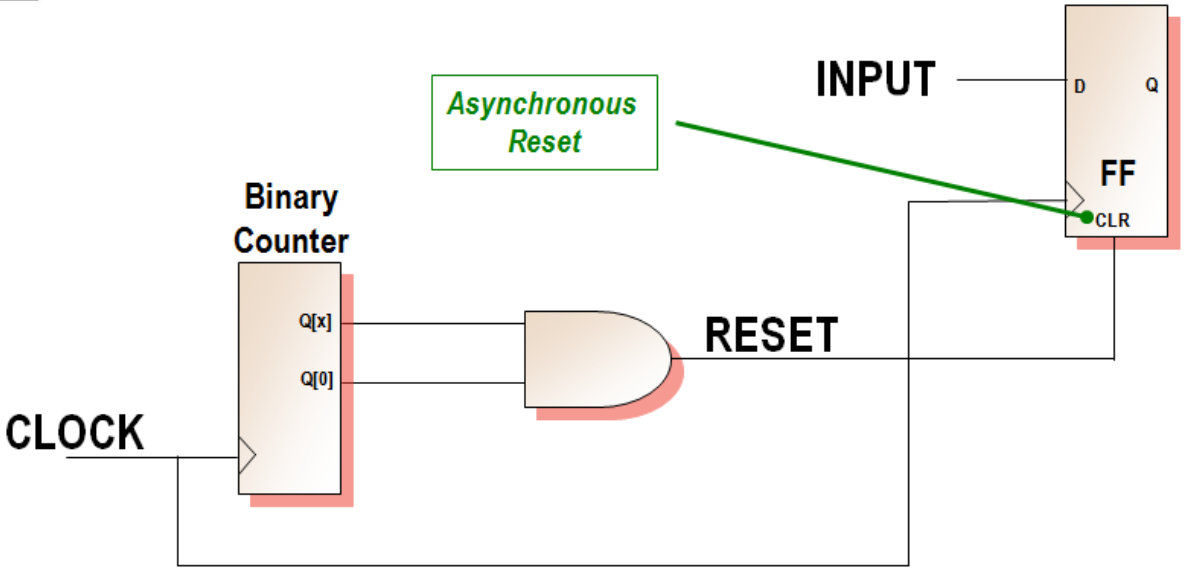
If you do not follow the correct order of precedence in your HDL code, the synthesis tool may build extra logic.

© Copyright 2019 Xilinx

Slide 22-12:

Set/Reset Glitches

112518**slide



112518!*note

Glitches on asynchronous set and asynchronous reset inputs can lead to incorrect circuit behavior. Other than the clock, there may be other signals in your design that must be glitch-free.

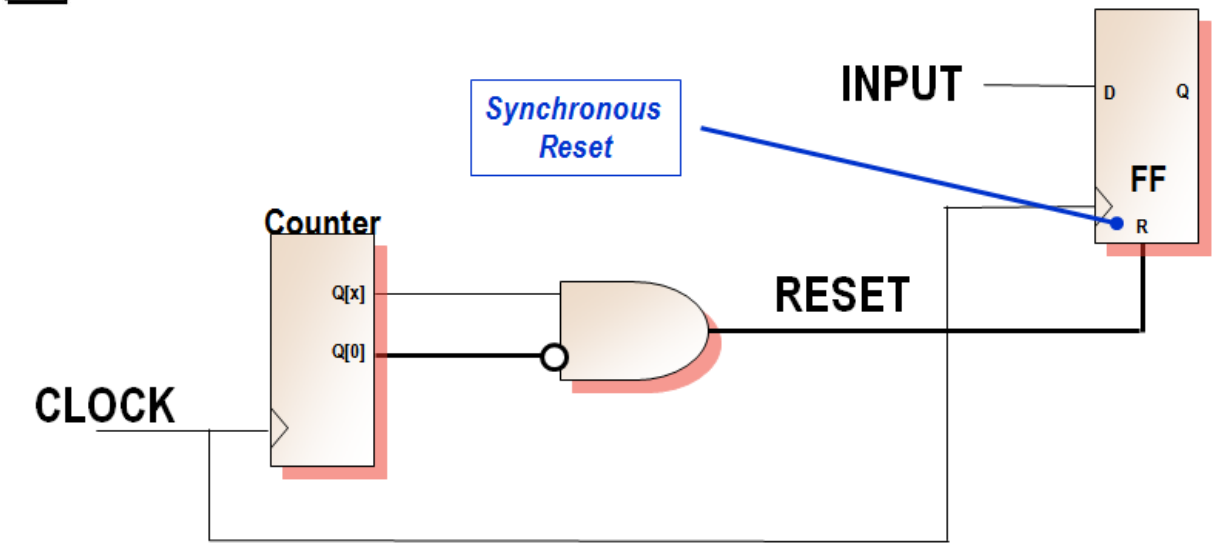
One example is asynchronous sets or resets. In this circuit, a glitch could occur on RESET during the 01 -> 10 transition of the counter output (01 -> 11 -> 10).

Slide 22-13:

© Copyright 2019 Xilinx

Avoid Set/Reset Glitches

112519**slide



112519!*note

Convert to synchronous set and synchronous reset when possible.

Convert to synchronous reset by modifying your HDL code.

Notice that the lower AND gate is decoding the 10 state of the counter instead of the 11 state; therefore, the flip-flop resets when the counter transitions to 11.

This is covered further in the *HDL Coding Techniques* RELs.

Slide 22-14:

© Copyright 2019 Xilinx

Coding Synchronous Flip-Flops

112520**slide
Verilog

```
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;
```

112520!*note

Asynchronous Reset (VHDL)

```
process(CLK, RESET)
begin
   if (RESET = '1')    then
      Q <= '0';
   elsif (CLK'event and CLK = '1') then
      Q <= D_IN;
   end if;
end process
```


© Copyright 2019 Xilinx

Synchronous Reset (VHDL)

```
process(CLK)

begin
   if (CLK'event and CLK = '1') then
      if (RESET = '1')    then
         Q <= '0';
      else
         Q <= D_IN;
     end if;
end if;

end process
```

Slide 22-15:

Summary

121399**slide

- Asynchronous circuits are less reliable
- Dedicated clock buffers are used to reduce the clock skew
- Clock glitches can be avoided by using clock enables

Slide 22-16:

Apply Your Knowledge

112521**slide

33) Why should you use global buffers for your clock signals?

34) What is an alternative to gating a clock?

© Copyright 2019 Xilinx

112521!*note

35) Why should you use global buffers for your clock signals?

Use global buffers for your clock signals to reduce clock skew.

36) What is an alternative to gating a clock?

Using a clock enable.

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Designing with the IP Integrator

Slide 23-1:

```
108930**slide

2019.2
108930!*note
```

This module introduces the IP integrator flow in the Vivado® Design Suite. 15 minutes.

© Copyright 2019 Xilinx

Slide 23-2:

Objectives

108931**slide

After completing this module, you will be able to:

- Describe the IP integrator (IPI)
- Create an IPI block design
- Explain the Tcl command support in IPI
- Describe the board tab features in IPI

© Copyright 2019 Xilinx

Slide 23-3:

What Is the Vivado IP Integrator?

108932**slide

- A hierarchical IP integration tool for processor based and non-processor-based systems
- A graphical and scriptable IP configuration and connection environment
  - Not a schematic entry tool
- Create complex systems
  - Add, customize, connect, create hierarchy, and connect blocks
- Domains support
  - Integration of algorithmic (Vivado HLS and System Generator) and RTL-level IP
  - Combination of DSP, video, analog, embedded, connectivity, and logic
- IP support within Reconfigurable Modules for RTL-based projects
  - Set any IP within RM to global synthesis
  - XCI cannot be top level of RM
  - Each XCI must have a unique name, even if it is in a different RM

108932!*note

The Vivado IP integrator is a graphical and scriptable block-based IP connection tool that allows users to create complex systems with accelerated IP Integration. Complex IP blocks are added to a diagram and rapidly interconnected together. The power of the IP integrator is that it enables interface-level connections (such as the AXI4 interface).

The IP integrator can make a connection between an IP containing an AXI master interface to another IP block containing an AXI slave interface in seconds. This connection is shown as one wire on the diagram, but represents several hundred individual signals.

© Copyright 2019 Xilinx

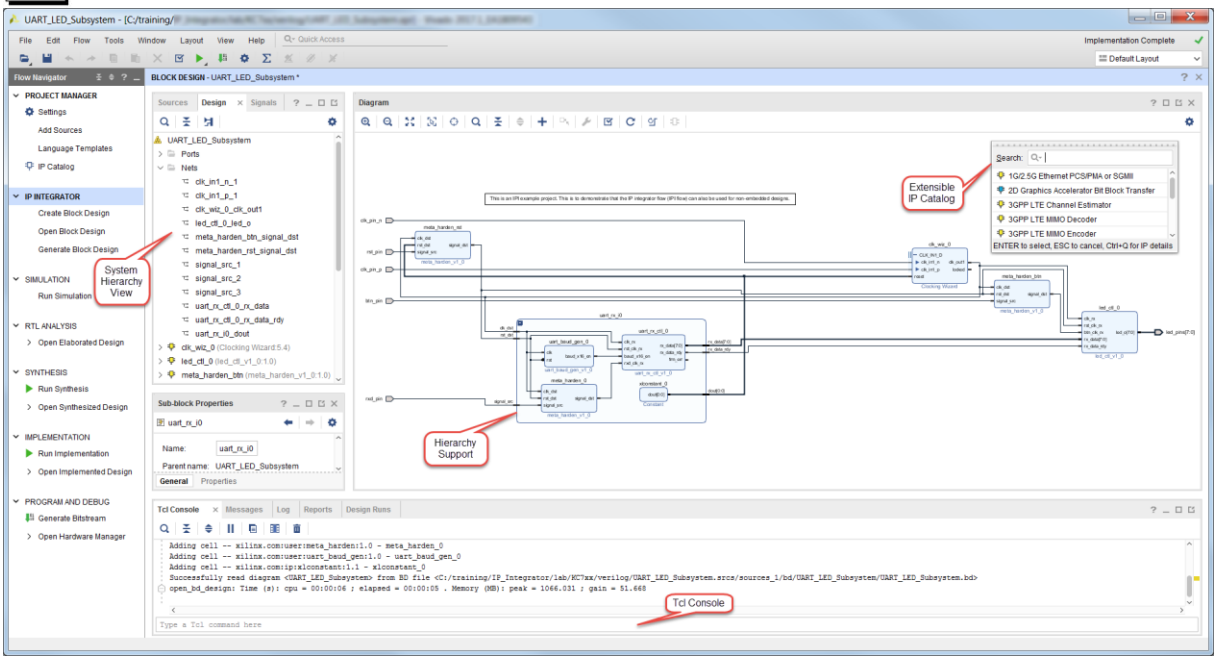
Because the IP integrator is part of the Vivado IDE, advanced users can create Tcl scripts that rapidly assemble and parameterize complex systems.

The IP integrator supports all classes of designs: pure logic, connectivity and networking designs, DSP designs, and also embedded designs. The IP integrator supports embedded processor design for the Zynq® SoC, and also MicroBlaze™ processor designs.

Slide 23-4:

Vivado IP Integrator: Intelligent IP Integration

108981**slide



108981!*note

The Vivado IP integrator creates complex systems by instantiating and interconnecting IP from the Vivado IP catalog on a design canvas.

You can add any IP to the canvas and interconnect them to create a block design.

© Copyright 2019 Xilinx

Slide 23-5:

Vivado IP Integrator: Features

108933**slide

- Interactive design canvas
- Platform aware
- Interface-level connections or simple net connections
- Create interface or simple ports
- Designer assistance
- Parameter propagation and resolution
- Web browser-based HTML interactive report or text output
- Hierarchy support
- Real-time DRCs
- Exporting of hardware definition to SDK
- Tcl support
- Layer visibility
- Powerful block diagram differencing tool
- Support for Vivado HLS and System Generator designs

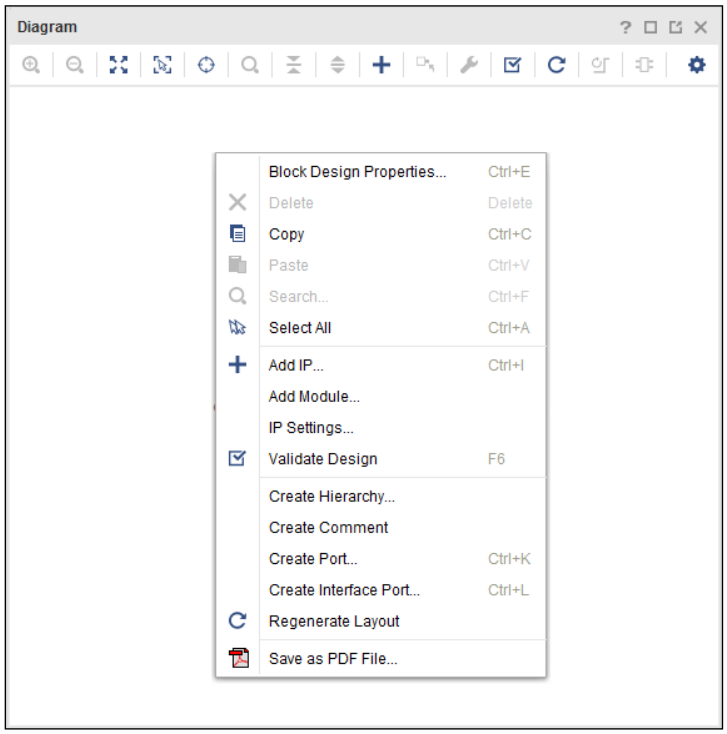
© Copyright 2019 Xilinx

Slide 23-6:

Design Canvas

108934**slide

- Design canvas
  - Workspace for creating block diagrams
  - IP catalog is integrated, easy to search for IPs
  - Validate a design
  - Regenerate a design
  - Create ports, interfaces
  - Make connections
  - DRCs are included to verify the design
  - Design assistance available
  - Support of locking placement on design canvas



108934!*note

The design canvas is the workspace for working with block diagrams in the Vivado Design Suite GUI. Design assistance is available within the design canvas to help you create your design.

IPI supports arbitrary levels of hierarchy. You can create an empty level of hierarchy or automatically move selected blocks. Design hierarchy can be cross-probed and the hierarchy is preserved during generation.

© Copyright 2019 Xilinx

When you are making connections, the cursor changes into a pencil when it is moved near an interface or pin connector on an IP block. If you want to make any connection, click an interface or pin connector on an IP block, press the left mouse button, and then draw the connection to a destination block. To create an interface port, right-click an interface or pin connector on the IP block and select **Create Interface Port**.

Ports enable signals in the block design to penetrate the boundary to connect with other IPs in the system. The ports in the block design are equivalent to an "entity" statement in VHDL or a "module" statement in Verilog.

When you are creating a signal as a port, the Create Port Wizard requires the following information:

- Name
- Direction
- Bus size
- Type of signal (clock, interrupt etc.)

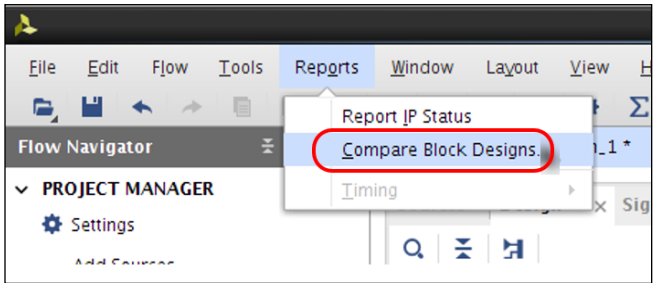
For best results, add all the desired IP to the canvas and then run block and connection automations as opposed to running the automations on an incremental basis (i.e., after adding each IP). This is because the tool takes into account the entire system at the time the automation is performed. So although automations are done on a per IP basis, results can vary depending upon the surrounding IP.

Slide 23-7:

Vivado IP Integrator: New Features

149736**slide

- Vivado Design Suite IP integrator new features include
  - Hide or color nets/blocks for better visibility in the block diagram



  - Powerful block diagram differencing tool
  - Web bowser-based HTML interactive report or text output

© Copyright 2019 Xilinx

- Cross-probing to Vivado Design Suite canvas with multi-user (network) capability
- Advanced filtering in interactive mode
- Performance improvement: Reduction in time to open or re-create block designs
- Block diagram format will be automatically saved in JSON-format vs. IP-XACT (XML)

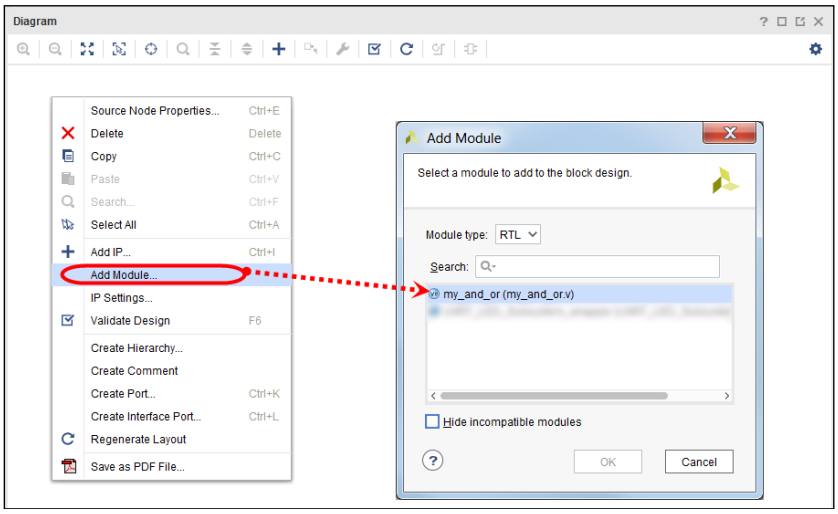
© Copyright 2019 Xilinx

Slide 23-8:

Module Referencing

129260**slide

- RTL file needs to be added to the project
- The Add Module dialog box lets you add a module in the current block design
- It displays the modules that are available to add to the block design
- If the entity/module name changes in the source RTL file, the referenced module instance must be deleted from the block design and a new module should be added



129260!*note

© Copyright 2019 Xilinx

Modules referencing in IPI is an alternate way of adding the IPs in your block design.

An RTL module in the block design has an “RTL” marking on the component symbol.

If you delete a referenced module from the block design, then the module is moved outside the block design hierarchy in the Sources window.

A block design containing a module reference cannot be packaged as an IP. Instead, you could separately package the module as an IP, and then package the BD including that IP.

In comparison of module referencing with the packaged IP, packaging the IP is lengthy process but adds the IP into IP catalog to allow that IP to be used in any other design, whereas module referencing is an easier and less time-consuming process but does not add the IP to the IP catalog.

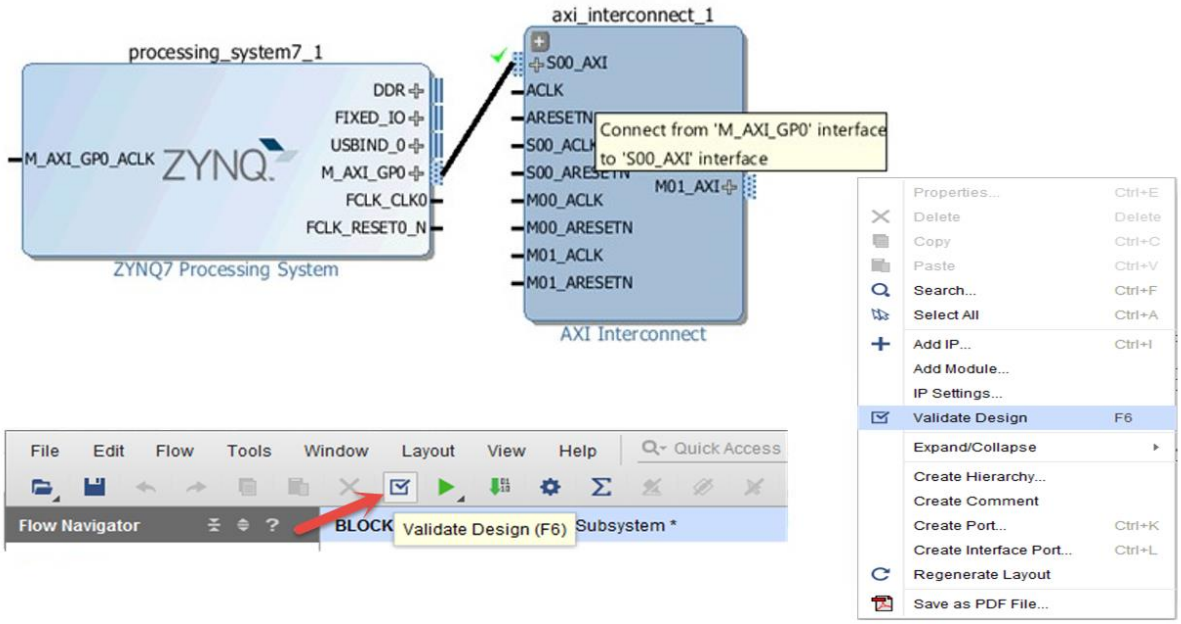
© Copyright 2019 Xilinx

Slide 23-9:

DRCs

108986**slide

- DRCs are run multiple times
  - While making connections
  - By running the **Validate Design** command



108986!*note

The IP integrator runs basic design rule checks in real time as the design is being put together. Once the block diagram is completed, Xilinx recommends that you run the Validate Design command.

© Copyright 2019 Xilinx

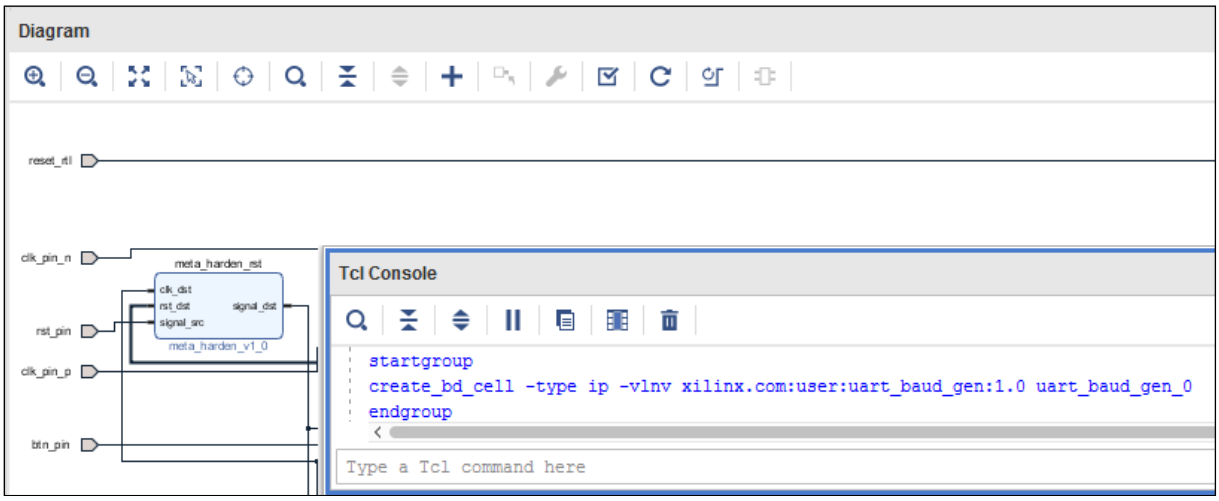
Slide 23-10:

Using Tcl in IP Integrator

108987**slide

- Every action in the IP integrator results in a Tcl command; reflected in Tcl Console

- write_bd_tcl is a good way to learn IPI Tcl
  - Writes Tcl script to recreate the current state of a design
  - Returns `TCL_OK` if successful or `TCL_ERROR` if it fails, unless `-quiet` is specified



108987!*note

Refer to the IP Integrator section in the *Vivado Design Suite Tcl Command Reference Guide* (UG835) for more information about IP integrator Tcl commands.

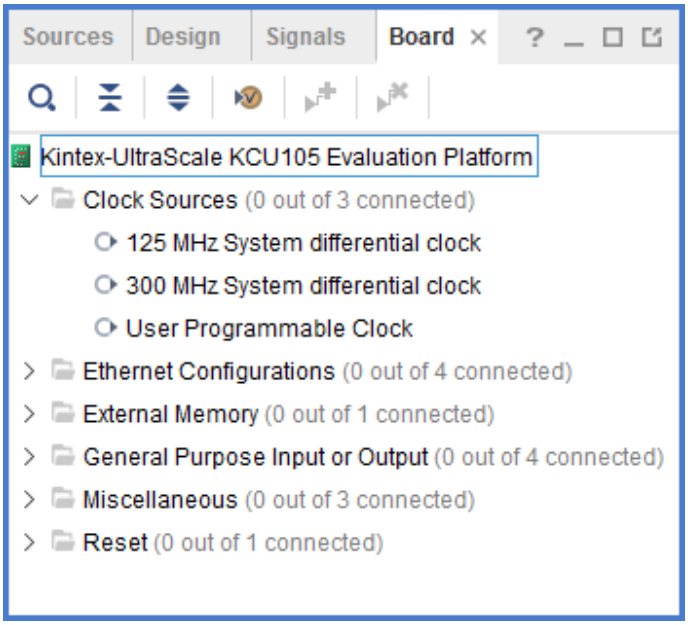
© Copyright 2019 Xilinx

Slide 23-11:

Board Tab

108938**slide

- IP integrator is board-aware
  - The Board tab provides easy access to interfaces available on selected hardware platform
  - Automates IP connection to interfaces
    - Allows easy access to IPs supported by corresponding interface
  - Available only after creating or opening block design in IP Integrator



108938!*note

The IP integrator shows all the interfaces to the board in a separate tab called the Board tab. When you use this tab to select the desired interfaces and Design Assistance is offered by the IP integrator, you can easily use available interfaces on the target hardware.

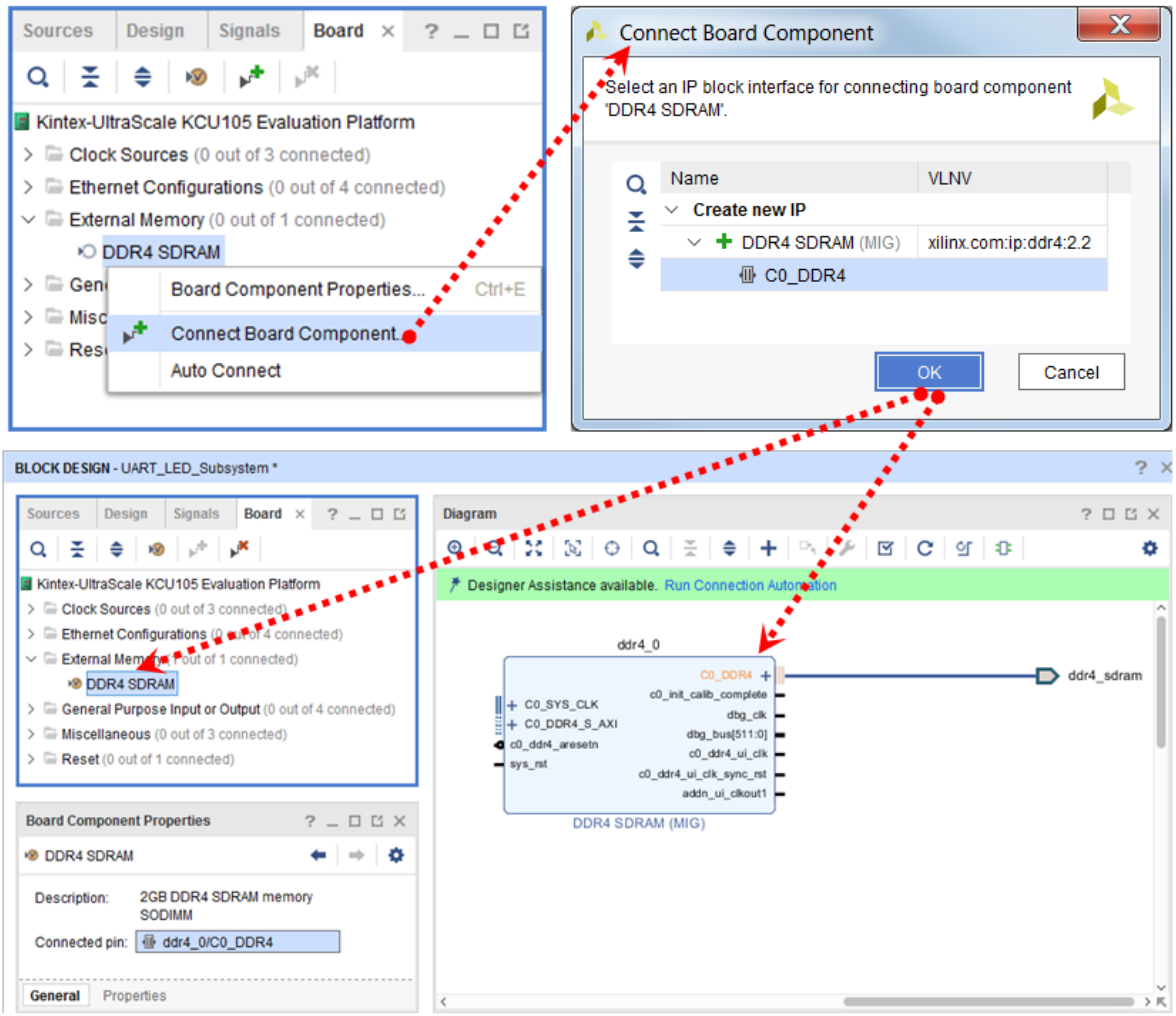
© Copyright 2019 Xilinx

על הטקסט שברצונך שיופיע כאן.Heading 1שגיאה! השתמש בכרטיסיה בית כדי להחיל

Slide 23-12:

Using Board Flow to Create a Block Design

108939**slide



108939!*note

Steps to create a block design using IP integrator board flow:

37) Click **Create Block Design** in the Flow Navigator.

38) Select the **Board** tab.

39) Select the desired Board Component ; DDR4 SDRAM, for example, from the Board tab.

© Copyright 2019 Xilinx

The Connect Board Component dialog box opens.

40) Select from the list of possible IP that can be connected to the selected component.

41) Click **OK** in the dialog box.

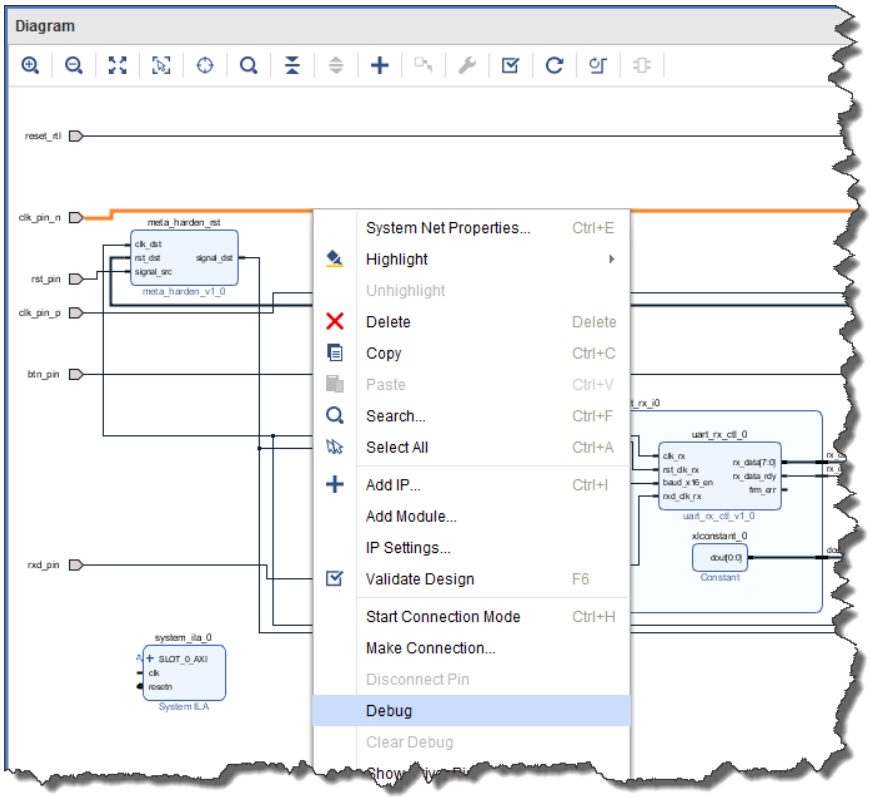
Notice that DDR4 SDRAM (i.e., MIG IP) is added to the deign canvas and is also displayed in the Connected Board section in the Board tab.

Slide 23-13:

Debugging IPI Designs

121463**slide

- Signals or interfaces can be marked for debugging
  - This automatically sets up waveforms in the HDL simulator to view
  - Also specifies signals to debug in hardware



Slide 23-14:

© Copyright 2019 Xilinx

Summary

108937**slide

- Vivado IP integrator
  - Is platform aware with real-time DRCs
  - Can build and reuse IP subsystems
  - Supports integrated debugging
  - Supports GUI and Tcl interfaces
  - Raises the level of design abstraction and increases design productivity

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Pipelining

Slide 24-1:

```
108596**slide
2019.1
108596!*note
```

This module describes how to improve design performance by pipelining the design.
10 minutes.

Slide 24-2:

Objectives

```
108604**slide
```

After completing this module, you will be able to:

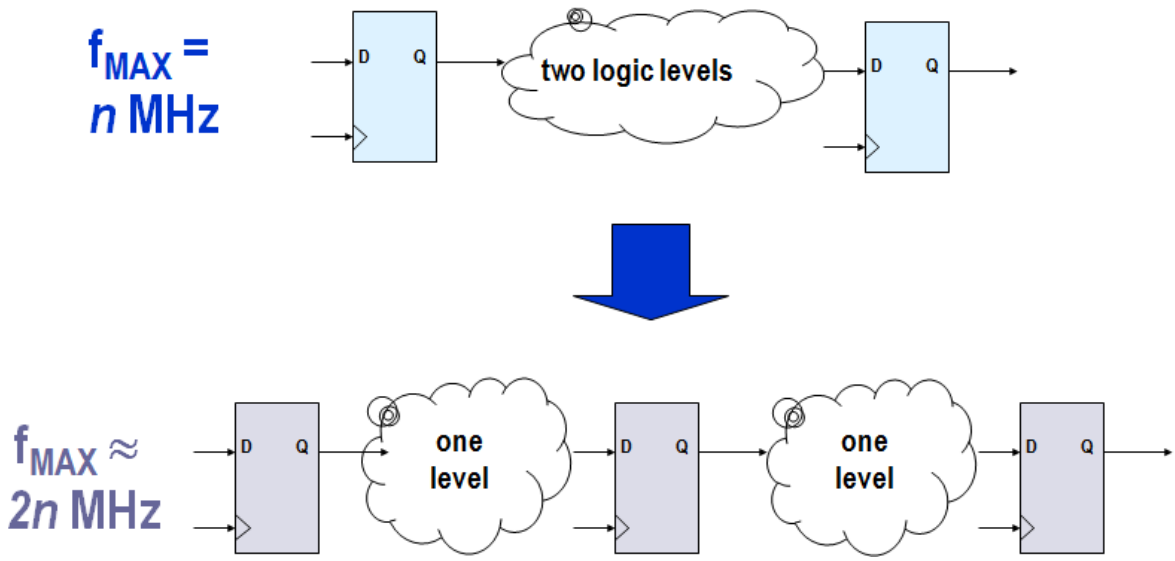
- Describe the pipelining technique that improves the throughput of a design

Slide 24-3:

© Copyright 2019 Xilinx

Pipelining Concept

121044**slide



121044!*note

© Copyright 2019 Xilinx

Inserting flip-flops into a datapath is called pipelining.

Pipelining breaks up the complex process that takes several clock cycles into smaller pipeline stages separated by registers. These registers hold the values passing from one stage to the next pipeline stage.

Pipelining increases performance by reducing the number of logic levels (LUTs) between flip-flops. Pipelining helps to reduce the critical path.

All Xilinx FPGA device families support pipelining. The basic slice structure is a logic level (6-input LUT for the UltraScale™ FPGAs) followed by a flip-flop.

Adding a pipeline stage, as shown in this example, will not exactly double f_{MAX}. The flip-flop that is added to the circuit has an input setup time and a clock-to-Q time that make the pipelined circuit run at less than double the original frequency, hence the clock speed is increased.

Slide 24-4:

Pipelining Considerations

121045**slide

- Are enough flip-flops available?
  - Refer to the utilization report after synthesis or implementation
  - 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 timing report after implementation
- Can the system tolerate *latency*?

121045!*note

Available flip-flops: The utilization report contains resource utilization information. In most cases, you will have enough flip-flops available to add pipeline stages.

© Copyright 2019 Xilinx

For highly utilized designs, unnecessary pipeline stages increase the number of flip-flops and routing resources, which limits place and route algorithm.

Logic levels: Timing reports show which paths are the longest and how many logic levels are in each path.

Implementing pipeline techniques results in higher utilization of flip-flops, which in turn results in more power consumption.

It is always better to consider pipelining upfront rather than later on, as it improves timing closure, implementation run time, and device power.

© Copyright 2019 Xilinx

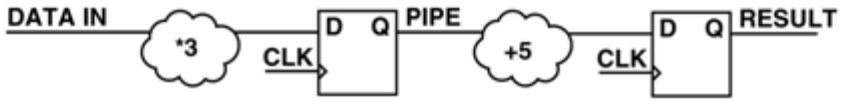
Slide 24-5:

Latency in Pipelines

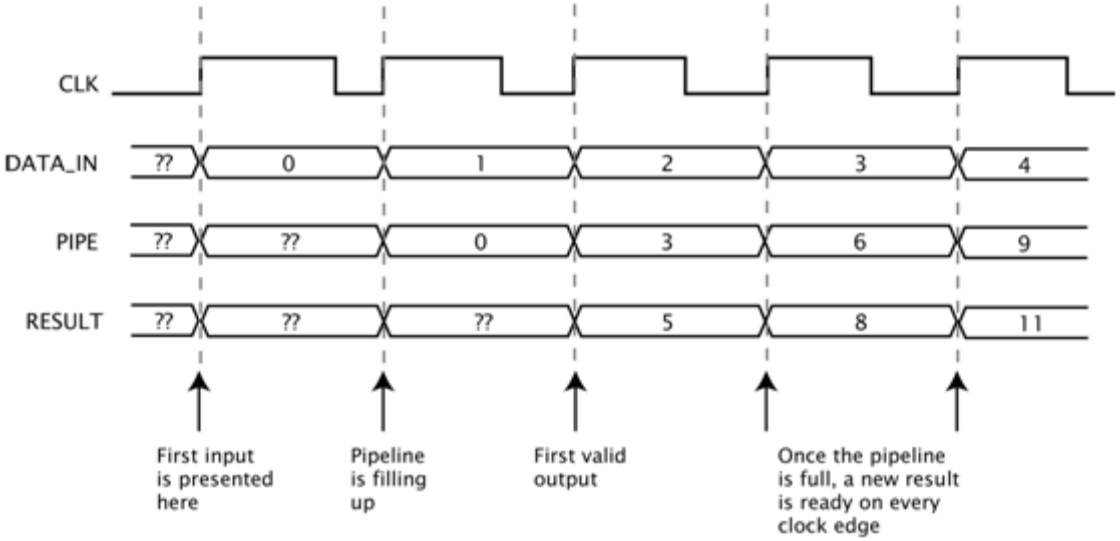
108599**slide

- 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



RESULT=(3*DATA IN) +5



108599!*note

© Copyright 2019 Xilinx

This is an example of a pipelined circuit. Follow the data as it goes through the multiplication and then the addition.

Latency in pipelines can be visualized as a factory assembly line. Raw materials enter the assembly line and then go through several stations. At each station, one production step is performed. When you start the assembly line, you have to wait before the first finished product is produced. After that waiting period, a constant stream of products comes off the assembly line.

Some operations inherently cannot be pipelined. In general, feedback makes pipelining difficult. For example, it is not possible to pipeline most counters or state machines; the internal state on any given clock is determined by the state in the previous clock—no additional pipeline stage can be placed in this feedback path.

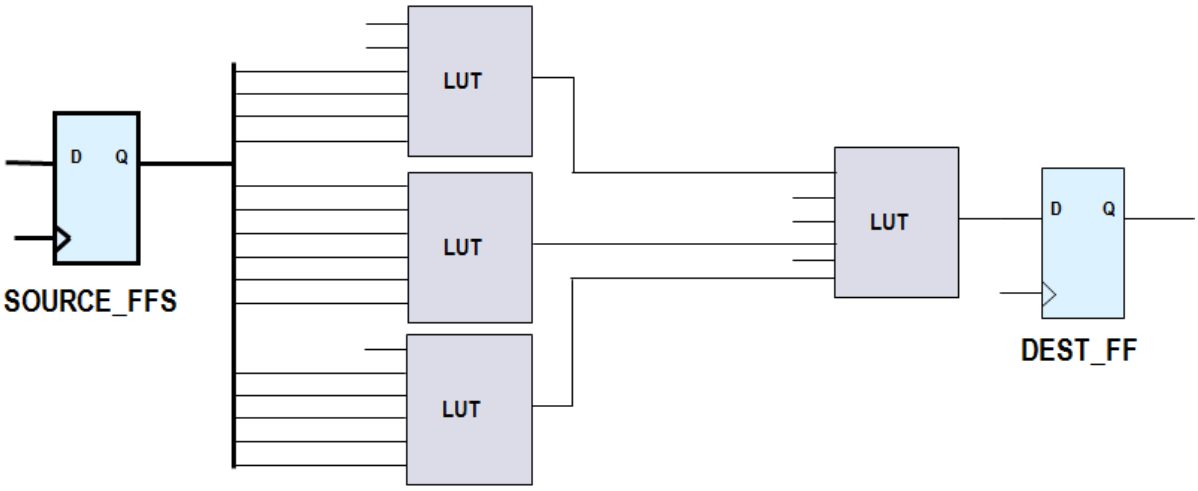
© Copyright 2019 Xilinx

Slide 24-6:

Pipelining Example

108600**slide

- Original circuit
  - Two logic levels between SOURCE_FFS and DEST_FF
  - f_{MAX} = ~251 MHz



108600!*note

This path has two logic levels between SOURCE_FFS and DEST_FF. The first logic level is the column of three LUTs in parallel. The second logic level is the single LUT on the right.

The delays in this path are:

- SOURCE_FFS clock-to-Q
- Net delay
- LUT delay
- Net delay
- LUT delay
- DEST_FF setup time

Estimating no clock skew and routing delays to be approximately 1.5 ns each, this circuit could run at about 251 MHz.

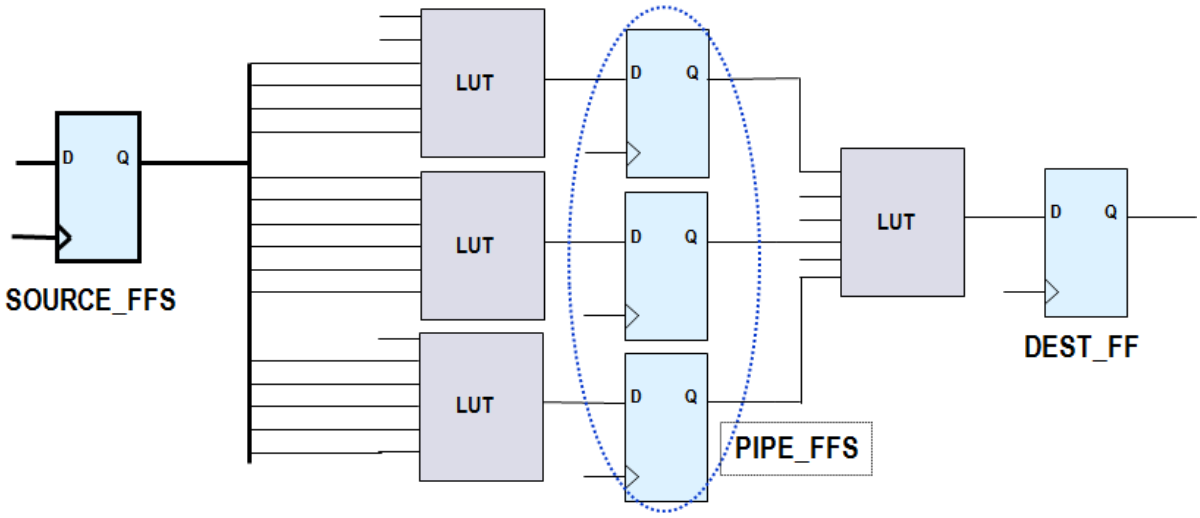
© Copyright 2019 Xilinx

Slide 24-7:

Pipelining Example

108601**slide

- Pipelined circuit
  - One logic level between each set of flip-flops
  - f_{MAX} = ~417 MHz



108601!*note

© Copyright 2019 Xilinx

After adding a pipeline stage, the circuit has been split into two paths. The first path is from SOURCE_FFS through one logic level to PIPE_FFS. The second path is from PIPE_FFS through one logic level to DEST_FF.

The delays in either path are:

- Starting FF clock-to-Q
- Net delay
- LUT delay
- Ending FF setup time

This circuit will have higher throughput but longer latency.

Estimating each routing delay to be approximately 1.5 ns, as in the original circuit, this circuit could run at about 417 MHz (a 66% increase).

Adding the registers between improves the timing by dividing the critical path into two paths with smaller delay.

© Copyright 2019 Xilinx

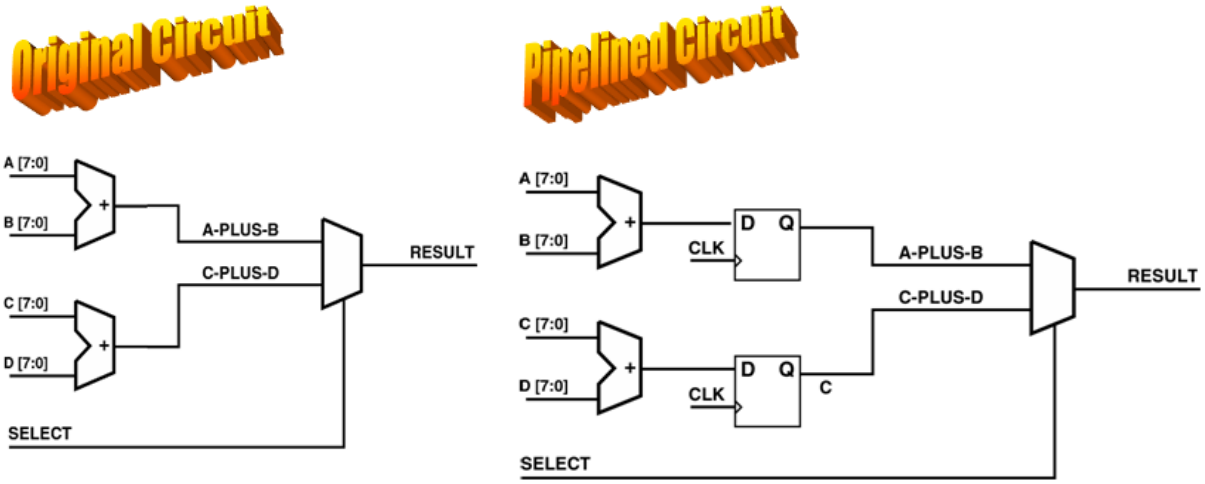
Slide 24-8:

Apply Your Knowledge

108602**slide

42) Given the original circuit, what is wrong with the pipelined circuit?

43) How can the problem be corrected?



108602!*note

This simple circuit is a multiplexed adder where the SELECT signal determines whether the output is (A + B) or (C + D).

The designer decides to add a pipeline stage to the circuit to improve performance, but something is not quite right. What is the correct way to pipeline this circuit?

© Copyright 2019 Xilinx

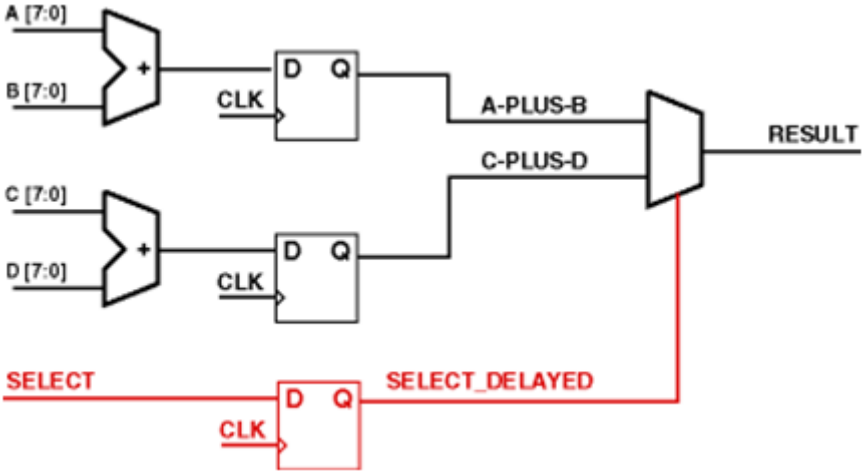
Slide 24-9:

Answers

121046**slide

44) What is wrong with the pipelined circuit?

- Latency mismatch
- Older data is mixed with newer data
- Circuit output is incorrect



45) How can the problem be corrected?

- Add a flip-flop on *SELECT*
- All data inputs now experience the same amount of latency

121046!*note

© Copyright 2019 Xilinx

When the designer simulated the circuit, the designer saw that the multiplexer was not selecting the correct sum. The designer forgot to add a pipeline stage to the SELECT input.

This concept of matching latency in pipelines can be easily overlooked, especially when you have multiple pipelined circuits in your design.

If the outputs of two different pipelines need to be combined, you must ensure that the pipelines have the same amount of latency. You can use the Shift Register LUT (SRL) feature of the UltraScale devices to match latency without using large numbers of flip-flops.

Block RAMs and DSPs within Xilinx FPGAs offer pipelining capabilities. Xilinx recommends using pipelines within dedicated primitives like block RAMS and DSPs.

While implementing the multiplier in DSP, Vivado® synthesis takes advantage of the pipelining capabilities of DSP blocks. Pipelining properly for logic inferred into the DSP block can greatly improve performance and power.

In the "Pipelining" lab of this topic cluster, you will analyze the performance, latency, and timing closure impacts of a pipelined design.

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

Synchronization Circuits

Slide 25-1:

```
108646**slide
2019.1
108646!*note
```

This module introduces the most commonly used synchronization circuits. 20 minutes.

Slide 25-2:

Objectives

```
108648**slide
```

After completing this module you will be able to:

- Identify scenarios that require synchronization circuits
- Design simple metastability resolution circuits
- Describe bus coherency and bus synchronizers
- Use clock-crossing FIFOs

© Copyright 2019 Xilinx

Slide 25-3:

Synchronization Circuits

121171**slide

- What is a synchronization circuit?
  - 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 clock constraints may be sufficient
  - Chip inputs that are asynchronous

121171!*note

For clock domains that have a clearly defined and constant phase relationship, proper timing constraints ensure that there are no setup or hold time violations.

When the setup and hold times of a flip-flop are not met, the flip-flop could be put into the metastable state. The time to transition to a stable state is unbounded.

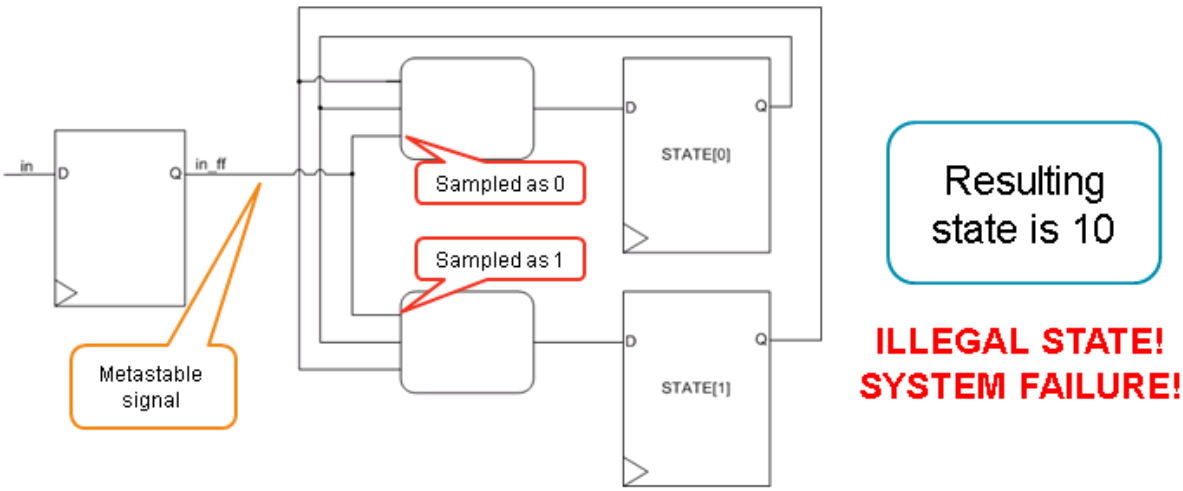
© Copyright 2019 Xilinx

Slide 25-4:

Risk of Metastable Signal

121172**slide

- A metastable signal is neither a logic 0 nor a logic 1
  - Digital logic receiving a metastable signal may
    - Behave as if the metastable signal is a logic 0
    - Behave as if the metastable signal is a logic 1
    - Also go metastable



121172!*note

To build a state machine, synthesis infers the **next state** logic in front of the D input of each bit of the state flip-flops. In this case, the next state logic for both flip-flops is dependent on the metastable signal *in_ff*.

Each of the two next state logic bubbles may interpret the metastable signal as a 0 or 1. If both interpret it as 0, then the next state will be 00. If both interpret it as 1, then the next state will be 11. Both of these states are “correct.” However, if one interprets it as a 1 and the other as a 0, then the state machine will enter the 10 or 01 state; these are both incorrect states.

In this case, the metastable event has caused a system error.

For a flip-flop, computing MTBF helps to avoid system failure.

© Copyright 2019 Xilinx

Mean Time Between Failure (MTBF) is exponentially related to the length of time that the flip-flop is given to recover. A few extra ns of recovery time can dramatically reduce the chances of a metastable event causing a circuit failure.

<recovery time> = <time before the data is used> - <datapath delay>

Note: The <datapath delay> should include the normal clock_to_q delays and setup times required.

Example: If CLK1 has a period of 10 ns, and the datapath delay is 4 ns, then the recovery time of FF1 is (10-4) = 6 ns.

Slide 25-5:

Metastability Resolution

108652**slide

- When a signal goes metastable it will eventually resolve to either a logic 0 or a logic 1
  - It is impossible to know to which state it will eventually resolve
  - The amount of time required before the signal resolves is probabilistic
    - The likelihood of the signal remaining metastable after time *t* decreases exponentially as *t* increases
    - The probability asymptotically approaches 0
      - Thus a metastable flip-flop is never guaranteed to resolve

- In order to reduce the probability of system failures due to metastability you must
  - Sample a potentially metastable signal with only one receiver
  - Leave enough time for metastability to resolve

108652!*note

© Copyright 2019 Xilinx

Because metastability is never guaranteed to resolve, any asynchronous signal can (and eventually will) cause a system failure.

However, you can use circuits that leave some time for metastability to resolve; by leaving enough time, you can reduce the probability that a metastable event can cause system failure.

But how long is long enough? This question can only be answered (probabilistically) in terms of MTBF. The longer you wait, the lower the probability that a metastable event can cause system failure. However, the probability of failure will only be 0 if you wait an infinite amount of time. Thus, you need to design systems that are tolerant enough to metastability events in order to attain the required system MTBF.

The required MTBF of a system is highly dependent on the system's use. For example, a piece of health care equipment that provides life sustaining support (e.g., a mechanical ventilator) needs a far higher MTBF then a piece of consumer equipment (e.g., a cell phone). Therefore, the robustness of the metastability circuits used in these systems will differ.

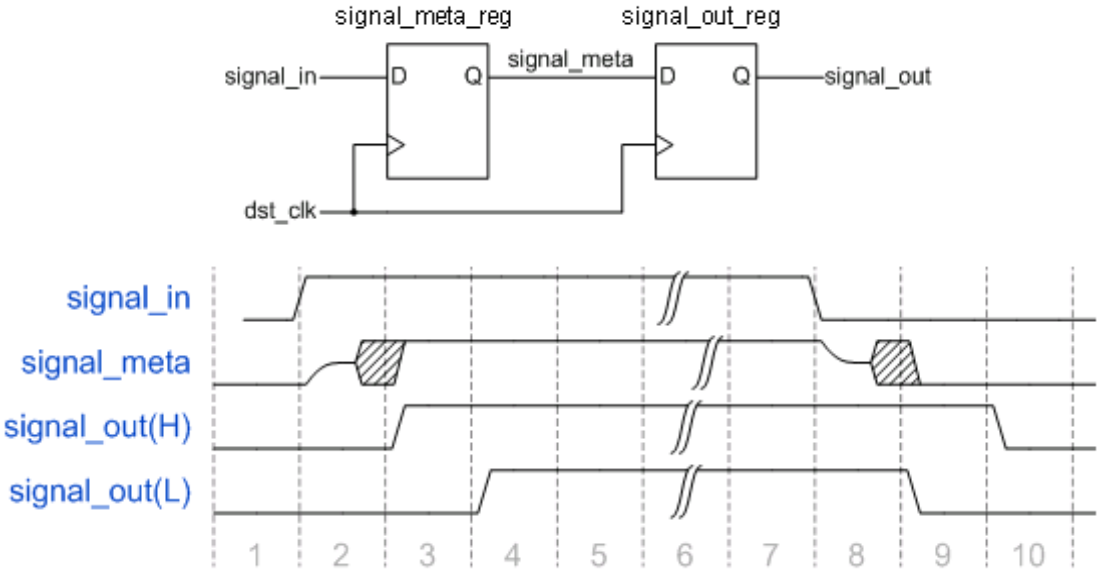
© Copyright 2019 Xilinx

Slide 25-6:

Metastability Resolution Circuit

108651**slide

- The simplest metastability resolution circuit is two flip-flops
  - The first flip-flop may go metastable, but it is sampled by only one load
  - Allows for up to one clock period for the metastability of the first flip-flop to resolve
- Used only for slow changing signals



108651!*note

Because *signal_in* is asynchronous to *dst_clk*, it will eventually change near the rising edge of the clock, violating the setup/hold window of *signal_meta_reg*. When this occurs, the flip-flop may stay at the old value, transition to the new value, or go metastable. If it goes metastable, there is a probability that the metastability will resolve before *signal_out_reg* samples the output of *signal_meta_reg*.

In either of these cases, the resulting signal sampled by *signal_out_reg* will be a stable value, but it is impossible to know if it will be the old value or the new value. Both cases are shown above; *signal_out(H)* is the resulting signal if the unknown signal is resolved as a logic 1, and *signal_out(L)* if it is sampled as a logic 0.

© Copyright 2019 Xilinx

Because *signal_in* is a slow changing signal, regardless of how it is sampled on the first clock edge after a state change, the second clock edge is guaranteed to sample the new value.

The simple metastability resolution circuit is sufficient for a slow changing single bit signal; subsequent changes on the input signal must be far enough apart for the first change to be resolved before the second change occurs.

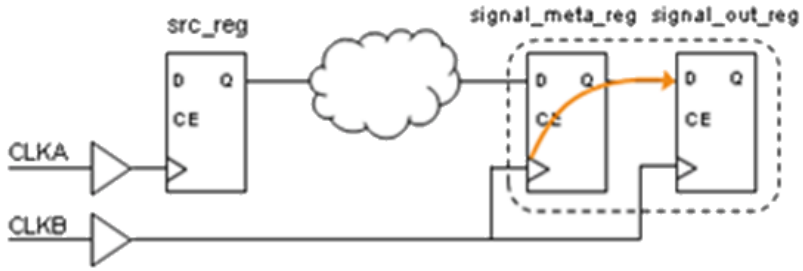
Slide 25-7:

Constraining the Single Bit Metastability Resolution Circuit

108653**slide

- The back-to-back flip-flops in the metastability resolution circuit allow time for a metastable event on the first flip-flop *(signal_meta_reg)* to resolve before the second flip-flop (*signal_out_reg*) samples it
- The Vivado® Design Suite timing engine sees the *signal_meta_reg* → *signal_out_reg* path as a normal static timing path, subject to a normal setup check
  - Allows one clock period of CLKB for the propagation, including routing
- To leave time for metastability resolution, the requirement on this path should be changed

```
set_max_delay -datapath_only -from [get_cells signal_meta_reg] - to [get_cells signal_out_reg] 2
set_property ASYNC_REG TRUE [get_cells {signal_meta_reg signal_out_reg}]
```



- The ASYNC_REG property should be set on both synchronizer flip-flops

108653!*note

© Copyright 2019 Xilinx

This constraint allows for T_{period}-2ns for the metastability event on *signal_meta_reg* to resolve before the single load (*signal_out_reg*) samples the signal. If *signal_meta_reg* is still metastable after this period, there is a finite probability that *signal_out_reg* will also go metastable, which can, in turn, cause system failure.

If this contributes too much MTBF, then additional time would be needed. In order to allow for additional metastability resolution time, you may need more than two back-to-back flip-flops for metastability resolution. This is particularly true if the frequency of CLKB is high.

By using three flip-flops (*signal_meta0_reg*, *signal_meta1_reg*, and *signal_out_reg*), the probability of a system failure is further reduced. By adding a third stage to the metastability resolution circuit with a similar constraint, even if the second flip-flop (*signal_meta1_reg*) goes metastable, there is a high probability that the metastability on this flip-flop will resolve in T_{period}-2ns before *signal_out_reg* samples it. Only if the metastability on *signal_meta1_reg* has not resolved when *signal_out_reg* samples it, and it also goes metastable is there a possibility of system failure.

The probability of system failure in a three-stage synchronizer is exponentially smaller than the probability of a two-stage synchronizer.

Setting the ASYNC_REG property on flip-flops prevents optimization of these cells during synthesis, `opt_design`, and `phys_opt_design` and forces packing of the flip-flops into the same slice.

Slide 25-8:

Constraints Between Different Clock Domains when Using the Single Bit Synchronizer

`108654**slide`

- All clocks in the Vivado Design Suite static timing engine are considered related
  - If the single bit asynchronous signal comes from another clock domain, the Vivado Design Suite timing engine will derive constraints for the path based on the clock relationship between the two clocks
  - If the clocks are unrelated then this requirement is invalid and needs to be overridden

© Copyright 2019 Xilinx

– Any of these constraints are acceptable for this clock-crossing circuit

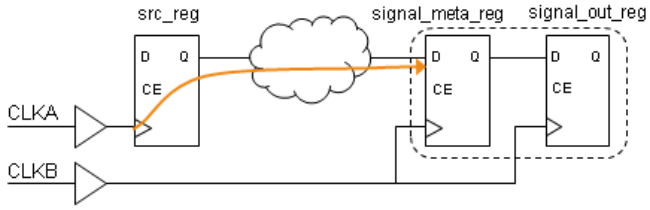
```
set_false_path -from [get_cells src_reg] -to [get_cells signal_meta_reg]
```

```
set_false_path -from [get_clocks CLKA] -to [get_clocks CLKB]
```

```
set_clock_groups -asynchronous -group CLKA -group CLKB
```

```
set_max_delay -from [get_cells src_reg] -to [get_cells signal_meta_reg] 5 -datapath_only
```

```
set_max_delay -from [get_clocks CLKA] -to [get_clocks CLKB] 5 -datapath_only
```



108654!*note

Note that if the input to *signal_meta_reg* comes from a primary input, the flip-flop *src_reg* does not actually exist. However, the relationship between CLKA (which may be a virtual clock) and the input arrival time in this circuit is normally constrained with a `set_input_delay` command.

For a single bit synchronizer, there is no expected time relationship between the signal change on the CLKA domain and the signal capture on the CLKB domain. In this case, it is acceptable to declare the path between *src_reg* and *signal_meta_reg* as a false path. This is what is done by the first constraint shown above.

If this is the **ONLY** path between these two domains, then it is also acceptable to declare the false path between the clocks. This can be done with the second or third constraint.

© Copyright 2019 Xilinx

Remember, though, that the `set_false_path` exception is the highest priority exception. If there are any other paths between CLKA and CLKB the clock-based `set_false_path` command (second constraint) or `set_clock_groups` command (third constraint) used for this synchronization circuit will affect all paths between these domains; there are synchronizers for which this is not an appropriate constraint.

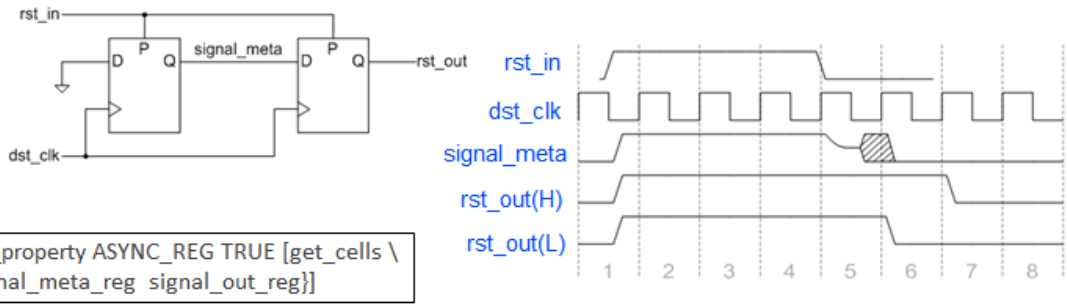
The `set_max_delay -datapath_only` constraint, which is more restrictive, is also acceptable for this synchronization circuit. As you will see later, this constraint is more appropriate for other synchronization circuits, and will have the desired effect of overriding the default requirement derived by the tools (thus also making it appropriate for this one as well).

Slide 25-9:

Reset Bridge

108655**slide

- The reset signal entering a synchronous system may not be synchronous to the clock
  - It needs to be synchronized
- Because it is a slow changing signal the simple metastability resolution circuit is sufficient
  - However it has the disadvantage of delaying the assertion of the reset
- A reset bridge can be used; the reset is asserted asynchronously immediately and de-asserted synchronously




```
set_property ASYNC_REG TRUE [get_cells \
{signal_meta_reg  signal_out_reg}]
```

108655!*note

© Copyright 2019 Xilinx

Regardless of whether the synchronous system uses asynchronous presets/clears or synchronous sets/resets internally, it is essential that the reset signal be synchronized to the clock.

For systems using synchronous sets/resets, the set/reset input of the flip-flop is simply another synchronous pin that has setup/hold requirements.

For systems using asynchronous presets/clears, the preset/clear input of the flip-flop must **not** be asynchronous to the clock; if the reset deasserts near the edge of the clock, the flip-flop may remain in the set/reset state, may assume the value of the D input, or may go metastable. The static timing engine checks paths that end in the "reset recovery" requirement for the flip-flop.

The reset bridge uses flip-flops with asynchronous presets; this ensures that the *rst_out* signal asserts immediately when the *rst_in* is asserted.

When the *rst_in* signal deasserts near the rising edge of *dst_clk* the first flip-flop may stay in the preset state (logic 1), may sample the input (logic 0), or may go metastable. Like the simple metastability resolution circuit, this circuit provides some time for the metastability to resolve before the second flip-flop samples the potentially metastable signal.

Slide 25-10:

Bus Coherency

108656**slide

- A data bus is more than just a collection of bits
  - The bits of the bus together form a piece of data
    - The bits form a code which represents some information
      - A count value, a character, a system state
- When bringing this data into a new clock domain it is required that the data not be corrupted
  - This requires more than simply dealing with metastability
  - You must ensure that the bits of the data stay coherent
    - All the bits of a coded datum remain together

© Copyright 2019 Xilinx

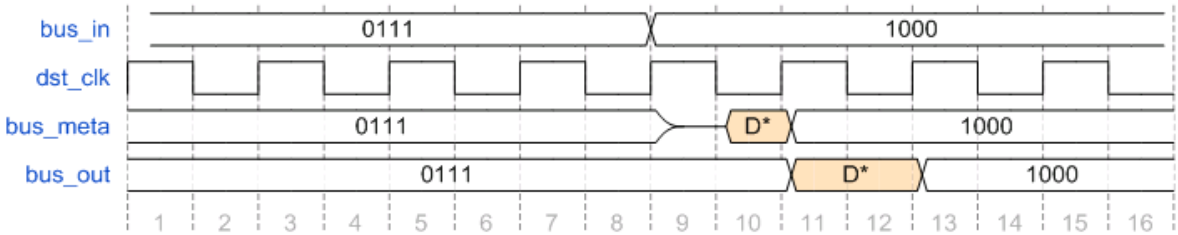
Slide 25-11:

Incorrect Bus Coherency

121173**slide

- Except under certain conditions, it is not sufficient to simply use metastability resolution circuits on each bit of a bus
  - Will cause bus coherency errors

- Consider a binary counter transitioning from the state 0111 → 1000
  - All four bits are transitioning; hence all four sampling flip-flops may go metastable
  - Using a metastability resolution circuit on each bit, the metastability of each bit will resolve to either a 0 or a 1
    - The resolved bus can therefore be any value from 0000 to 1111
      - This is a system error because it is neither of the two correct codes



121173!*note

If the bus synchronization circuit is done correctly, the cycle marked D* should either be the old value (0111) or the new value (1000).

If one uses independent metastability resolution circuits on each bit of the bus, each bit that is transitioning may resolve to either its old value or its new value. For example, if bits 0 and 2 stay at the old value but bits 1 and 3 take the new value, then D* will be 1101. This is an incorrect value, and hence the system has failed.

Slide 25-12:

© Copyright 2019 Xilinx

Capturing a Bus – Solutions

108640**slide

- Solutions
  - Sample bus only when it is known to be stable
  - Ensure only one bit is changing at a time (Gray code)
  - Use a clock-crossing FIFO

© Copyright 2019 Xilinx

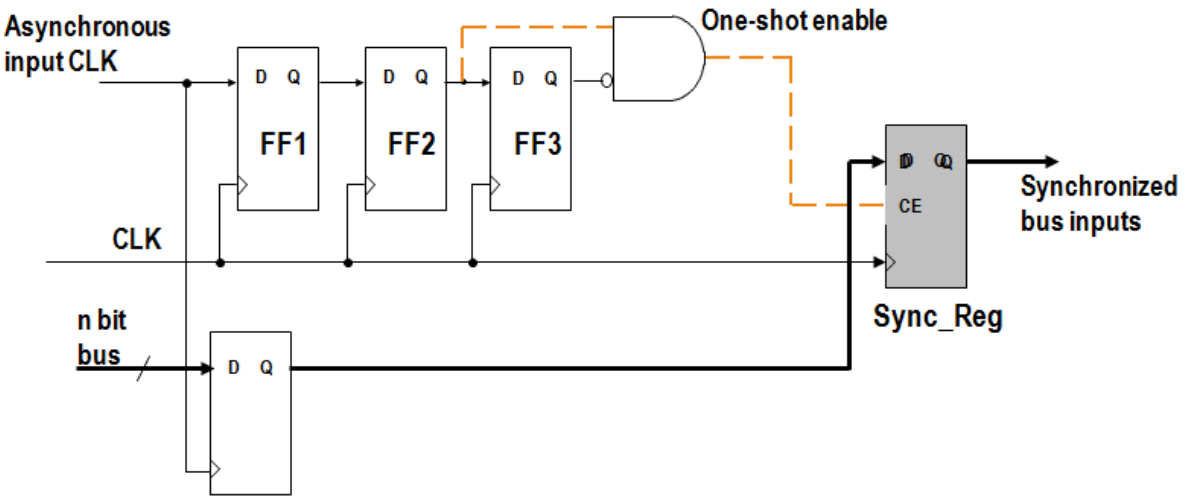
Slide 25-13:

Capturing a Bus When Stable (1)

108641**slide

- Assuming the source clock is much slower
  - Can use rising edge of source clock to identify time where bus is stable

- Leading edge detector



108641!*note

First, the data bus is registered by the source (slow) clock.

Second, the source clock is metastability hardened into the destination domain.

Finally, the rising edge of the synchronized signal is detected, which is used to capture the bus into the destination clock domain.

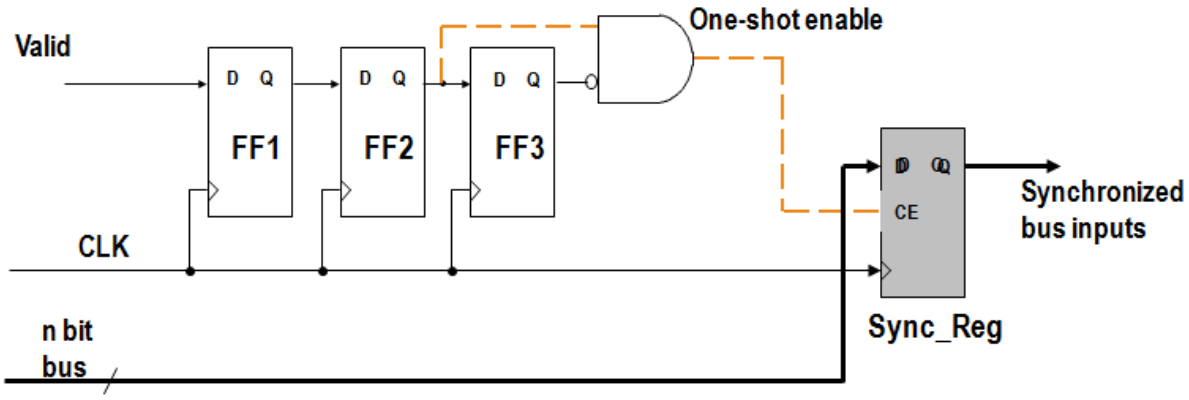
© Copyright 2019 Xilinx

Slide 25-14:

Capturing a Bus When Stable (2)

108642**slide

- If the bus is accompanied by a valid signal, then use it instead
  - Valid is asserted on the first clock where the bus changes
  - Valid comes directly from a flip-flop (is glitchless)
  - Valid is longer than one destination clock pulse
  - Valid does not happen more than once every four destination clocks



108642!*note

The bus must become stable at the same time that the valid is asserted and must remain valid for more than four destination clock cycles.

The valid signal must be asserted for at least two destination clock cycles and deasserted for at least two destination clock cycles.

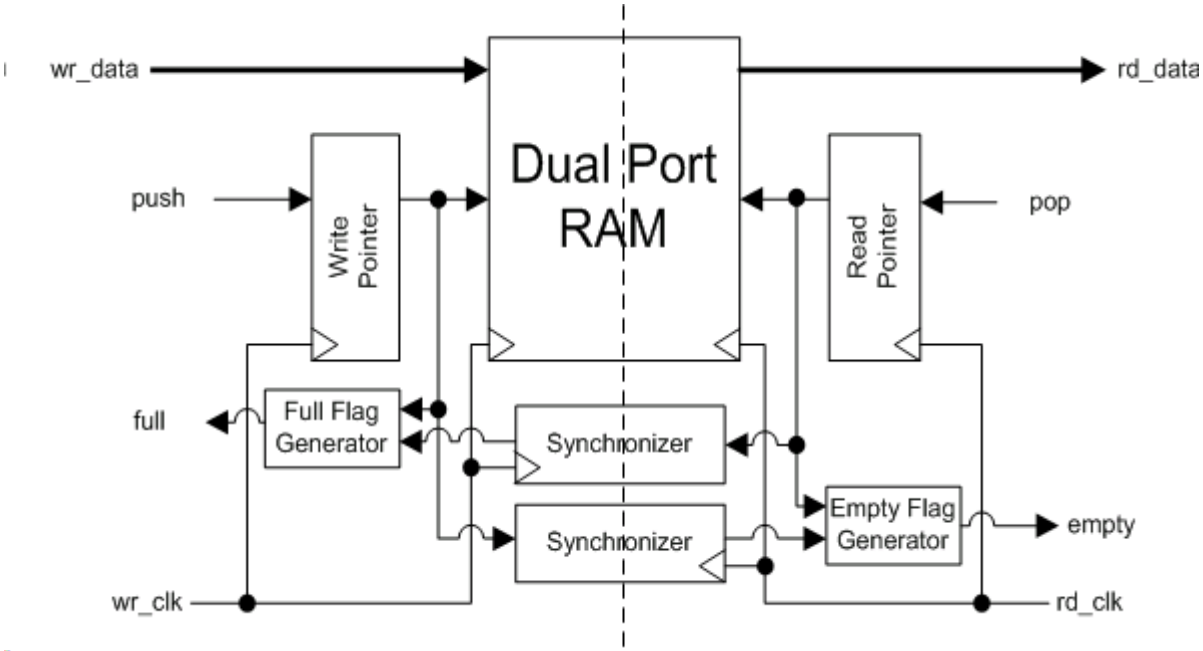
© Copyright 2019 Xilinx

Slide 25-15:

Clock-Crossing FIFO

121174**slide

- FULL and EMPTY flags ensure the data is in the RAM long enough
  - PUSH operation never performed on a full FIFO
  - POP operation never performed on an empty FIFO



121174!*note

A clock-crossing FIFO generally consists of a dual-port RAM for the storage of data and circuitry around the RAM to implement the FIFO functionality.

The RAM can be implemented with a dual-port block RAM if the amount of storage required is larger or can be implemented with distributed RAM for smaller FIFOs.

The circuitry around the RAM consists of read and write pointers; these are address generators that generally traverse the RAM in a specific order (i.e., a counting sequence) and wrap around. The separation between the read and write pointers determine the fill level of the FIFO, and hence if the FIFO is empty or full.

The write pointer address operates on the write clock, determining where to write the next data on a PUSH operation. The read pointer operates on the read clock, determining the read address on a POP operation.

© Copyright 2019 Xilinx

The critical operation in a FIFO is the generation of the FULL and EMPTY flags.

The FULL flag must assert on the clock after the RAM is filled to prevent overflow. Therefore, it needs to operate on the write clock. However, the FULL condition can only be determined by knowing the value of both the read and write pointers. Therefore, the read pointer must be synchronized to the write clock domain.

Similarly, the EMPTY flag must be asserted on the clock after the last data is read from the RAM in order to prevent underflow, and hence needs to be on the read clock. Because it needs the write pointer to determine if the FIFO is empty, the write pointer must be synchronized to the read clock domain.

There are many possible implementations for the pointers and the synchronizers (although Gray coded addresses are often used). The latency through the synchronizers determines the flag deassertion latency; when the FIFO is empty and the first PUSH operation is performed, it may take several clocks for the EMPTY flag to deassert. During this time, however, additional PUSH operations can be performed; the data will start to accumulate in the FIFO. The same is true of the deassertion of the FULL flag when a POP operation is performed.

Slide 25-16:

Analyzing the CDC Report

100319**slide

- The `report_cdc` command reports the clock domain crossing (CDC) paths in the design
  - Available on a synthesized or implemented design
  - Analyzes and reports paths where both source and destination clocks have been defined

- Severity of the reported CDC path can be any of following depending on the CDC topology that is identified
  - Critical
    - An unknown synchronization topology is critical and needs to be reviewed
  - Warning
    - A double register synchronizer with a missing ASYNC_REG property is a warning
  - Info

© Copyright 2019 Xilinx

- A synchronizer with a SYNC_REG property detected has an info message

100319!*note

This report shows in detail the clock domain crossing (CDC) paths in the current synthesized or implemented design. The command analyzes paths between asynchronous clocks or clocks with no common period, as well as synchronous paths ignored by the user due to false path or max delay datapath only exceptions.

By default the `report_cdc` command reports domain crossing among all clocks in the design. However, you can limit the clocks of interest by using the `-from` and `-to` options to specify the clock domains of interest.

The terminology for safe, unsafe, and endpoints is different in the context of Cross Domain Crossing (CDC) and inter-clock timing analysis.

In the context of CDC, an asynchronous crossing is safe when proper synchronization circuitry is used to prevent metastability. For example, a safe single-bit CDC can be implemented by a synchronizer, which is a chain of registers with same clock and control signals.

Conversely, an unsafe CDC is when the CDC analysis engine does not recognize a known safe synchronization circuit on an asynchronous CDC path. The number of endpoints reported for CDC between two clock domains can be different than the number of endpoints reported by the timing analysis commands. For example, an asynchronous reset synchronizer involves multiple timing path endpoints. However, the synchronization circuitry is reported as a single element and therefore as a single CDC endpoint.

Slide 25-17:

Analyzing the MTBF Report

100320**slide

- MTBF stands for mean time between failure
  - Comments on the reliability of a system

© Copyright 2019 Xilinx

- `report_synchronizer_mtbf` command computes the mean time between failures and displays the report
  - Supported for UltraScale™ architecture devices only and does not support 7 series devices

- Reports MTBF for
  - Overall design
  - Each synchronizer chain in the design

`100320!*note`

The `report_synchronizer_mtbf` command reports the mean time between failures (MTBF) of each clock domain crossing (CDC) synchronizer chain in a design and provides an overall MTBF covering all synchronizers.

The synchronizer registers must have ASYNC_REG properties with the value TRUE to be properly identified as synchronizers for reporting.

© Copyright 2019 Xilinx

Slide 25-18:

Summary

108659**slide

- Most systems must interact with asynchronous events and/or data
  - Events from asynchronous sources
  - Events generated by asynchronous clocks
- These events must use synchronization circuits to reduce the mean time between failures (MTBF) of systems due to metastability and bus coherency
- Setup and hold time violations will put the flip-flop into metastable state
- Different synchronization circuits can be used for different types of events
- All of these synchronization circuits require appropriate constraints for proper functionality

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

AXI: Transactions

Slide 26-1:

```
111740**slide

2017.3
111740!*note
```

This module describes the different types of AXI transactions.

After completing this module, you will be able to:

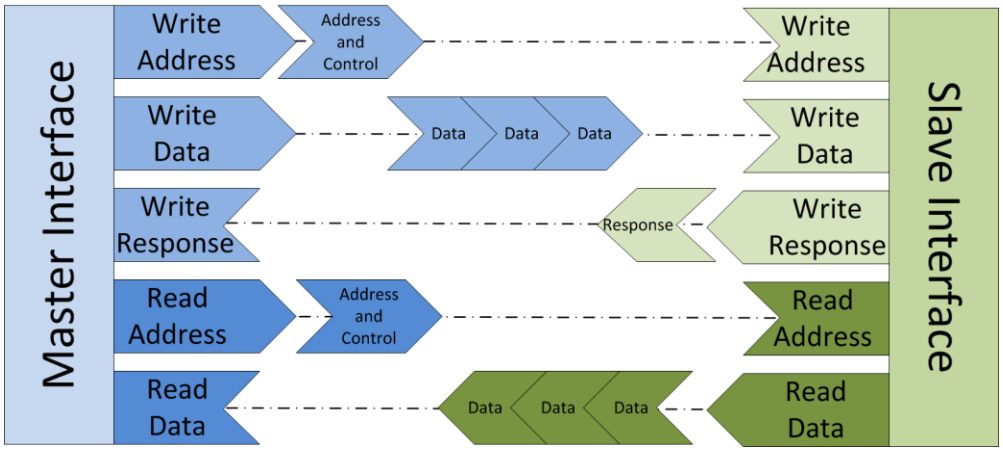
- Identify the different types of AXI transactions

Slide 26-2:

Memory-Map AXI Channels

```
111741**slide
```

- Read address channel
- Read data channel
- Write address channel



- Write data channel
- Write response channel
  - Non-posted write model: there will always be a *write response*

© Copyright 2019 Xilinx

- Other signals are present to indicate when the information on the channel is valid, etc.

111741!*note

© Copyright 2019 Xilinx

The AXI interface has separate and independent read and write channels that can be used simultaneously.

Each channel has its own address and data buses.

Both channels are non-posted; that is, there is always a response. In the read case, the response is simply the read data coming back. For a write, a separate response bus acknowledges data delivery.

Summary of the five AXI channels:

- Read address channel
- Read data channel
- Write address channel
- Write data channel
- Write response channel

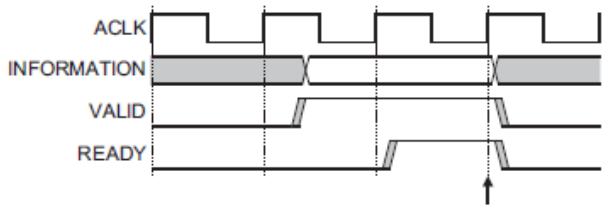
Slide 26-3:

AXI Interface: Handshaking

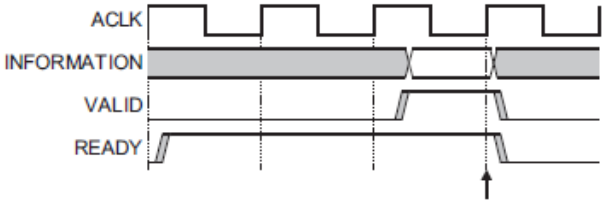
111742**slide

© Copyright 2019 Xilinx

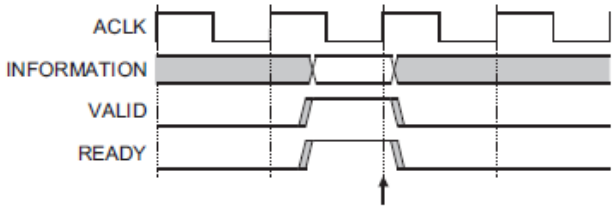
- AXI uses a valid/ready handshake acknowledge
- Each channel has its own valid/ready
  - Address (read/write)
  - Data (read/write)
  - Response (write only)
- Flexible signaling functionality
  - Inserting wait states
  - Always ready
  - Same cycle acknowledge



Inserting Wait States



Always Ready



Same Cycle Acknowledge

111742!*note

It is up to the master to assert the valid signal and the slave to assert the ready signals for all channels except the read data channel, where the slave asserts valid to indicate that it is returning data.

The agent that asserts ready determines the flexibility as seen in the three waveform options.

This ready/valid acknowledge protocol provides masters and slaves the flexibility to pull wait states to prevent data overrun and starvation situations.

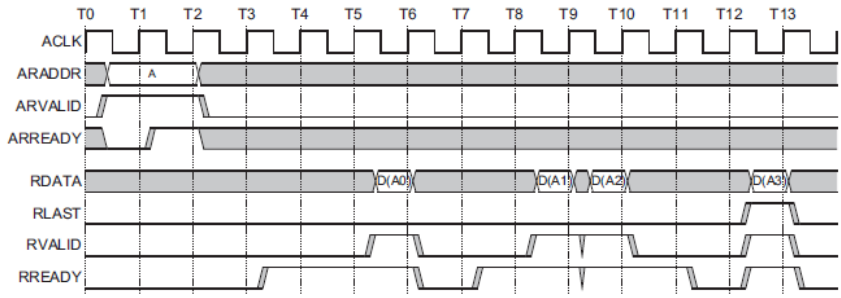
Slide 26-4:

AXI Interface: Memory-Map Read

111743**slide

© Copyright 2019 Xilinx

- Two channels
  - Address
  - Data
- Up to 256 transfer data phase
- Selectable data transfer size
- See notes for signal detail of each channel



AXI – Burst Read

111743!*note

The AXI4 read interface allows for data phase transfer up to 256 beats as opposed to 16 beats that were supported for AXI3. AXI-Lite has identical timing except that there is no support for data bursts - that is, there is only one beat.

Some signal features may not be required for all types of transfers, depending on user requirements and capabilities.

Read Address Channel

- **ARID[3:0]** – Master read address ID. This signal is the identification tag for the read address group of signals.
- **ARADDR[31:0]** – Master read address. The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
- **ARLEN[7:0]** – Master burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
- **ARSIZE[2:0]** – Master burst size. This signal indicates the size of each transfer in the burst.
- **ARBURST[1:0]** – Master burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.

© Copyright 2019 Xilinx

- **ARLOCK[1:0]** – Master lock type. This signal provides additional information about the atomic characteristics of the transfer.
- **ARCACHE[3:0]** – Master cache type. This signal provides additional information about the cacheable characteristics of the transfer.
- **ARPROT[2:0]** – Master protection type. This signal provides protection unit information for the transaction.
- **ARVALID** – Master read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. 1 = address and control information valid. 0 = address and control information not valid.
- **ARREADY** – Slave read address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready; 0 = slave not ready.

Read Data Channel

- **RID[3:0]** – Slave read ID tag. This signal is the ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
- **RDATA[31:0]** – Slave read data. The read data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide (the PL limits this to a maximum width of 256).
- **RRESP[1:0]** – Slave read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
- **RLAST** – Slave read last. This signal indicates the last transfer in a read burst.
- **RVALID** – Slave read valid. This signal indicates that the required read data is available and the read transfer can complete: 1 = read data available; 0 = read data not available.
- **RREADY** – Master read ready. This signal indicates that the master can accept the read data and response information: 1= master ready; 0 = master not ready.

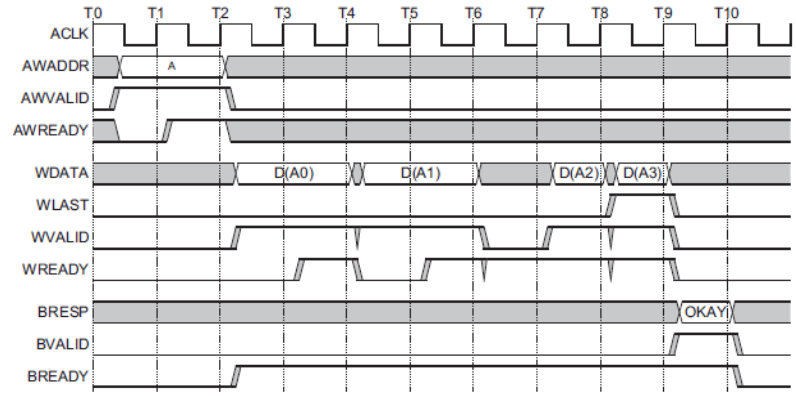
© Copyright 2019 Xilinx

Slide 26-5:

AXI Interface: Memory-Map Write

111744**slide

- Three channels
  - Address
  - Data
  - Response
- Up to 256 transfer data phase
- Selectable data transfer size
- See notes for signal detail of each channel



AXI – Burst Write

111744!*note

The AXI4 write interface allows for data phase transfer up to 256 beats as opposed to 16 beats that were supported for AXI3. AXI-Lite has identical timing except that there is no support for data bursts - that is, there is only one beat.

Some signal features may not be required for all types of transfers, depending on user requirements and capabilities.

Write Address Channel

- **AWID[3:0]** – Master write address ID. This signal is the identification tag for the write address group of signals.
- **AWADDR[31:0]** – Master write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.

© Copyright 2019 Xilinx

- **AWLEN[3:0]** – Master burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
- **AWSIZE[2:0]** – Master burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.
- **AWBURST[1:0]** – Master burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
- **AWLOCK[1:0]** – Master lock type. This signal provides additional information about the atomic characteristics of the transfer.
- **AWCACHE[3:0]** – Master cache type. This signal indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.
- **AWPROT[2:0]** – Master protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
- **AWVALID** – Master write address valid. This signal indicates that valid write address and control information are available: 1 = address and control information available; 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes HIGH.
- **AWREADY** – Slave write address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready; 0 = slave not ready.

Write Data Channel

- **WID[3:0]** – Master write ID tag. This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.
- **WDATA[31:0]** – Master write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide (the PL limits this to a maximum width of 256).
- **WSTRB[3:0]** – Master write strobes. This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)].
- **WLAST** – Master write last. This signal indicates the last transfer in a write burst.
- **WVALID** – Master write valid. This signal indicates that valid write data and strobes are available: 1 = write data and strobes available; 0 = write data and strobes not available.
- **WREADY** – Slave write ready. This signal indicates that the slave can accept the write data: 1 = slave ready; 0 = slave not ready.

© Copyright 2019 Xilinx

Write Response Channel

- **BID[3:0]** – Slave response ID. The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
- **BRESP[1:0]** – Slave write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
- **BVALID** – Slave write response valid. This signal indicates that a valid write response is available: 1 = write response available; 0 = write response not available.
- **BREADY** – Master response ready. This signal indicates that the master can accept the response information. 1 = master ready. 0 = master not ready.

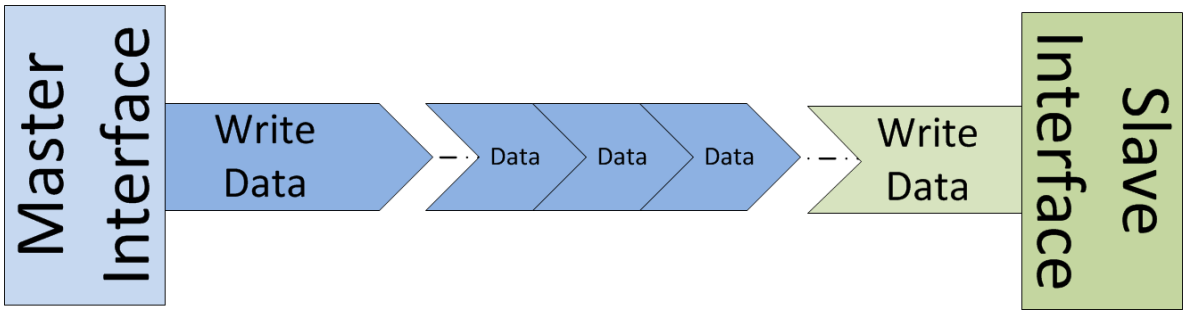
Slide 26-6:

AXI Interface: Streaming

111759**slide

- Data moves in a single direction, from a master to a slave
  - Simplex communication: master-to-slave
  - Dual direction would be dual simplex; individual channels, for example

- Data only
  - No address channel
  - Minimum control
  - Unlimited burst length
  - Sideband signals optionally used to condition data
  - Contents of data stream interpreted by slave



© Copyright 2019 Xilinx

111759!*note

Data flowing across the streaming connection may be in packet format or not. The handshaking signals may provide packet start/end information or the data itself may contain special characters to mark the beginning and end of a packet. The important point here is that both master and slave must have prior knowledge of data format. Just to drive the point home, the use of hardware signaling is optional and may be used if the designer deems it necessary or useful, or it may be completely disregarded.

AXI streaming does not have an address phase; all transactions go to the same place. Also note that the direction is always from master to slave.

AXI streaming is very similar to the MicroBlaze™ processor fast simplex link (FSL) except that there is no requirement that a processor be involved.

© Copyright 2019 Xilinx

Note that the AXI-Stream signals are less numerous than for AXI4 or AXI4-Lite, as only a single channel is required. In many instances, not all of the signals are required for a given application.

Signal	Source	Description
ACLK	Clock source	The global clock signal. All signals are sampled on the rising edge of **ACLK**.
ARESETn	Reset source	The global reset signal. **ARESETn** is active-LOW.
TVALID	Master	**TVALID** indicates that the master is driving a valid transfer. A transfer takes place when both **TVALID** and **TREADY** are asserted.
TREADY	Slave	**TREADY** indicates that the slave can accept a transfer in the current cycle.
TDATA[(8n-1):0]	Master	**TDATA** is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
TSTRB[(n-1):0]	Master	**TSTRB** is the byte qualifier that indicates whether the content of the associated byte of **TDATA** is processed as a data byte or a position byte.
TKEEP[(n-1):0]	Master	**TKEEP** is the byte qualifier that indicates whether the content of the associated byte of **TDATA** is processed as part of the data stream. Associated bytes that have the **TKEEP** byte qualifier deasserted are null bytes and can be removed from the data stream.
TLAST	Master	**TLAST** indicates the boundary of a packet.
TID[(i-1):0]	Master	**TID** is the data stream identifier that indicates different streams of data.
TDEST[(d-1):0]	Master	**TDEST** provides routing information for the data stream.
TUSER[(u-1):0]	Master	**TUSER** is user defined sideband information that can be transmitted alongside the data stream.

The following stream terms are used:

- **Transfer**: A single transfer of data across an AXI4-Stream interface. A single transfer is defined by a single TVALID, TREADY handshake.
- **Packet**: A group of bytes that are transported together across an AXI4-Stream interface. A packet is similar to an AXI4 burst. A packet can consist of a single transfer or multiple transfers. Infrastructure components can use packets to deal more efficiently with a stream in packet-sized groups.
- **Frame**: The highest level of byte grouping in an AXI4-Stream interface. A frame contains an integer number of packets. A frame can be a very large number of bytes; an entire video frame buffer, for example.
- **Data stream**: The transport of data from one source to one destination. A data stream can be:
  - A series of individual byte transfers
  - A series of byte transfers grouped together in packets

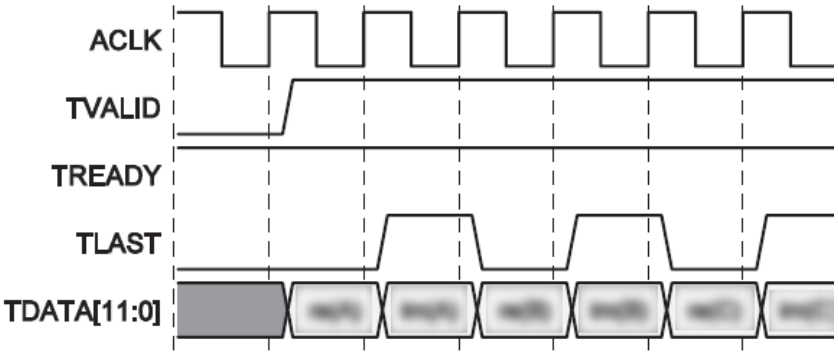
© Copyright 2019 Xilinx

Slide 26-7:

AXI Interface: Streaming Write

111748**slide

- Only write data channel exists
  - Side-band signals are indicated when data is valid and when device is ready



- Other side-band signals may be used for other needs
  - As required by the design

Thi
111748!*note

This example illustrates how TLAST might be used to indicate the type of data being sent. Consider streaming data conveying a frame of information (such as video or audio). How can the device know where the frame begins or ends? Here, TLAST helps the receiving device determine how to interpret the incoming data.

Consider a stream of single-channel sensor data that a device is averaging and thresholding. In such a case, TLAST would not matter, as all the data is treated the same and there is no *start* or *finish* to the packet-less information.

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

JTAG to AXI Master Core

Slide 27-1:

```
109168**slide

2019.1
109168!*note
```

This module discusses how to use a JTAG to AXI debug core to generate AXI transactions for reading and writing the data in AXI peripherals. 5 minutes.

Slide 27-2:

Objectives

```
109169**slide
```

After completing this module, you will be able to:

- Enumerate the benefits of the JTAG to AXI debug core
- Use the JTAG to AXI debug core in your design

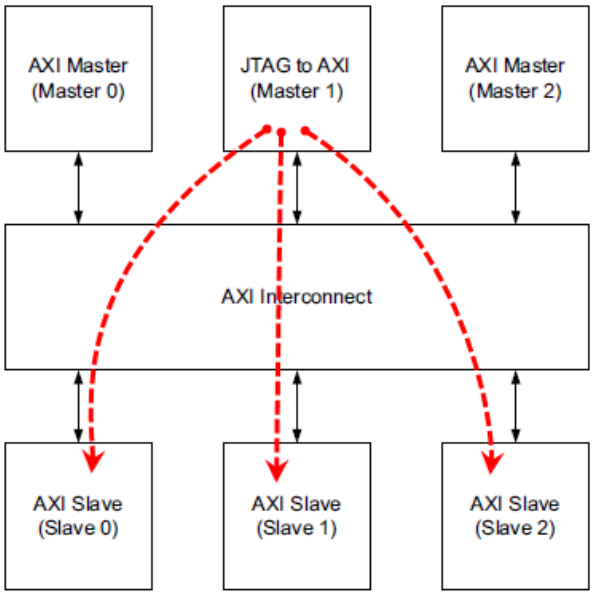
© Copyright 2019 Xilinx

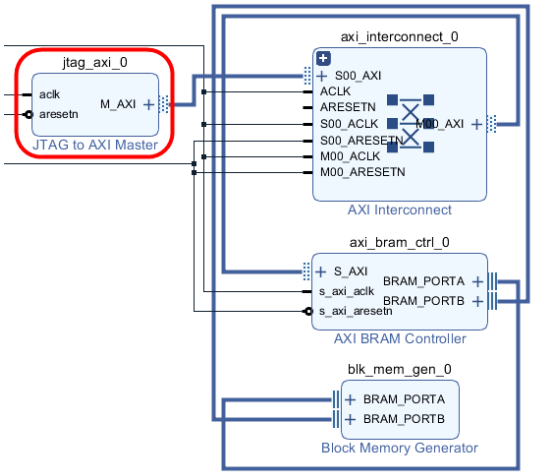
Slide 27-3:

JTAG to AXI Core

121277**slide

- JTAG to AXI Master debug core
  - Customizable IP that works as an AXI Master to drive AXI transactions
  - Used in the Vivado IP integrator
  - Can be instantiated in HDL in a Vivado Design Suite project
- Core does not have its own address space
  - Responds to all the addresses user initiate





121277!*note

© Copyright 2019 Xilinx

The JTAG to AXI Master core can be used in embedded and non-embedded systems where AXI-based IP or the system needs to be debugged. It can also be used during testing to drive the AXI transaction as test vectors on hardware and can serve as an AXI traffic generator. It performs quick read and write transactions during runtime.

The ILA IP core can be used to monitor the traffic on the AXI port of the JTAG to AXI Master core.

© Copyright 2019 Xilinx

Slide 27-4:

JTAG to AXI Core Features

129107**slide

- Parameterized protocol choice
  - AXI4 and AXI4-Lite
- Address width of 32
- ID width (up to four bits) that allows user-defined ID signals
- Fixed AXI4-Lite data width of 32
- Parameterized AXI4 data width of 32 and 64
- Support for all memory-mapped AXI interface transactions
  - Burst type – INCR, FIXED, and WRAP
  - Burst length
    - 1 to 256 for INCR and FIXED
    - 2, 4, 8, and 16 for WRAP
- Support for cache signals
- Hardware debug run time Tcl Console control to
  - Enable and disable AXI read/write
  - Simultaneous read/write

Slide 27-5:

Communication with the Core

129108**slide

- The JTAG to AXI Master debug core can only be communicated with using Tcl commands
- Reset the debug core
  - `reset_hw_axi`
- Create an AXI transaction
  - `create_hw_axi_txn`
- Run the AXI transaction
  - `run_hw_axi`
- Report the result of transaction
  - `report_hw_axi_txn`

© Copyright 2019 Xilinx

129108!*note

The `report_property` command can also be used to get the results of a transaction.

In the *JTAG to AXI Master Core* demo of this topic cluster, you will be introduced to the use of Tcl commands for runtime interaction with the JTAG to AXI debug core.

© Copyright 2019 Xilinx

Capture Your Notes Here

© Copyright 2019 Xilinx

JTAG to AXI Master Core Demo Script

Introduction

This demonstration introduces the JTAG to AXI debug core that can generate AXI transactions and drive internal AXI signals in a design at run time. The demonstration also highlights the Tcl commands used at runtime interaction with this core in the Vivado® hardware manager environment.

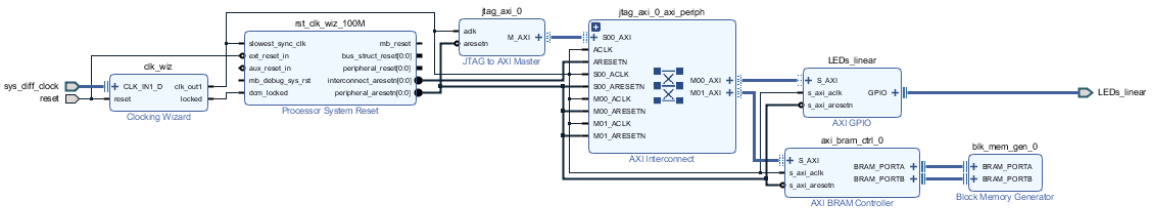
Preparation:

- Necessary files are located in the following directory:
  `$jtag2axi/demo/KC705`
- Required hardware: KC705 Evaluation Platform
- Required software: Vivado Design Suite 2019.1 (any edition)

JTAG to AXI Master Debug IP Transactions

Action with Description	Point of Emphasis and Key Takeaway
• Launch Vivado Design Suite 2019.1.	
• Open the project **`jtag2axi.xpr`** from the following directory using the Vivado IDE: `/home/xilinx/training/jtag2axi/demo/KC705`	• The Open Project selection allows the designer to access existing projects.
• Notice that the design is empty. • You will create an IPI block design that has two peripherals: block RAM and GPIO (which connects to the LEDs on the board).	There are several ways to create a block design: • One way is to use a Tcl script to generate the required block design.

© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
• Enter the following command in the Tcl Console of the Vivado IDE to generate the required block design for this demonstration: `source /home/xilinx/training/jtag2axi/demo/KC705/Create_jtag2axi_block-design.tcl` • Right-click and select **Regenerate Layout**.	• The Tcl script is used for the interest of time in this demonstration. • Once the Tcl script finishes, you should be able to see the IPI block design as shown below.
	

• View the **Address Editor** tab.
• Note that the Vivado Design Suite assigns addresses to the PL slave peripherals:
  • AXI block RAM controller, which is connected to the dual-port BRAM block
  • AXI GPIO-LEDs_linear, which drives the board LEDs
	• The Address Editor tab shows the base and high address along with the size of each peripheral. The base address of the block RAM controller is 0xC0000000 and the AXI GPIO is 0x40000000.

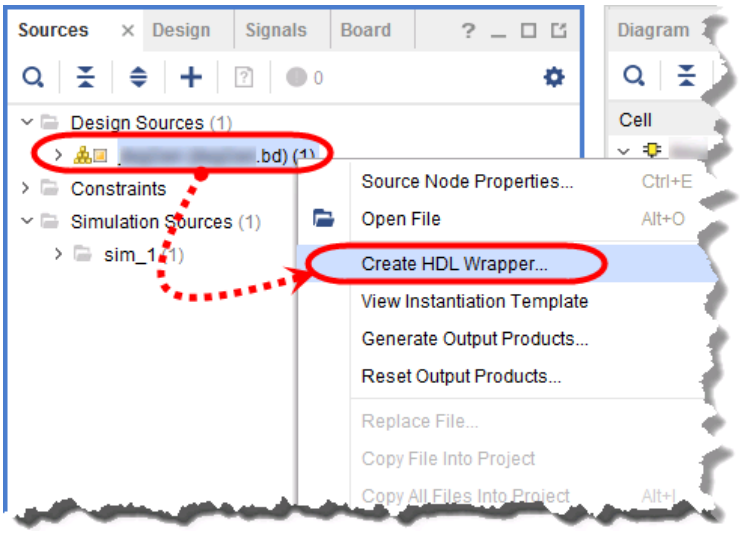
© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
• View the IPI block design and quickly summarize the position of the JTAG to AXI core.	• The JTAG to AXI master core is an AXI-based embedded system that connects to the slave interface of the AXI interconnect that has slave peripherals. • The JTAG to AXI master core is placed before the AXI interconnect as a master to make data transactions to block RAM and drive the LEDs on the board.

• Validate and save the updated design.
  • Select **Tools** > **Validate Design** to scan for any errors and critical warnings. Resolve any issues.
  • Select **File** > **Save Block Design**.
	• The updated IPI block design needs to be validated for any errors before you save it.

• Create a HDL top-level wrapper for the block design by letting the Vivado Design Suite manage the wrapper and auto-update.	• The HDL top-level wrapper instantiates the final IPI block design with port mapping. • A top-level source file is required; it can be generated by the Vivado Design Suite.

© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
	
• Now that the HDL wrapper is generated, it is time to implement the design and generate the bitstream. • However, in the interest of time, the bitstream has already been generated and is available in the project directory.	

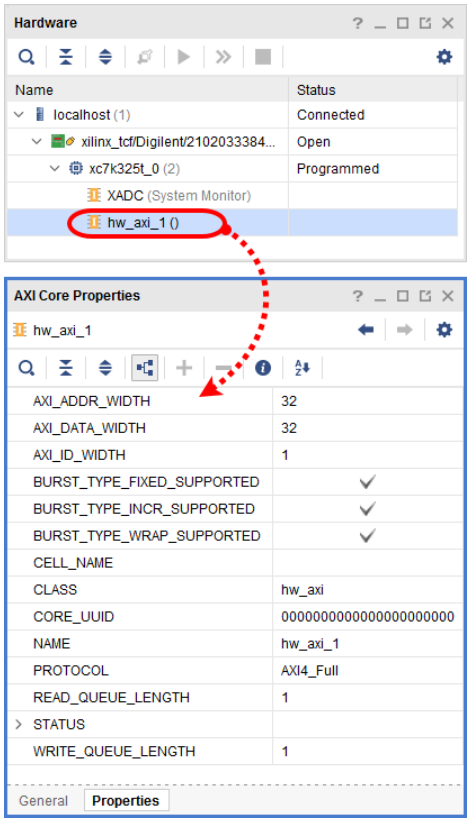
• Set up and connect the board, or verify that this has properly been done before turning on the power.
  • All the connections are made to the board itself.
  • Power on the board.
	• Powering up the board allows you to launch the Vivado hardware manager.

© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
• Click **Program and Debug** > **Open Hardware Manager** from the Flow Navigator to establish a connection to the board. • Click **Open target** > **Auto Connect** to connect to the target board automatically with the default settings.	• A hardware session is the utility of the Vivado Design Suite that enables the monitoring of debug cores that were added to a design.
Now that connection to the board is established, your first task is to typically download a bitstream to your board. • Right-click the **xc7k325t_0** device and select **Program Device**. • Select the bitstream (**jtag2axi_wrapper**.**bit**) and probe (**debug_nets**.**ltx**) files from the project directory.	• A bitstream programming file is used to download to your hardware device, whereas debug probe files contain details of the probing signals for cores like VIO and ILA. • You can ignore the *debug.ltx* probe file in this case as you are not monitoring any internal design signals. • **Note:** The JTAG to AXI master core that you added to the design appears in the Hardware window under the target device. If you do not see the JTAG to AXI core appear, right-click the device and select **Refresh Device**. This re-scans the FPGA device and refreshes the Hardware window.
The hardware is now programmed and you can see the AXI debug core in the device tree. • Select **hw_axi_1(AXI)** and explore the AXI core properties.	• The core is AXI4 Full type, which is a master to the AXI4 Lite peripherals on the other side of the interconnect.

© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
• Review the properties of JTAG to AXI master core in the AXI Core Properties window.	• The JTAG to AXI master debug core is a customizable core that can generate the AXI transactions and drive the AXI signals internal to an FPGA at run time. • The core supports all memory-mapped AXI and AXI-Lite interfaces and can support a 32-bit or 64-bit wide data interface. • The JTAG to AXI master core can only be communicated with using Tcl Console commands. • You can create and run AXI read and write transactions using these Tcl Console commands.



© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway

• Reset the JTAG to AXI master debug core.
  • Before creating and issuing transactions, it is important to reset the JTAG to AXI master core using the following Tcl command:
  `reset_hw_axi [get_hw_axis hw_axi_1]`
	• The AXI side of the logic is reset when the JTAG-to-AXI master core samples this as low on the rising edge of the input clock. You need to use the `reset_hw_axi` Tcl command to reset the core.

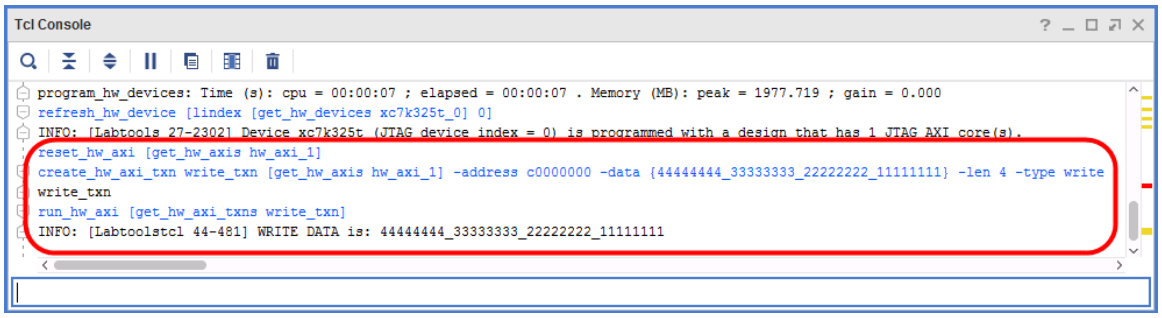
© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway

• Create a write transaction into the block memory in the design.
  • The following Tcl command creates a 4-word AXI write burst transaction to the memory controller base address:
  `create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -address c0000000 -data {44444444_33333333_22222222_11111111} -len 4 -type write`
  where:
  ▪ *write_txn* is the user-defined name of the transaction.
  ▪ *[get_hw_axis hw_axi_1]* returns the hw_axi_1 object.
  ▪ *-address c0000000* is the base address of bram controller.
  ▪ *-data {44444444_33333333_22222222_11111111}*
  The -data direction is LSB to the left (i.e., address 0) and MSB to the right (i.e., address 3).
  ▪ *-len 4* sets the AXI burst length to four words.
	• The 4-word AXI burst transaction writes to the locations of the block RAM starting from the base address.

© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
• Run the write transaction that was just created by using the `run_hw_axi` command: `run_hw_axi [get_hw_axi_txns write_txn]`	• After creating the transaction in the above step, you ran it as a write transaction by using the `run_hw_axi` command. • The data *44444444_33333333_22222222_11111111* is now written to the memory.



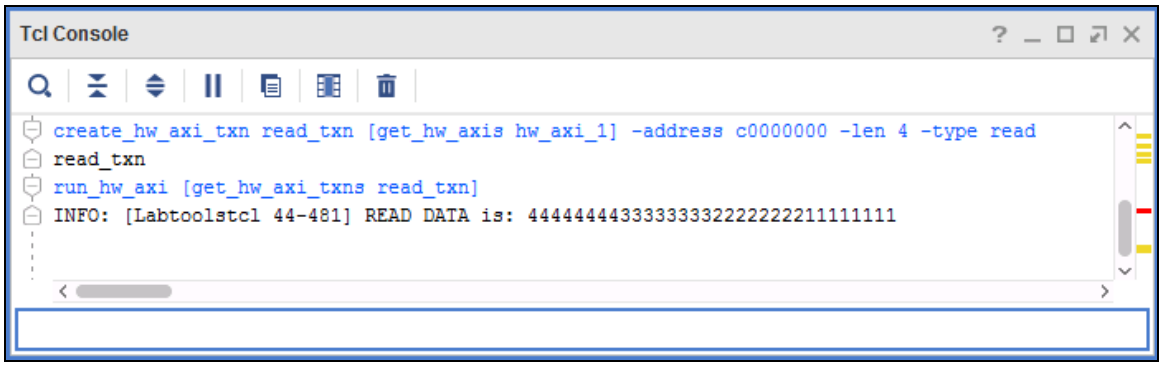
© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway

• Create a read transaction to the block memory in the design.
  • The following Tcl command creates a 4-word AXI read burst transaction from the memory controller base address that was written in the previous steps:
  `create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -address c0000000 -len 4 -type read`
  where:
  ▪ *read_txn* is the user-defined name of the transaction.
  ▪ *[get_hw_axis hw_axi_1]* returns the hw_axi_1 object.
  ▪ *-address c0000000* is the base address of bram controller.
  ▪ *-len 4* sets the AXI burst length to fourwords.
	• The 4-word AXI burst transaction reads from the locations of the block RAM starting from the base address.

• Run the read transaction that was just created by using the `run_hw_axi` command: `run_hw_axi [get_hw_axi_txns read_txn]`	• After creating the transaction in the above step, you ran it as a read transaction by using the `run_hw_axi` command. • The data *44444444333333332222222211111111* is now read back from the memory.

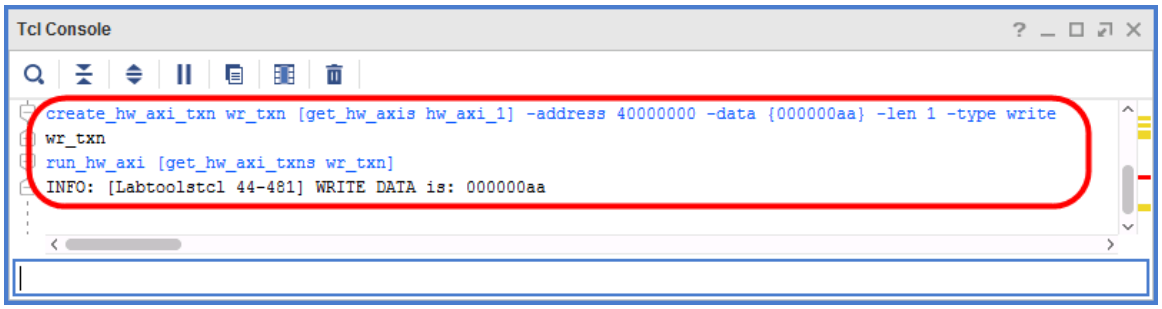
© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
	

• Create a write data transaction on the board user 8-bit LEDs that are enabled in the design.
  • The following Tcl command creates an 1-word AXI write transaction to the AXI GPIO-LEDs base address:
  `create_hw_axi_txn wr_txn [get_hw_axis hw_axi_1] -address 40000000 -data {000000aa} -len 1 -type write`
	• The 1-word AXI data transaction writes to the base address of the user LEDs.

• Run the read transaction on user LEDs that was just created by using the `run_hw_axi` command: `run_hw_axi [get_hw_axi_txns wr_txn]`	• After creating the transaction in the above step, you ran it as a write transaction by using the `run_hw_axi` command. • The written data "000000aa" will be displayed on the board user LEDs.

© Copyright 2019 Xilinx

Action with Description	Point of Emphasis and Key Takeaway
	
• Close the hardware manager and exit the Vivado Design Suite.	• Exit the GUI and applications.
• Power off the evaluation board.	

Summary

This demonstration showed how to add and integrate the JTAG to AXI debug IP core available to an embedded or non-embedded system and how to perform essential connections to the AXI peripherals in the IPI block design.

Launching the hardware server to open, set up a JTAG connection, and program the device was also shown. The use of the Vivado logic analyzer Tcl Console interface for run-time interaction with the target hardware was demonstrated as well. Note that the JTAG to AXI master debug core can only be communicated via Tcl commands.

© Copyright 2019 Xilinx

© Copyright 2019 Xilinx