Large FPGA Methodology Guide

Including Stacked Silicon Interconnect (SSI) Technology

UG872 (v14.3) October 16, 2012

This document applies to the following software versions: Vivado Design Suite 2012.2 through 2013.3 and ISE Design Suite 14.3 through 14.7

XILINX®

Revision History

Date	Version	Revision
10/16/2012	14.3	• Updated device-specific information
04/24/2012	14.1	• Updated *Interconnects between SLRs* in Table 3-1, Key Resources Available in Each Virtex-7 SLR Type. • Updated *SLL Components* in Table 3-2, SLL Components for Each SLR Crossing. • Updated Figure 3-5, Staggered SLLs Crossing in an SSI Device. • Updated Figure 3-6, Representation of SLL Connectivity in the SLR.

Table of Contents

Appendix A: Additional Resources

Chapter 1

Introduction

This guide addresses designs targeting large FPGA devices. This guide includes, but is not limited to, designs using Stacked Silicon Interconnect (SSI) technology.

Design Strategies

This guide details strategies for:

- System level planning
- Design creation
- Implementation
- Analysis

As discussed in Chapter 2, Large FPGA Device Methodology, these strategies can help you achieve optimal results from your large FPGA devices with respect to:

- Routing Utilization
- Design Performance
- Power Consumption
- Project Costs

Large FPGA Devices

The term *large FPGA device* is an ever-changing expression. As used in this guide, *large FPGA device* means the larger devices in the Xilinx® Virtex®-6 and Virtex-7 device families.

As illustrated in the following figure, device capacity increases significantly with each new FPGA device family.

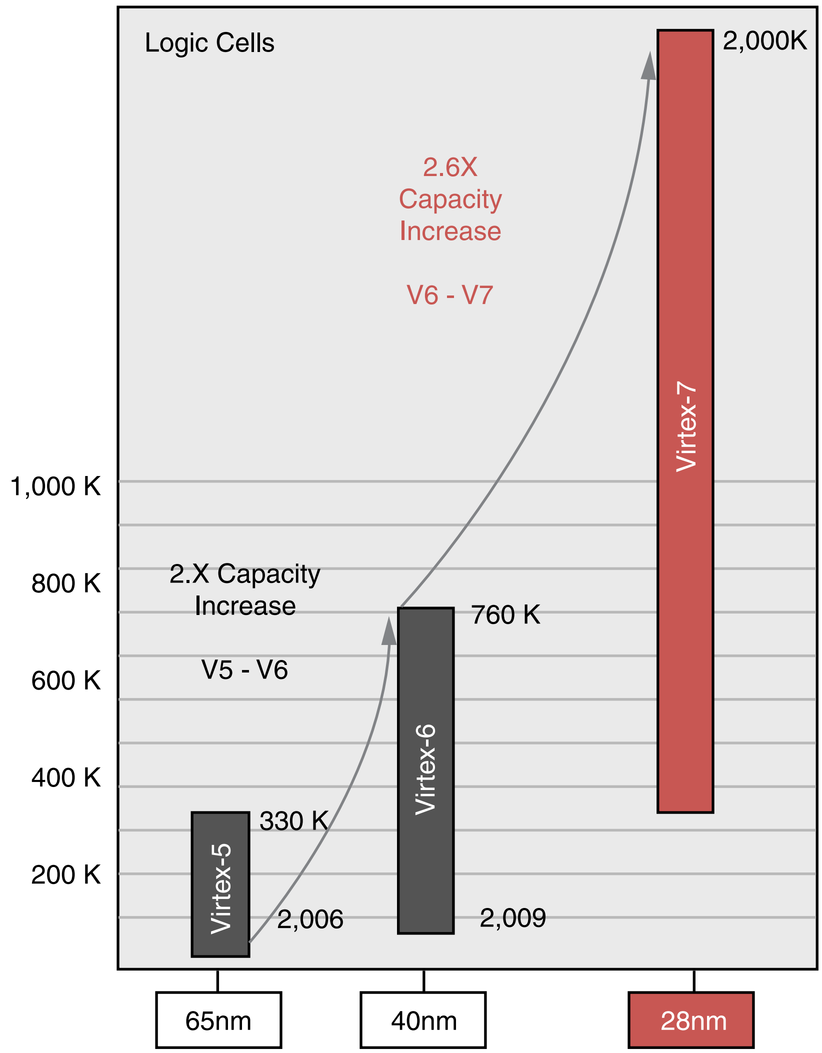

Figure 1-1: **FPGA capacity has increased over 6x in less than 5 years**

Today's largest FPGA devices typically contain:

- Over 1 million 6-input LUTs
- Over 2 million registers
- Thousands of block RAM components and DSP blocks
- Over 1000 general purpose I/O components
- Up to 96 multi-Gigabit Transceivers (GT)
- Many other functions and resources

This leap in capability allows for larger system integration onto fewer chips, or even onto a single chip.

SSI Technology

The Virtex-7 FPGA devices with the highest capacity and capability are created using a manufacturing process known as Stacked Silicon Interconnect (SSI) technology.

When targeting Virtex-7 devices that use SSI, many of the same tools, techniques, and methods apply that are used in any large FPGA design. However, because of the specifics of the Virtex-7 architecture, some additional considerations are required.

For more information, see Chapter 3, Stacked Silicon Interconnect (SSI).

Chapter 2

Large FPGA Device Methodology

The synthesis and implementation tools must make optimal use of the fixed resources of the FPGA device.

In order to accomplish this goal, this guide presents a large FPGA device methodology that includes specific:

- Coding styles
- Implementation methods
- Design techniques

Benefits

Because of exponential FPGA device growth, many traditional coding styles and implementation methods are no longer adequate to achieve your design goals, including utilization, performance, and power.

The Xilinx® large FPGA device methodology allows your design to achieve optimal device and design characteristics, such as:

- Routing Utilization
- Design Performance
- Power Consumption

This methodology also allows you to achieve efficiencies in:

- Software runtime
- Debugging capability
- Portability

Inefficiencies in the code or implementation can seriously hinder achieving your design goals. While this is true for any size design or device, it is especially true of designs targeting large devices.

Most of the topics discussed in this guide are not new, nor are they unique to large FPGA devices. However, by applying these methods to your next large FPGA design, you will be more likely to meet or even surpass your design goals.

Routing Utilization

Routing in FPGA devices is a fixed and finite resource.

Mismanagement of routing resources can negatively impact FPGA design characteristics, such as:

- Resource utilization
- The ability to meet performance goals
- The ability to meet or reduce power

Use of routing is directly correlated to:

- System level design choices
- Design entry
- Coding styles
- Implementation and debugging methods

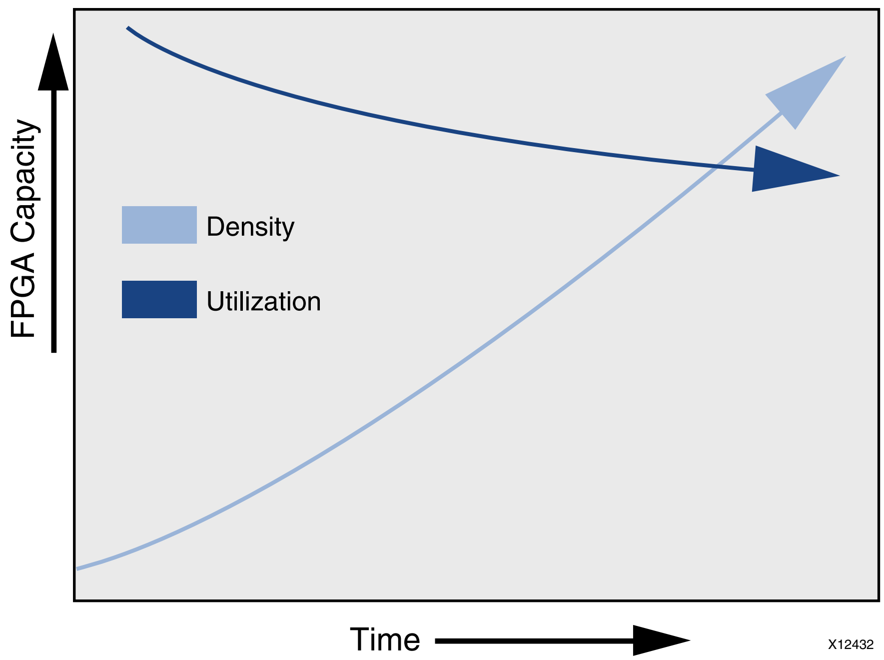

Figure 2-1: **As density increases, more effort is required to achieve high utilization in large FPGA devices.**

Consequences of Inefficient Use of Routing Resources

Choices leading to inefficient use of routing resources can cause routing congestion and limit routing choices for the tools. This can result in:

- Additional routing capacitance (which increases power)
- Additional delay (which affects performance)
- Inability (in the worst cases) to completely route the design in the device

In larger devices, large arrays of logic tend to:

- Push related logic functions further apart.
- Grow the fanouts of some signals to staggering numbers.
- Require significantly more routing resources and routing density to realize a completed design.

Once the routing channels are exhausted, resource utilization must diminish.

Improving Routing Utilization

The techniques discussed in this guide can:

- Improve routing utilization.
- Lead to higher usable logic capacity.
- Reduce the need for some FPGA resources.
- Allow for applications to consume less of the device.
- Give more room for future growth.
- Allow transitioning to a smaller device, thus improving cost, power, and other design factors.

Design Performance

Unmanaged design performance can lead to:

- More difficult timing closure
- Longer runtimes
- More iterations
- Diminished specifications for design performance

Larger designs are often forced into non-optimal placement because of many factors, including:

- High number of I/O connections
- Very wide data buses
- Large fanout signals
- Too many logic levels

Non-optimal placement leads to more routing resource usage. This leads to longer routes and diminished performance.

Because of the scope and size of large designs, countless timing paths must be analyzed and closed. Performance management in large designs is even more essential than in smaller designs.

Power Consumption

Power consumption has been significantly reduced in newer Xilinx FPGA devices.

Without such a reduction, design power can grow disproportionately to design size. This requires special efforts to mitigate the heat dissipated inside the chip, as well as the required supply power to the chip.

Even with careful attention to power at a *system* level, power can grow to undesirable levels at the *device* level if left unchecked.

Many of the recommendations in this guide can help reduce power consumption for a specific design or function.

For more information on power analysis and power reduction techniques, see the *Power Methodology Guide (UG786)*, cited in Appendix A, Additional Resources.

Project Costs

Large FPGA designs consume large amounts of resources. Resource utilization can increase at an alarming rate when measured with respect to fixed resources such as:

- Look Up Tables (LUT)
- Flip flops (FF)
- Clocking resources
- I/O components
- RAM components
- DSP components

This resource consumption can force designs onto the largest FPGA devices, or even onto multiple large FPGA devices.

Inefficiencies in FPGA resource management can make large designs even larger, driving up the cost of the project. For example, a ten percent inefficiency of resources in a large design might constitute a much larger cost than a ten percent inefficiency in a smaller design.

This additional size and complexity can drive up expenses, measured by not only device cost, but also board costs and additional costs to the project schedule.

Stacked Silicon Interconnect (SSI)

This guide addresses all designs targeting large FPGA devices. This chapter discusses designs specifically using the Stacked Silicon Interconnect (SSI) technology.

The SSI technology combines multiple Super Logic Region (SLR) components mounted on a passive Silicon Interposer.

Compared to traditional devices, SSI technology enables Xilinx to construct FPGA devices with the following characteristics:

- The devices are much larger.
- The devices have more dedicated features.
- The devices have a lower power envelope.

Note: The terms *traditional device* and *monolithic device* refer to devices not using SSI technology.

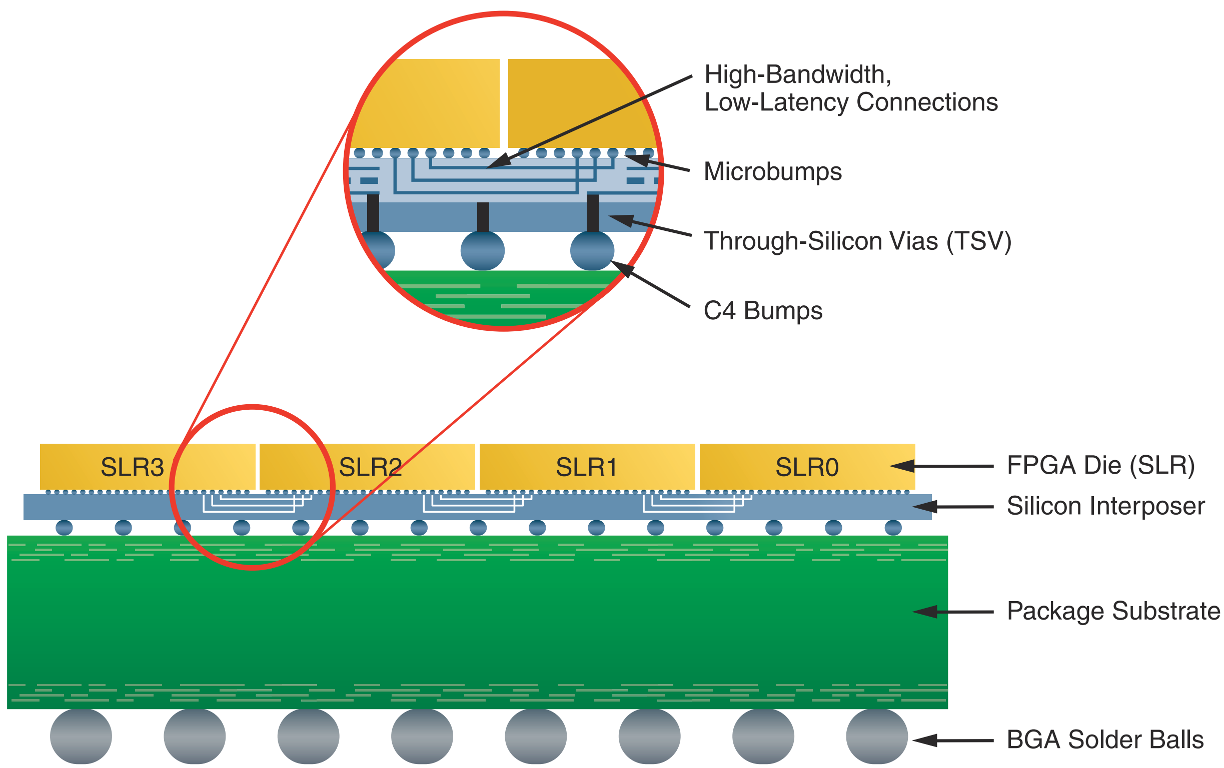

Figure 3-1: **Representative SSI Device Construction**

SSI Components

This section discusses Stacked Silicon Interconnect (SSI) components, and includes:

- Super Logic Region (SLR)
- Silicon Interposer
- Super Long Line (SLL) Routes
- Master Super Logic Region (SLR)

Super Logic Region (SLR)

A Super Logic Region (SLR) is a single FPGA die slice contained in an SSI device.

Active Circuitry

Each SLR contains the active circuitry common to most Xilinx FPGA devices. This circuitry includes large numbers of:

- 6-input LUTs
- Registers
- I/O components
- Gigabit Transceivers (GT)
- Block memory
- DSP blocks
- Other blocks

SLR Components

Multiple SLR components are assembled to make up an SSI device.

The general aspect ratio of an SLR is wider than it is tall. The orientation of the SLR components is stacked vertically onto the interposer.

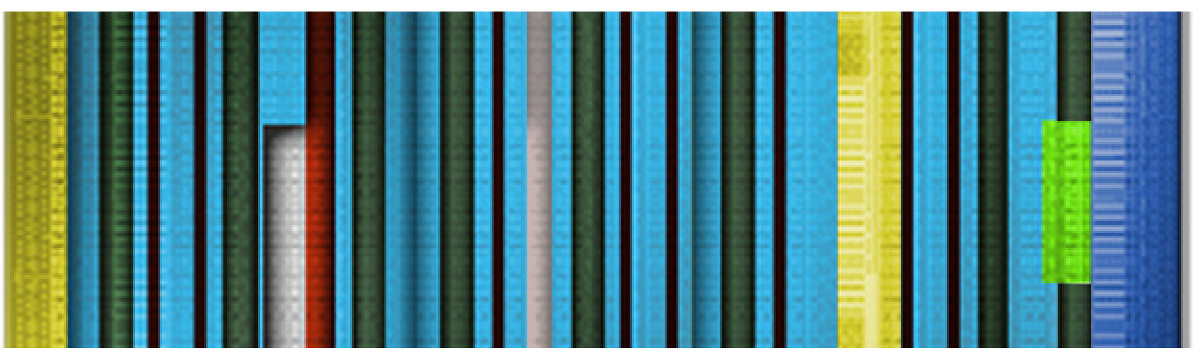

Figure 3-2: **Single SSI SLR**

Multiple SLR components are stacked vertically to create the SSI devices.

- The bottom SLR is **SLR0**.
- Subsequent SLR components are incremented as they ascend vertically.

For example, there are four SLR components in the XC7V2000T device.

- The bottom SLR is **SLR0**.
- The SLR directly above **SLR0** is **SLR1**.
- The SLR directly above **SLR1** is **SLR2**.
- The top SLR is **SLR 3**.

The Xilinx tools (including the PlanAhead™ design analysis tool) clearly identify SLR components in the graphical user interface (GUI) and in reports.

SLR Nomenclature

Understanding SLR nomenclature for your target device is important in:

- Pin selection
- Floorplanning
- Analyzing timing and other reports
- Identifying where logic exists and where that logic is sourced or destined

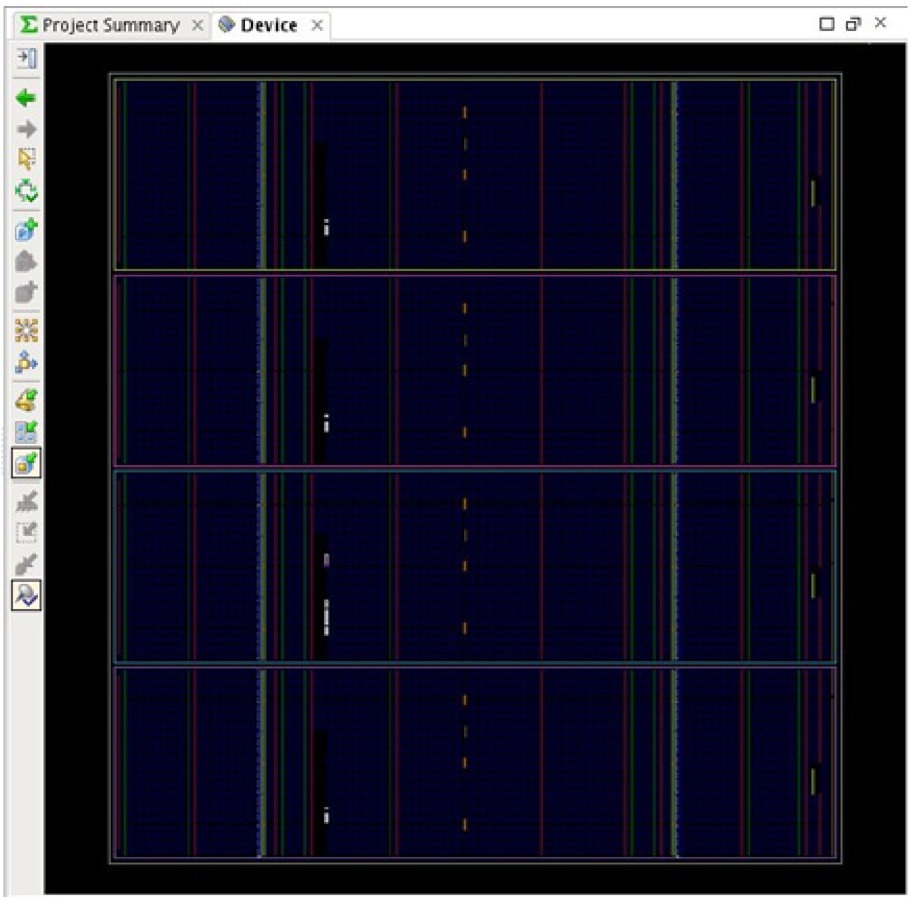

Figure 3-3: **Vivado Tool Representation of a 2000T Device**

Virtex-7 Device Family SLR Components

Two different SLR components are used to create the Virtex®-7 device family:

- xc7v2000t Devices
- xc7vx1140t and Virtex-7 HT Device Family

xc7v2000t Devices

The **xc7v2000t** devices share the same type of SLR containing:

- Approximately 500, 000 logic cells
- A mix of the following components:
  - I/O
  - Block RAM
  - DSP blocks
  - GTX Transceivers
  - Other blocks

xc7vx1140t and Virtex-7 HT Device Family

The **xc7vx1140t** devices and the Virtex-7 HT device family utilize SLR components containing:

- Approximately 290,000 logic cells
- GTH Transceivers
- A larger number of block RAM and DSP components than the **xc7v2000t** SLR components

Table 3-1: **Key Resources Available in Each Virtex-7 SLR Type**

	Virtex-7 T SLR	**Virtex-7 XT/HT SLR**
Logic Cells	488,640	284,800
Slices	76,350	44,500
Block RAM	323	470
DSP Slices	540	840
Clock Regions/MMCM	6	6
I/O	300	300
Transceivers	12	24
Interconnects between SLRs	12,864	10,560

Silicon Interposer

The silicon interposer is a passive layer in the SSI device.

This layer routes the following between SLR components:

- Configuration
- Global clocking
- General interconnect

The silicon interposer provides:

- Power and ground
- Configuration
- Inter-die connectivity
- Other required connectivity

The active circuitry exists on the SLR. The silicon interposer is bonded to the packaging substrate using Through-Silicon Via (TSV) components. These components connect the circuitry of the FPGA device to the package balls.

The silicon interposer is the conduit between SLR components and the packaging substrate. It connects the following to the device package:

- Power and ground connections
- I/O components
- Gigabit Transceivers (GT)

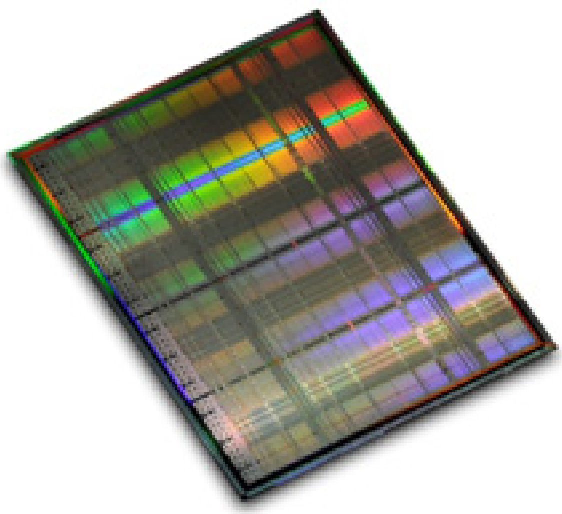

Figure 3-4: **Silicon Interposer**

Super Long Line (SLL) Routes

- Super Long Line (SLL) routes provide the general connectivity for signals that cross from one SLR to another.
- SLL routes are located in the Silicon Interposer.
- SLL routes are connected to the SLR components by microbumps connected directly to the interconnect in the SLR.
- SLL routes connect to the center of Vertical 12 routes in the SLR.

SLL Components in Virtex-7 Devices

In Virtex-7 devices, each SLL component spans the vertical length of 50 interconnect tiles (equivalent to 50 Slice components). This is exactly the height of one clock region in Xilinx 7 series FPGA devices.

Consequently, in SLR adjacent clock regions, there is one interconnect point connecting to the neighboring SLR at every interconnect tile in the clock region.

Table 3-2: **SLL Components for Each SLR Crossing**

Virtex-7 Device	SLL Components
7V2000T	13,270
7VX1140T	10,560

The **7VX1140T** device has fewer SLL components because it has more DSP and Block Memory columns. These columns displace more interconnect tiles for the same given area.

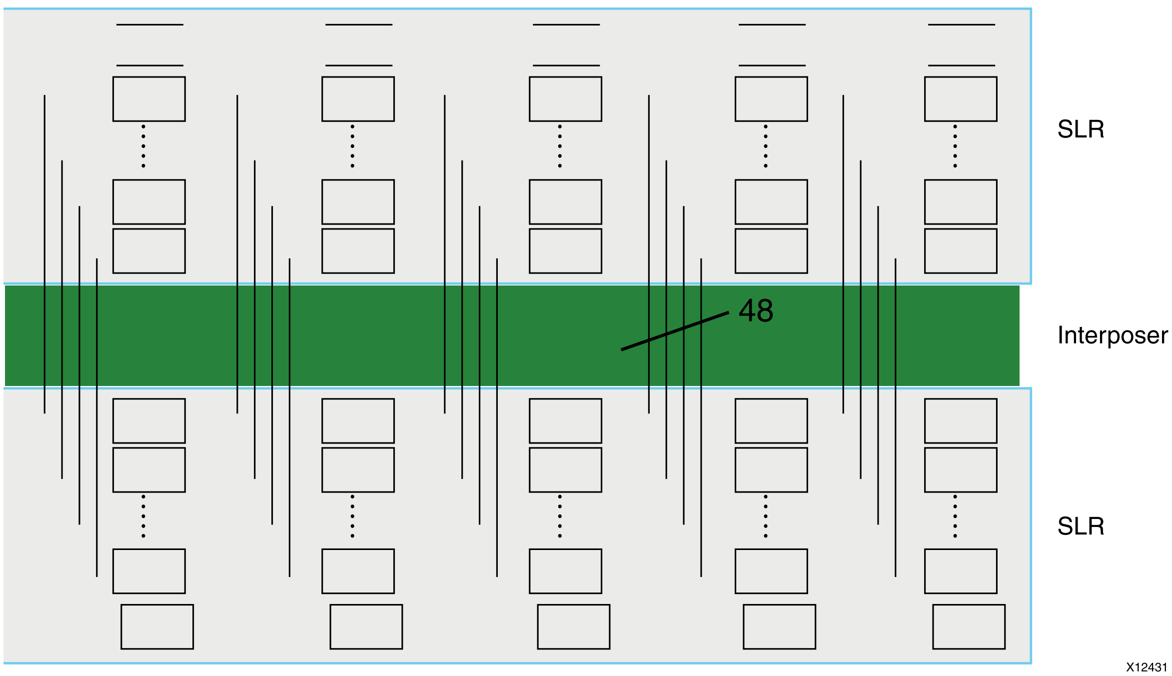

Figure 3-5: **Staggered SLLs Crossing in an SSI Device**

The ratios and gap size between SLR components is for illustration purposes only. The actual gap is comparatively much smaller.

The SLL components connect to the SLR at the center point of a Vertical 12 Long Line, which spans 12 interconnect tiles in the SLR.

This connectivity provides three optimal places to enter or exit an SLL from SLR to adjacent SLR, and gives additional flexibility to placement with little penalty to performance or power.

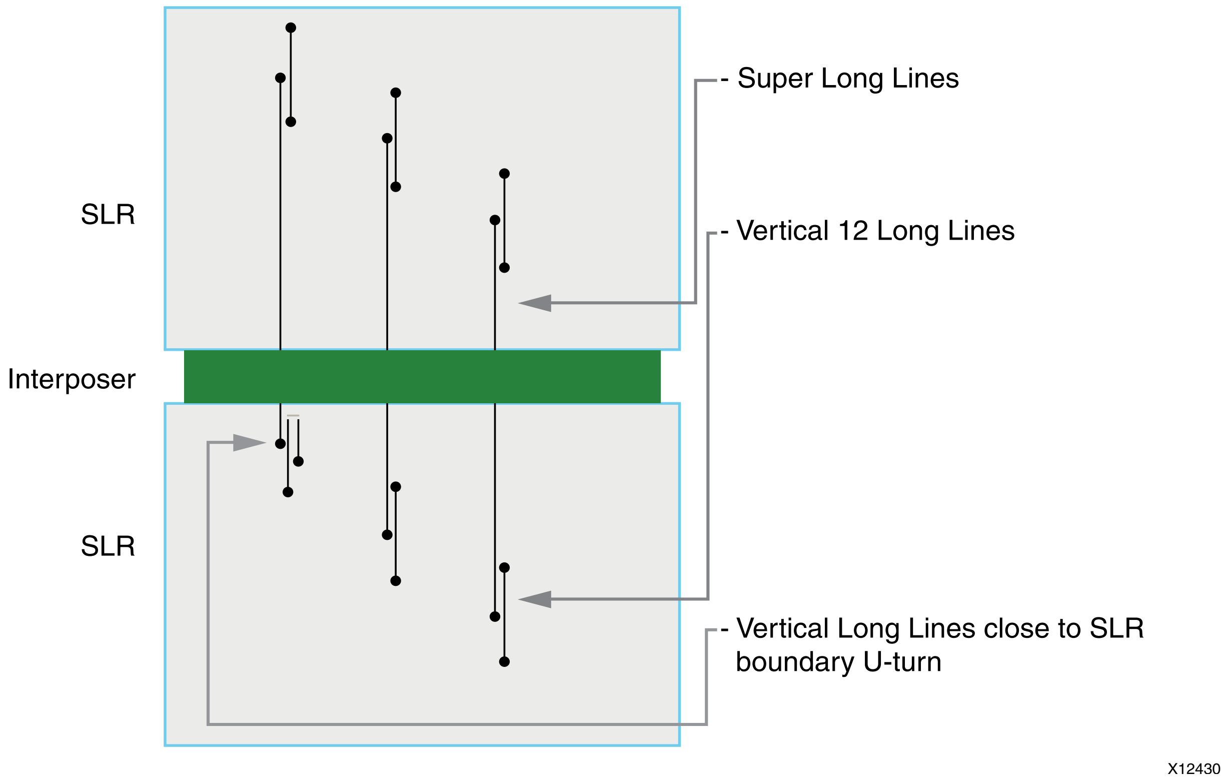

Figure 3-6: **Representation of SLL Connectivity in the SLR**

Propagation Limitations

SLL signals are the only data connections between SLR components.

The following do not propagate across SLR components:

- Carry chains
- DSP cascades
- Block RAM address cascades
- Other dedicated connections such as DCI cascades

The tools normally take this limit on propagation into account. To ensure that designs route properly and meet your design goals, you must also take this limit into account when you build a very long DSP cascade and manually place such logic near SLR boundaries; and when you specify a pinout for the design.

Master Super Logic Region (SLR)

Every SSI device has a single master SLR. In all SSI devices, **SLR1** is the master SLR.

The master SLR contains the primary configuration logic that initiates configuration of the device and all other SLR components.

The master SLR is the only SLR that contains dedicated circuitry such as:

- DEVICE_DNA
- USER_EFUSE
- XADC

To access this circuitry, place associated pins or logic into the SLR when manually constraining pins or logic to the device. When using these components, the place and route tools can assign associated pins and logic to the appropriate SLR. In general, no additional intervention is required.

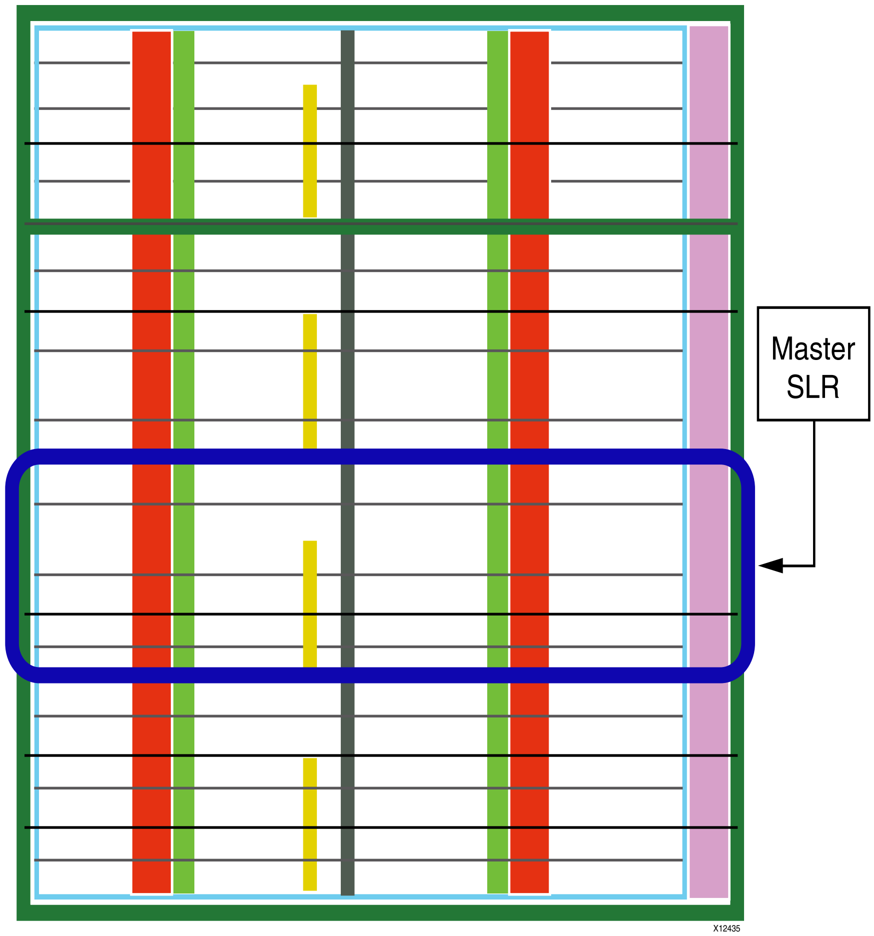

Figure 3-7: **Master SLR in an xc7v2000t Device**

Clocking

This section discusses clocking, and includes:

- Regional Clocking
- Global Clocking (BUFG)

Regional Clocking

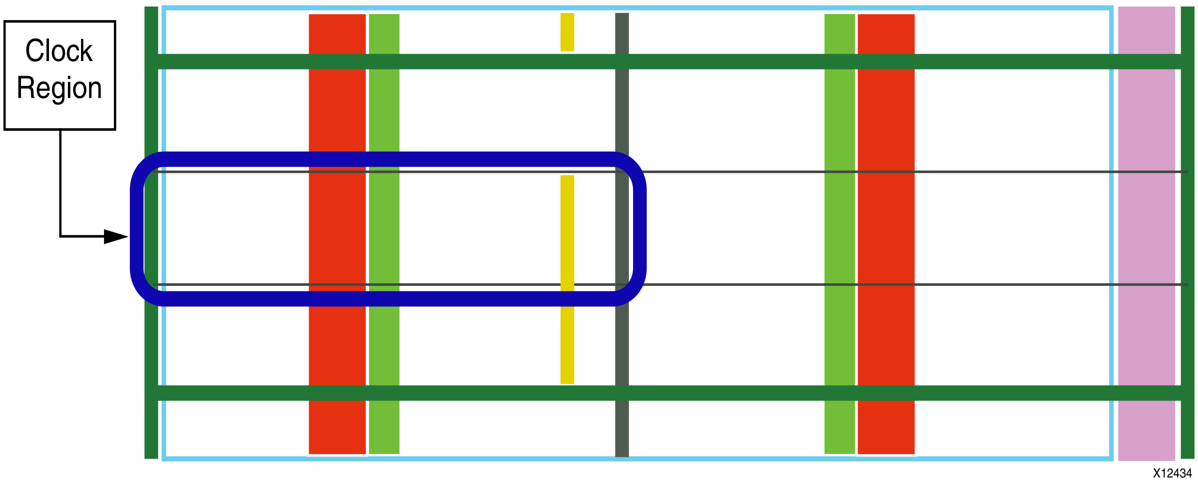

Figure 3-8: **SLR Showing Clock Regions (Enlarged)**

The clocking architecture for SSI devices is similar to other Xilinx 7 series FPGA devices.

The regional clocking from the following components have the same connectivity and behavior as in Xilinx 7 series FPGA devices:

- BUFIO
- BUFR
- BUFH

Exception

There is a single exception.

For a BUFMR or BUFMRCE, the buffer does not span across SLR components.

If a BUFMR or BUFMRCE is located in the clock region of an SLR that directly borders a different SLR, then the following limitations apply:

1. The BUFMR or BUFMRCE can access only:
   a. The clock region in which it is placed
   b. The clock region directly adjacent in the same SLR
2. The BUFMR or BUFMRCE cannot access the adjacent SLR.

Xilinx recommends that you place a BUFMR or BUFMRCE in the center clock region of a given SLR. This gives it full access to span the clock regions above and below.

It might not be necessary now to encapsulate all three clock regions for that particular clock domain. But doing so now gives you greater flexibility in providing the clock to all regions later on. Be sure to take this fact into account during clock and pin planning.

Global Clocking (BUFG)

Global clocking (BUFG) for SSI devices is also similar to other Xilinx 7 series FPGA devices.

- The global clocking topology is identical in the SLR.
- There are 32 available BUFG components that can span the entire device.
- Each BUFG component is capable of driving one of 12 horizontal clocks (BUFH) in a given clock region in the SLR.

For all SLR components in an SSI device (including clocking), make the same assumptions as for any other Xilinx 7 series FPGA device.

The BUFG components in an SLR can also clock synchronous elements in other SLR components. This is demonstrated by the connections and topology of the inter-SLR clocking. Each of the 32 BUFG components in an SLR drives one of 32 vertical tracks called the vertical global clocking line (also known as the global clocking backbone).

This connection traverses to the top and the bottom of the SLR for each BUFG, and allows connectivity to the horizontal row clocking.

At the boundary of the SLR, these vertical clocking spines connect to a very short interposer hop to connect the spine to the corresponding spine of the neighboring SLR.

This process drives the horizontal clocking resources of that SLR, and can continue up or down until all SLR components are connected. It creates a truly global clocking resource.

Because the vertical global clocking lines are a shared resource between the BUFG components of each SLR, some management of these resources might be required.

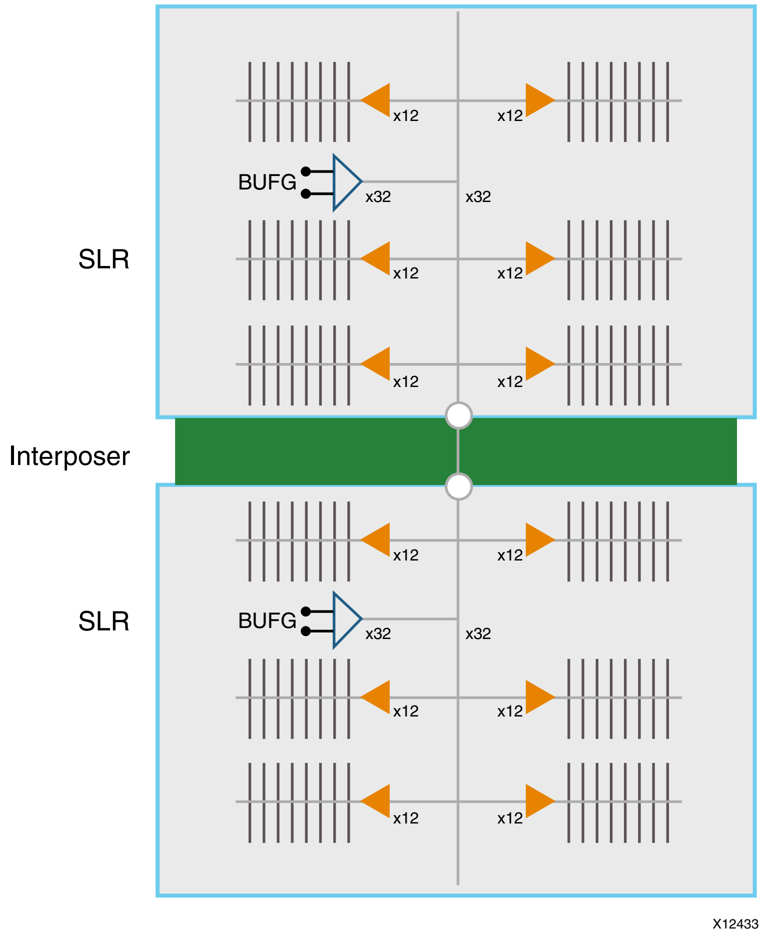

Figure 3-9: **Global Clocking Connections in an SSI Device**

The ratios and gap size between SLR components is for illustration purposes only. The actual gap is comparatively much smaller.

For more information, see Chapter 5, Clocking.

Management of Design Placement in SLR Components

This section discusses management of design placement in SLR components, and includes:

- Automatic SLR Assignment
- Manual SLR Assignment

Because SSI devices are composed of multiple SLR components, some management might be required. You must ensure that the design is properly placed in the device in order to route, function, and meet all timing goals.

Automatic SLR Assignment

Xilinx tools have built-in algorithms to manage the placement of resources in an SSI device. The tools automatically attempt to fit a design into the SSI part, and select logic placement with SLR components.

Placement Strategies

Using the built-in placement algorithms, the tools attempt to:

1. Place the design in a way that does not exceed SLL resources.
2. Limit the number of timing critical paths that must cross SLR components.
3. Balance the resources in a way that does not overly fill an SLR with a given resource.
4. Limit the number of SLL crossings to a minimum.

By following these strategies, the tools try to strike a balance placement while meeting performance requirements.

Benefits of Limiting SLL Crossing

Limiting SLL crossing in general reduces power, and allows for more design growth without impacting inter-SLR connectivity.

Other Factors That Influence SLR Selection

Other design and implementation factors can also influence SLR selection. These factors include:

1. Pin placement
2. Clock selection
3. Resource type
4. Physical constraints such as floorplanning (PBlocks) and LOC constraints
5. Timing constraints
6. I/O Standards and other constraints

Xilinx recommends that you allow the tools to assign SLR components while making intelligent pin placement, clock selection, and other design choices.

Floorplanning

Floorplanning might be necessary for high performance portions of the design. Floorplan only when necessary.

Manual SLR Assignment

Manual SLR assignment might be necessary when the tools do not find a solution that meets design requirements, or when run-to-run repeatability is important.

Performing Manual SLR Assignment

To perform manual SLR assignment:

1. Create large PBlocks (area groups).
2. Assign portions of the design to those area groups.

To assign large sections of the design to a single SLR:

1. Create a PBlock that encompasses a single SLR.
2. Assign the associated hierarchy of the logic to that PBlock.

While you can assign logic to multiple adjacent SLR components, you must ensure that the PBlock encompasses the entire SLR.

Do not create PBlocks that cross SLR boundaries without constraining the entire SLR. Doing so can it make it difficult for the automatic SLR placement algorithms to legalize placement.

Manual SLR Assignment Guidelines

When you manually assign logic to the SLR components, Xilinx recommends that you:

1. Place the design in a way that does not exceed SLL resources.
2. Limit the number of timing critical paths that must cross SLR components.
3. Balance the resources in a way that does not overly fill an SLR with a given resource.
4. Limit the number of SLL crossings to a minimum.

These guidelines are the same as the Placement Strategies followed in Automatic SLR Assignment. Following these guidelines makes it less likely that the assignments will violate present or future design rules.

For more information, see the *Floorplanning Methodology Guide (UG633)* cited in Appendix A, Additional Resources.

SSI Hierarchy

A well defined hierarchy for an SSI design facilitates SLR assignment. Register the outputs of the hierarchical instances in order to ease the timing requirements if those signals must cross SLR components.

Follow this recommendation as well when using partitions for:

- Design reuse
- Team design
- Partial reconfiguration

Achieving High Performance Design in SSI Devices

Additional manual intervention might be required in order to achieve very high speed designs in SSI devices.

The performance in an SLR is the same as the performance in a comparable speed grade of any other Virtex-7 device.

With few or no logic levels in the paths, performance at SLR boundary crossings can achieve speeds in excess of 400 MHz.

Pipelining

When designing high performance register-to-register connections for SLR boundary crossings, the appropriate pipelining must be described in the HDL code, and controlled at synthesis.

This ensures that the Shift Register LUT (SRL) inference and other optimizations do not occur in the logic path that must cross an SLR boundary.

Modifying the code in this manner defines where the SLR boundary crossing occurs. You must define the SLR assignment to correspond to those design changes.

Floorplanning

The following might also be required:

- Manual floorplanning
- LOC constraints or Area Group constraints

The LOC constraints and Area Group constraints confine the launch and destination registers for the SLR boundary crossing. This allows the tools to find the optimal placement for maximum speed.

These techniques are not always required, but might be necessary to achieve peak limits near the performance limits of the device.

SSI Configuration

Configuring an SSI device is similar to configuring any traditional device. The tools create a single bitstream. All configuration features (such as encryption and SEU) and configuration modes are supported.

Configuration Details

Multi-SLR configuration is handled entirely by the configuration circuitry and Xilinx tools. Each SLR has its own configuration engine, which is virtually identical to that of a traditional device. The Master SLR contains the master configuration engine. The configuration engines of all other SLR components are treated as slaves.

The Xilinx tools create a single bitstream. When loaded, the bitstream sequentially configures the individual SLR components to provide the correct portion of the bitstream to the appropriate SLR.

Signals Tied Together

The following signals are tied together in the interposer:

- INIT
- DONE
- KEYCLEAR

This allows functions such as *clear configuration key* to operate quickly and consistently on all SLR components at once. Configuration feedback signaling *completion of configuration* or *configuration error* behaves the same as in a traditional device.

Bitstream Decryption

For operations such as bitstream encryption, a single key is used in all SLR components. Using a single key simplifies the management of keys and configuration data, and allows the SSI device to appear and operate similar to all other Xilinx FPGA devices.

The bitstream decryption is performed in the SLR. The passing of data from SLR to SLR in the interposer remains encrypted to further prevent tampering or interception of configuration data.

Operations on a Per SLR Basis

While most operations behave identically as in a traditional device, the following operate on a per SLR basis:

- CAPTURE
- READBACK
- FRAME_ECC

This can help improve time to collect data, and, for ECC, correct any corrupted bits.

Components Existing Only in the Master SLR

Some configuration and device access components exist only in the Master SLR. Components such as the following are accessible only in the Master SLR:

- DEVICE_DNA
- USER_EFUSE
- XADC

While Boundary Scan exists in all SLR components, there is a weighted preference to use the Master SLR for its function.

Partial Reconfiguration

Like device configuration, partial reconfiguration is treated much the same as in traditional devices.

1. A single bitstream is created, whether you use external or internal methods to perform partial reconfiguration.
2. The master SLR directs the bitstream to the appropriate SLR components.

Chapter 4

System Level Design

This chapter discusses System Level Design, and includes:

- Pinout Selection
- Control Sets
- HDL Coding Styles

For information on clocking, see Chapter 5, Clocking.

Pinout Selection

This section discusses Pinout Selection, and includes:

- Consequences of Pinout Selection
- Using Xilinx Tools in Pinout Selection
- General Pinout Selection Recommendations
- Specific Pinout Selection Recommendations
- Device Migration

Consequences of Pinout Selection

Good pinout selection leads to good design logic placement.

Bad pinout selection leads to poor placement options for the logic that it drives. Poor placement creates longer routes. Longer routes increase power consumption, and reduce performance.

Bad pinout => Poor placement => Longer routes => Increased power, reduced performance

These consequences of bad pinout selection are particularly true for large FPGA devices. Because large FPGA devices exist in large dies, a spread out pinout can cause related signals to span longer distances to form the desired logic structures in the array.

Using Xilinx Tools in Pinout Selection

Xilinx® tools assist in design planning and pinout selection. These tools are only as effective as the information you provide them.

Tools such as the PlanAhead™ design analysis tool can assist pinout efforts. These tools can:

- Graphically display the I/O placement.
- Show relationships among clocks and I/O components.
- Provide Design Rule Check (DRC) capability to analyze pin selection.

Required Information

For the tools to work effectively, you must provide as much information about the I/O characteristics and topologies as possible.

You must specify the electrical characteristics, including:

- I/O Standard
- Drive
- Slew

You must also take into account all other relevant information, including:

- Clock topology
- Timing constraints

Clocking choices in particular can have a significant influence in pinout selection, and vice-versa. Pinout choices can also have a large influence on clocking choices.

General Pinout Selection Recommendations

In general, choose a pin selection that keeps related signals close together, and closer to the loads that they will eventually drive.

You can easily miss this detail when refining pinouts for board layout purposes or for late ECO changes. However, creating and maintaining a good pinout is important for good FPGA design. This is especially true for large FPGA designs that can consume over 1000 interface pins.

Pins close together in the device package might not necessarily be close together in the device array. For internal timing, having pins close together in the array is more important than having them close together in the device package.

Specific Pinout Selection Recommendations

Xilinx recommends the following specific pinout selection with respect to:

- Interface Data, Address, and Control Pins
- Interface Control Signals
- Very High Fanout, Design-Wide Control Signals
- Xilinx IP Containing I/O Interfaces
- CCIO and CMT Usage
- Components Located in a Particular SLR (SSI)

Interface Data, Address, and Control Pins

Group the same interface data, address, and control pins into the same bank.

- If you cannot group these components into the same bank, group them into adjacent banks.
- For SSI devices, group all pins of a particular interface into the same SLR.

Interface Control Signals

Place the following interface control signals in the middle of the data buses they control:

- Clocking
- Enables
- Resets
- Strobes

Very High Fanout, Design-Wide Control Signals

Place very high fanout, design-wide control signals towards the center of the device.

For SSI devices, place the signals in the SLR located in the middle of the SLR components they drive.

Xilinx IP Containing I/O Interfaces

For Xilinx IP containing I/O interfaces, such as Memory Interface Generator (MIG), generate the interface and use the recommended pinout.

CCIO and CMT Usage

Balance CCIO and CMT usage between the upper and lower halves of the device in order to balance the access to upper and lower BUFG components.

For SSI devices, balance upper and lower CCIO components or CMT components in an SLR against the other SLR components.

Components Located in a Particular SLR (SSI)

For SSI devices, when planning pinouts for components that are located in a particular SLR, place the pins into the same SLR.

For example, when using the device DNA information as a part of an external interface, place the pins for that interface in the master SLR in which the DEVICE_DNA exists.

Device Migration

Many Xilinx devices allow you to migrate your design to a larger or smaller device in the same package. To migrate your design with reduced risk, carefully plan the following at the beginning of the design process:

- Device selection
- Pinout selection
- Design criteria

Take the following into account when migrating to a larger or smaller device in the same package:

- Pinout
- Clocking
- Resource management

Control Sets

This section discusses Control Sets, and includes:

- About Control Sets
- Resets

About Control Sets

A control set is the grouping of control signals that drive a specific RAM or register. Control signals include:

- Set/reset
- Clock enable
- Clock

A unique control set is formed for every unique combination of control signals.

Because registers in a slice share common control signals, only registers with a common control set can be packed into the same slice.

Designs with several unique control sets might exhibit:

- Lower device utilization
- Fewer options for placement

This can result in higher power and lower performance.

Designs with fewer control sets have more placement options and flexibility. This generally gives better results.

Resets

This section discusses Resets, and includes:

- About Resets
- When and Where to Use Resets
- Defining an Initial State on Inferred Synchronous Elements
- Synchronous and Asynchronous Resets
- Active-High and Active-Low Resets

About Resets

Resets are one of the most common and important control signals. If left unmanaged, resets can negatively affect performance, area, and power.

Inferred synchronous code might not only result in LUTs and registers, but many design elements other than registers. Shift Register LUTs (SRLs), Block or LUT Memory, DSP48 registers and other resources can result from general synchronous code. However, the choice and use of reset can affect the selection of such components, resulting in less optimal resources used for a given design

- A misplaced reset on an array can mean the difference between inferring one block RAM component or several thousand registers.
- A reset described unnecessarily on a delay line can mean the difference from a few Shift Register LUT (SRL) LUTs to several hundred registers.
- An asynchronous reset described at the input or output of a multiplier can result in registers placed in the slice rather than the DSP block.

These can significantly impact:

- Resources
- Power
- Performance

When and Where to Use Resets

A reset is not required to initialize the device.

FPGA devices have dedicated global set/reset signals (GSR). At the end of device configuration, the GSR is automatically asserted to initialize all registers to the initial state specified in the HDL code.

If an initial state is not specified, it generally defaults to a logic **0** (zero). Every register is at a known state at the end of configuration, regardless of the reset topology specified in the HDL code. You do not need to code a global reset for the sole purpose of initializing the device.

Limiting Reset Use

Limiting reset use can do the following:

- Limit the fanout of the reset net.
- Reduce the amount of interconnect necessary to route the reset.
- Simplify the timing of the reset paths.
- Improve performance and power.

Deciding Whether a Reset is Required

Use care in deciding whether a reset is required.

1. Evaluate each synchronous block.

   If you are not sure whether a reset is required, do not code the reset.
2. Use functional simulation.

   If the functional simulation operates correctly, it should operate correctly in the implemented design.

When No Reset is Coded

For logic in which no reset is coded, there is greater flexibility in selecting FPGA resources to map the logic.

For example, for a simple delay line (shift register), if a reset is coded, the tools will likely map that into a set of registers with a common reset.

If a reset is omitted, that same logic can result in:

- A Shift Register LUT (SRL)
- A combination of SRL and registers
- All registers
- LUT or block memory

The synthesis tool can then select the best resource for that code, taking into account:

- Functionality
- Performance requirements
- Available device resources
- Power

Defining an Initial State on Inferred Synchronous Elements

The GSR net initializes all registers to the specified initial value in the HDL code.

- If no initial value is supplied, the synthesis tool can assign the initial state to either **0** (zero) or **1** (one).
- The initial state generally defaults to **0** (zero), with a few exceptions such as One Hot state machine encoding.

Any inferred Shift Register LUT (SRL), memory, or other synchronous element, can also have an initial state defined that will be programmed into the associated element upon configuration.

Xilinx recommends initializing all synchronous elements accordingly. Initialization of registers is completely inferable by all major FPGA synthesis tools. All synchronous elements start with a known value in the FPGA device after configuration.

Starting with a known value does the following:

- Lessens the need to add a reset for the sole purpose of initialization.
- Makes the RTL code more closely match the implemented design.
- Allows functional (RTL) simulation to come up in a known state.

Synchronous and Asynchronous Resets

If resets are required, Xilinx recommends that you code synchronous resets.

Synchronous resets have many advantages over asynchronous resets. Synchronous resets can directly map to more resource elements in the FPGA architecture.

Some resources such as DSP48 components and block RAM components have only synchronous resets for the register elements in the block.

When asynchronous resets are used on register elements associated with these elements, those registers can not be inferred directly into those blocks without a functional difference.

For example, for a **7v2000t** device, approximately 650,000 DSP registers and 93,000 BRAM registers are accessible only if an asynchronous reset is *not* described. Those registers support synchronous resets only.

Synchronous resets also give more flexibility for control set remapping if higher density or fine tuned placement is required.

Active-High and Active-Low Resets

For the most flexibility in routing and placement, do not mix polarities for high fanout control signals such as:

- Clock enables
- Resets

Xilinx recommends active-High. Active-High can often map into the FPGA architecture with less logic and fewer logic levels.

Polarity

A consistent polarity throughout the design is even more important. Mixed polarities can create mixed control sets. Mixed control sets can negatively impact placement and routing, and, in the worst cases, cause timing and fitting issues into the FPGA device.

Virtex-6 and Virtex-7 Devices

For the slice and internal logic of Virtex-6 devices and Virtex-7 devices, all clock enables and resets are inherently active-High.

Describing active-Low resets or clock enables can result in additional LUTs being used as simple inverters for those routes.

HDL Coding Styles

A good HDL coding style has the following advantages:

- It makes the job of the synthesis tool easy.
- It runs fast in simulation and synthesis.
- It is portable between different FPGA architectures.
- It translates well to the target architecture.
- It is easy to read and debug.

Ways to achieve a good HDL coding style include:

- Using HDL constructs that result in efficient inference to the device resources
- Carefully considering:
  - Logic
  - Clocking
  - Control Signals
- Good hierarchy decisions
- Careful use of synthesis attributes

Inference to Device Resources

You must take into account the key arithmetic, storage, and logic elements in the targeted architecture.

When you are coding the design, understanding and anticipating the synthesis mapping can provide early insight into potential problems.

The following guidelines show how RTL code can be mapped into a Xilinx FPGA resource.

Larger Than 4 Bit Addition, Subtraction, and Addition-Subtraction

For larger than 4 bit addition, subtraction, and addition-subtraction, a carry chain is generally used. One LUT per 2 bit addition is also used.

For example, an 8 bit by 8 bit adder uses 8 LUTs and the associated carry chain.

For ternary addition (or when the result of an adder is added to another value without using a register in between), one LUT per 3 bit addition is used. For example, an 8 bit by 8 bit by 8 bit addition also uses 8 LUTs and the associated carry chain).

If more than one addition is required, it might be advantageous to specify registers after every two levels of addition. This cuts device utilization in half by allowing a ternary implementation to be generated.

Multiplication

In general, multiplication is targeted to DSP blocks.

- Signed bit widths less than 18x25 map into a single DSP block.
- Unsigned bit widths less than 17x24 map into a single DSP block.
- Multiplication requiring larger products might map into more than one DSP block.

Pipelining properly for logic inferred into the DSP block can greatly improve performance and power. When a multiplication is described, three levels of pipelining around it gives the best clock frequency, setup, clock-to-out, and power characteristics.

Very light pipelining (one level or none) might lead to timing issues and increased power for those blocks.

Shift Registers or Delay Lines

Shift registers or delay lines that do not require reset or multiple tap points are generally mapped into Shift Register LUT (SRL) components.

The following can be mapped into a single LUT:

- Two SRL components with 16 bits or less depth
- Single SRL components up to 32 bits

In order to best utilize SRL components, consider very careful reset specifications for those blocks. If a reset is not necessary, you might achieve better device utilization, performance, and power.

Memory Arrays Up to 64 Bits

Memory arrays described up to 64 bits deep are generally implemented in LUTRAM components.

- Depths 32 bits and fewer are mapped two bits into a LUT.
- Depths up to 64 bits can be mapped one bit per LUT.

Deeper RAMs can also be implemented in LUTRAM depending on:

- Available resources
- Synthesis tool assignments

Slight deviations in coding styles for these blocks can result in using the wrong resources. For example, an asynchronous write or reset coding that results in changing the array value can cause this to be implemented in an array of registers rather than LUTRAM.

Memory Arrays Deeper than 256 Bits

Memory arrays deeper than 256 bits are generally implemented in block memory. Virtex-6 devices and Xilinx 7 series FPGA devices create flexibility to map these structures in different width and depth combinations. Understanding these configurations helps you to understand the number and structure of block RAM components used for larger memory array declarations.

Slight deviations in coding styles for these blocks can result in using the wrong resources. For example, an asynchronous read or reset coding resulting in reset of the array can cause this to be implemented in an array of registers rather than LUTRAM or arrays of registers.

Conditional Code Resulting in Standard MUXes

Table 4-1: **Conditional Code Resulting in Standard MUXes**

MUX	Implemented Into	Logic (LUT) Levels
4-to-1	• Single LUT	One
8-to-1	• Two LUTs • One MUXF7	One
16-to-1	• Four LUTs • Combination of MUXF7 and MUXF8 resources	One

Understanding this code can lead to better resource management, and help you better understand and control the logic levels for the data paths in the design.

General Logic

For general logic, take into account the number of unique inputs for a given registered output. From that number, you can arrive at an estimation of LUTs and logic levels.

- Six (6) inputs or less results in one (1) logic level.
- Eleven (11) inputs or less can be placed in two (2) or fewer logic levels.

The larger the number of inputs, and the more complex the logic equation, the more LUTs and logic levels are required.

Taking into account the number of logic levels early allows easier modification up front rather than later during timing closure.

Choosing Good Design Hierarchy

Design hierarchy is often defined in part by:

- The separate logical sections of the design
- Use of cores or IP
- The hierarchy definition of legacy code

Design Hierarchy Guidelines

Guidelines for defining design hierarchy include:

- Register the Outputs of Data Paths
- Place Clocking Elements Toward the Top Level
- Infer I/O Components

Register the Outputs of Data Paths

If possible, register the outputs of the larger hierarchical models, especially towards the top level.

This might improve timing, especially when hierarchy boundary optimization is prevented because of:

- Hierarchical design methods
- Floorplanning
- Debugging

This methodology always contains critical paths in modules or at the boundary of one module to another, rather than potentially crossing several hierarchies. This can be difficult to analyze and repair if timing or functionality becomes an issue.

Place Clocking Elements Toward the Top Level

Placing clocking elements toward the top level allows easier clock sharing between modules, and helps in resource utilization management. It might also require fewer clocking resources., and improve performance and power.

Infer I/O Components

Infer I/O components when possible.

- When instantiation is required, place I/O components towards the top level of the code.
- For hierarchical design and partial reconfiguration, place the components at or near the top level.
- I/O debugging is also simplified when the circuitry is not buried deep in the hierarchy.

Hierarchical Design

Well defined boundaries divided into suitable areas facilitate hierarchical design methodologies such as:

- Partitions
- Partial reconfiguration
- Team design

Team design often forces such a division.

Areas might need to be broken out separately if they are sensitive to factors such as:

- Timing closure
- Functional debugging

A well defined hierarchy helps with floorplanning, and can greatly assist in achieving timing closure later. Critical paths in a level of hierarchy that are grouped together can often simplify floorplanning.

For SSI devices, you might need to manually assign logic to the individual SLR components.

- Defining the hierarchy so that all the logic can be assigned to a common SLR makes such assignments easier to create.
- Defining the hierarchy so that all the logic can be assigned to a common SLR allows place and route to complete faster.
- Defining the hierarchy so that all the logic can be assigned to a common SLR facilitates design analysis.

Functional and Timing Debugging

Timing debugging is easier when critical timing paths are localized to areas of the logical design.

The Xilinx tools allow designs to have gate-level timing netlists written out by hierarchy. This can facilitate and simplify the timing and functional debugging of a large FPGA design.

Pipelining

Two major factors can limit timing closure:

- Poor placement leads to routes that are too long to satisfy timing constraints.
- There are too many logic levels to allow the design to operate at the designed speed.

Benefits of pipelining can include the following:

- Pipelining might be the only way to reduce logic levels enough to meet timing requirements.
- Planning pipelining sooner (rather than later) can greatly simplify timing closure.

  Adding pipelining to certain paths can propagate latency differences across the circuit. One seemingly small change can turn into a major redesign of portions of the code.
- Identifying pipelining opportunities early in the design can:
  - Facilitate timing closure.
  - Reduce implementation runtime (because of easier-to-solve timing problems).
  - Reduce device power consumption (because of reduced switching of logic)

Managing Fanout Non-Clock Nets

Managing high fanout nets is easier early in the design process. What constitutes a high fanout net is often determined by performance requirements and the construction of the paths.

Observe nets with loads measured in the thousands early to understand their possible impact on the design.

Mitigating High Fanout Nets

When a high fanout net is identified, mitigation techniques include:

- Reduce Loads
- Use BUFG and BUFH
- Manage Replication in the Tools

Reduce Loads

Reduce loads to portions of the design that do not require it.

- For high fanout control signals, determine whether all coded portions of the design require that net to function properly. Reducing the load demand can greatly improve timing.
- For data paths, determine whether restricting the logic can reduce fanout.

Use BUFG and BUFH

Use BUFG and BUFH components to drive the net.

For more information, see Using Clock Buffers for Non-Clock Nets in Chapter 5, Clocking.

Manage Replication in the Tools

Manage replication in the tools.

Most synthesis tools can automatically or manually control replication of register or logic to help minimize the impact of large fanout signals.

This capability can greatly improve timing for high fanout nets, often without modifying any RTL code.

Chapter 5

Clocking

The design decisions you make before the first synthesis run, or even before you write the first line of HDL code, can significantly impact your design goals. Smart planning and a small additional investment of time early on ensures good up front decisions and saves project time.

Selecting Clocking Resources

Xilinx® recommends that you select clocking resources as one of the first steps of your design, well before pinout selection. Your clocking selections can dictate a particular pinout, and can also direct logic placement for that logic. Proper clocking selections can yield superior results.

Virtex®-6 and Virtex-7 architectures contain 32 global clock buffers known as BUFG.

BUFG can meet most clocking requirements for designs with less demanding requirements with respect to:

- Number of clocks
- Design performance
- Low power demands
- Other clocking characteristics such as:
  - Clock gating
  - Multiplexing
  - Other clocking control

BUFG components are easily inferred by synthesis, and have few restrictions, allowing for most general clocking.

However, if clocking demands exceed the capabilities of BUFG, or if better clocking characteristics are desired, Xilinx recommends that you:

1. Analyze the clocking needs against the available clock-in resources.
2. Select and control the best resource for the task.

Global Clocking

Global clocking buffers have other functionality besides clocking. This additional functionality can be accessed only by manually intervening in the design code, or during synthesis.

Global clocking buffers include:

- BUFGCE
- BUFGMUX
- BUFGCTRL
- IP and Synthesis

BUFGCE

The BUFGCE primitive allows access to a synchronous, glitchless clock enable (gating) capability, and does not use any additional logic or resources.

Use BUFGCE to stop the clock for a period of time and create lower skew and lower power clock division such as one half (½) or one fourth (¼) frequency clocks from a higher frequency base clock.

This is particularly useful for generating different frequencies at different times of circuit operation.

BUFGMUX

Use BUFGMUX to safely change clocks without glitches or other timing hazards from one clock source to another, and to generate different clock frequencies for different time or operating conditions.

BUFGCTRL

Use BUFGCTRL to access all capabilities of the global clocking network.

This allows for asynchronous control of the clocking for more complicated clocking scenarios, such as a lost or stopped clock switchover circuit.

IP and Synthesis

IP and synthesis can also use these more advanced clocking features.

For example, when using the Memory Interface Generator (MIG), special clocking buffers can be used for high speed data transmit and capture at the I/O.

During your clocking architecture and planning, you must take into account the clock resources used for the individual IP.

Usually, however, in order to achieve the desired clocking behavior, the component must be instantiated in the code, and the proper connections made.

For more information, see the *Clocking Guides* and *Libraries Guides*, cited in Appendix A, Additional Resources.

Regional Clocking

This section discusses Regional Clocking and includes:

- Horizontal Clock Region Buffers (BUFH, BUFHCE)
- Regional Clock Buffers (BUFR)
- I/O Clock Buffers (BUFIO)
- Multi-Regional Clock Buffers (BUFMR)

Horizontal Clock Region Buffers (BUFH, BUFHCE)

Horizontal clock region buffers (BUFH, BUFHCE) can be used:

- With BUFG
- As standalone buffers

Uses for Horizontal Clock Region Buffers

Use horizontal clock region buffers to cain tighter control of the clocking and placement of the associated logic connected to the clock, and to provide additional clocking resources for designs with a high number of clock domains.

The BUFH and BUFHCE resources allow the design to use the portions of the global clock network (BUFG) that connect to a given clock region. This allows access to a low skew resource from otherwise unused portions of the global clock network for smaller clock domains that can be constrained in a clock region.

Glitchless Clock Enable

The BUFHCE has the same glitchless clock enable as BUFGCE allowing for simple and safe clock gating of a particular clock domain.

When used independently of the BUFG network, the BUFH is a simple buffer representation of the BUFHCE, and thus is the same resource.

Xilinx recommends using BUFH when a clock enable is not required.

Medium Grained Clock Gating

The BUFHCE can be used as a medium grained clock gating function when driven by a BUFG.

The BUFHCE can be an effective clocking resource for portions of a clock domain ranging from a few hundred to a few thousand loads in which you want to stop clocking intermittently.

A BUFG can drive multiple BUFH components in the same or different clock regions. This allows for several low skew clock domains in which the clocking can be individually controlled.

Using BUFH and BUFHCE Independently of BUFG

BUFH and BUFHCE can also be used independently of BUFG to add additional clocking capabilities.

When used independently, all loads connected to the BUFH must reside in the same clock region. This makes it well suited for very high speed, more fine grained (fewer loads) clocking needs.

You must ensure that:

1. The resources driven by the BUFH do not exceed the available resources in the clock region.
2. No other conflicts exist.

The phase relationship might be different between the BUFH and clock domains driven by BUFG components, other BUFH components, or any other clocking resource.

The one exception occurs when two BUFH components are driven to horizontally adjacent regions. In this case, the skew between left and right clock regions when both BUFH components driven by the same clock source should have a very controlled phase relationship in which data can safely cross the two BUFH clock domains.

Regional Clock Buffers (BUFR)

The Regional Clock Buffer (BUFR) is generally used as the slower speed I/O and fabric clock for capturing and providing higher speed I/O data.

Uses for Regional Clock Buffers

Use Regional Clock Buffers (BUFR) to enable or disable (gate) the clock, and to perform common divisor clock division.

In Virtex-6 devices, BUFR can drive:

- The clock region in which it exists.
- The clock region above and below.

Medium Grained Clock Gating

BUFR is well suited for many medium grained clocking needs. In a Virtex-7 device, BUFR can drive only the region in which it exists. It is thus better suited for slightly smaller clocking networks.

Because the performance of BUFR is somewhat lower than BUFG and BUFH, Xilinx does not recommend BUFR for very high speed clocking. However, BUFR is well suited for many medium to lower speed clocking needs.

The added capability of built-in clock division makes BUFR suitable for divided clock networks coming from an external clock source such as a high speed I/O interface clock.

I/O Clock Buffers (BUFIO)

Use the I/O Clock Buffer (BUFIO) to:

- Capture I/O data into the input logic.
- Provide an output clock to the output logic to the device.
- Capture high speed, source synchronous data in a bank.
- Gear down the data to more manageable speeds when used with a BUFR and an ISERDES or OSERDES logic.

A BUFIO can drive only the input and output components that exist in the **ILogic** and **OLogic** structures such as:

- IDDR
- ODDR
- ISERDES
- OSERDES
- Simple dedicated input or output registers.

When using BUFIO, you must be sure to reliably transfer the data from the I/O logic to the fabric, and vice versa.

Multi-Regional Clock Buffers (BUFMR)

The Multi-Regional Clock Buffer (BUFMR) is new for Xilinx 7 series FPGA devices, and is not available, and is unnecessary, in Virtex-6 devices and before.

The BUFMR allows a single clock pin (MRCC) to drive the BUFIO and BUFR:

- In its bank
- In the I/O banks above and below it (if any)

For more information about targeting SSI devices, see Chapter 3, Stacked Silicon Interconnect (SSI).

Clocking for SSI Devices

In general, all standard clocking rules apply to SSI devices. However, there are a few additional considerations when targeting SSI devices.

Regional clocking is the same, except that a BUFMR cannot drive clocking resources across an SLR boundary.

Xilinx recommends placing the clocks driving BUFMR into the bank or clocking region in the center clock region in an SLR. This gives access to all three clock regions on the left or right side of the SLR.

Designs Requiring 16 or Fewer Global Clocks

For designs requiring 16 or fewer global clocks (BUFG components), no special considerations are necessary with respect to global clocking. In addition, the tools automatically assign BUFG components to avoid any possible conflicts.

Designs Requiring More Than 16 But Fewer Than 32 Global Clocks

When more than 16 (but fewer than 32) BUFG components are required, pin selection and placement must be considered in order to avoid any possibility of contention of resources based on global clocking line contention, placement of clock loads, or both.

As in all other Xilinx 7 series FPGA devices, Clock-Capable I/O (CCIO) components and their associated Clock Management Tile (CMT) have restrictions on the BUFG components they can drive in the SLR.

CCIO components in the top or bottom half of the SLR can drive BUFG components only in the top or bottom half of the SLR (respectively). Accordingly, pin and associated CMT selection must be done in a way in which no more than 16 BUFG components are required in either the top or bottom half of all SLR components collectively.

The tools automatically assign all BUFG components in a way that allows all clocks to be driven to all SLR components without contention.

Designs Requiring More Than 32 Global Clocks

For designs requiring more than 32 global clocks, Xilinx recommends that you use BUFR and BUFH for smaller clock domains. Doing so reduces the number of global clock domains.

BUFR components that use a BUFMR can drive resources in three clock regions which encompasses one half of an SLR. This equals approximately 250,000 logic cells in a Virtex-7 device class SLR.

In some horizontally adjacent clock regions, both left and right HROW buffers can be driven in a low skew manner enabling a clocking domain of one third of an SLR. This equals approximately 167,000 logic cells.

Using these resources can:

- Lead to fewer considerations for clocking resource contention.
- Improve design placement.
- Improve performance and power.

Segmenting BUFG Global Clocking Spines

If more than 32 global clocks are required, you can segment the BUFG global clocking spines.

Isolation buffers exist on the vertical global clock lines at the periphery of the SLR components that allow use of two BUFG components in different SLR components that occupy the same vertical global clocking track without contention.

In most cases the Xilinx software can automatically manage the placement and allocation of BUFG components, even when more than 32 are required. In some more complex situations however, manual placement of BUFG components, the associated logic being driven by that resource, or both, might be needed.

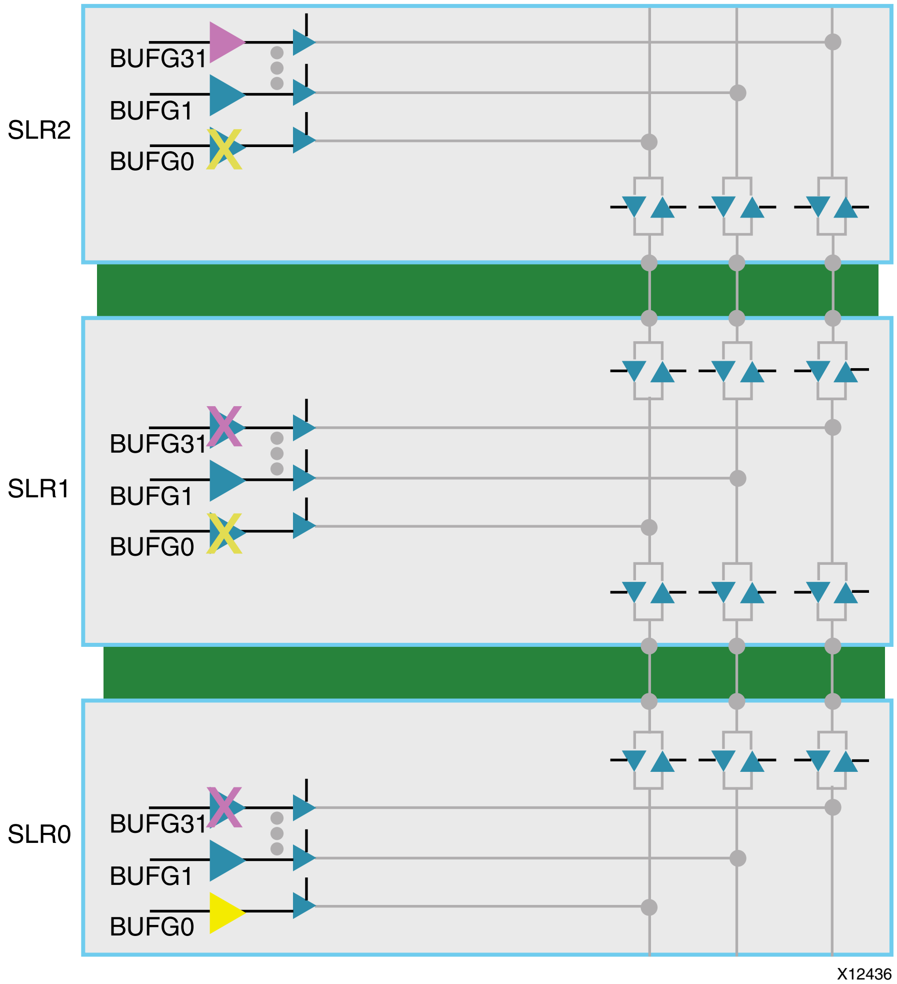

Figure 5-1: **SSI Global Clocking Distribution and Shared Resources**

Clock Skew in SSI Devices

Clock skew in a large FPGA device can represent a significant portion of the timing budget for a given path.

If left unmanaged and unchecked, excessive clock skew can create issues with:

- Maximum clock speed
- Hold times

SSI devices resemble other large FPGA devices in this respect. Skew is represented by:

- Unevenness in the distribution of the clock network from the source to destination.
- Small differences in process voltage and temperature (PVT) that can speed up or slow down portions of the clock path in respect to the data path.

Multiple Die

Multiple die SLRs in a device exaggerate the process portion of the PVT equation.

Multiple die SLRs are managed by the Xilinx assembly process, in which only die of similar speed are ever packaged together.

Even with that extra action, the Xilinx timing tools account for these differences as a part of the timing report. During path analysis, the tools analyze these aspects as a part of the setup and hold calculations. The tools then report them as a part of the path delay against the specified requirements.

No additional calculations or considerations are necessary for SSI devices. The timing analysis tools take this into account.

Path Delay Differences

Path delay differences can also be exaggerated if you are using the top or bottom SLR. The difference to reach points farther away from each other might see more skew.

For this reason, Xilinx recommends placing global clocks that must drive more than one SLR into the center SLR. This allows a more even distribution of the clocking network across the part, generally resulting in less clock skew.

Specifying the Clocking in the Design

This section discusses Specifying the Clocking in the Design, and includes:

- Inference
- Synthesis Attributes
- Intellectual Property (IP)
- Instantiation

Inference

The synthesis tool automatically specifies a global buffer (BUFG) for any clock structures up to the maximum allowed in an architecture, unless otherwise specified or controlled by the synthesis tool.

The BUFG provides a well controlled, low skew network suitable for most clocking needs. Nothing more is required unless your design clocking exceeds the number or capabilities of BUFG components in the part.

Controlling the clocking structure can result in better characteristics with respect to factors such as:

- Jitter
- Skew
- Placement
- Power
- Performance

Synthesis Attributes

Using the proper synthesis attributes is a simple way to control clocking resources.

Uses of Synthesis Attributes

- Synthesis attributes can prevent BUFG inference.
- Synthesis attributes can replace a BUFG with an alternative clocking structure.
- Synthesis attributes can specify a clock buffer when one would not otherwise exist.
- Synthesis attributes can control clocking resources without modifying the HDL source code.

Using the BUFFER_TYPE Attribute in XST

You can use the BUFFER_TYPE attribute in the Xilinx Synthesis Technology (XST) tool to:

- Not specify a clocking buffer.
- Specify a specific buffer type such as BUFH or BUFR.

The BUFFER_TYPE attribute can be placed:

- Directly in the HDL source code

  This allows the constraint to persist in the code.
- In the XST Constraint File (XCF)

  This allows you to control clocking resources without modifying the HDL source code.

Intellectual Property (IP)

Some intellectual property (IP) can help create clocking structures, including:

- Clocking Wizard and I/O Wizard
- Complex IP

Clocking Wizard and I/O Wizard

Clocking Wizard and I/O Wizard can help select and create clocking resources and structures, including:

- BUFG
- BUFIO
- BUFR
- Clock modifying blocks such as:
  - Mixed Mode Clocking Manager (MMCM)
  - I/O Phase Lock Loop (PLL)

Complex IP

Complex IP such as the following can also include clocking structures:

- Memory Interface Generator (MIG)
- PCIe
- Transceiver Wizard

This IP can provide additional clocking resources when properly taken into account.

If not properly taken into account, this IP can limit some clocking options for the remainder of the design.

Xilinx recommends that, for any instantiated IP, you leverage the clocking requirements, capabilities, and resources in other portions of the design.

Instantiation

The most low level and direct method of controlling clocking structures is to instantiate the clocking resources into the HDL design.

- Benefits of Instantiation
- Instantiating Clocking Resources at the Top
- Dangers of Redundant Clocking Resources

Benefits of Instantiation

Instantiating the clocking resources into the HDL design allows you access and control all capabilities of the device.

Instantiation is generally the only option for clocking structures that require extra logic and control, including:

- BUFGCE
- BUFGMUX
- BUFHCE

Even for simple buffers, the quickest solution is to be direct and instantiate it into the design.

Instantiating Clocking Resources at the Top

Xilinx recommends placing the clocking resources in a separate entity or module at the top, or near the top, of the code.

An entity or module at or near the top level can be more easily distributed to multiple modules in the design.

Dangers of Redundant Clocking Resources

- Redundant clocking resources can waste FPGA resources.
- Redundant clocking resources can consume more power.
- Redundant clocking resources can create conflicts and placement decisions that result in:
  - Longer implementation tool runtimes
  - More complex timing scenarios

Controlling Clock Phase, Frequency, Duty Cycle, and Jitter

This section discusses Controlling Clock Phase, Frequency, Duty Cycle, and Jitter, and includes:

- Using Clock Modifying Blocks
- Using IDELAY to Control Phase
- Using Gated Clocks
- Reducing Dynamic Power

Using Clock Modifying Blocks

You can use an MMCM or PLL to change the characteristics of an incoming clock.

MMCM is most commonly used to remove the insertion delay of the clock (phase align the clock to the incoming system synchronous data).

The MMCM can also be used to:

- Create tighter control of phase.
- Filter jitter in the clock.
- Change the clock frequency.
- Correct or change the clock duty cycle.

This gives tight control over an important aspect of the design.

Using MMCM and PLL components is fairly common for conditioning and controlling the clocking resource.

In order to use the MMCM or PLL, several attributes must be coordinated to ensure that the MMCM is operating in specification, and delivering the desired clocking characteristics on its output.

Clocking Wizard

Xilinx recommends using the Clocking Wizard to configure this resource.

Direct Instantiation

The MMCM or PLL can also be directly instantiated.

While direct instantiation allows even more control over the resource, you must exercise extra caution to ensure that the proper settings are used to obtain the desired result and to meet timing goals.

Incorrect settings on the PLL can increase clock uncertainty because of increased jitter; can build incorrect phase relationships; and can cause other problems that make timing more difficult than necessary.

Using IDELAY to Control Phase

If only minor phase adjustments are needed, you can use IDELAY or ODELAY instead of MMCM and PLL.

Using IDELAY or ODELAY adds additional delay, and increases the phase offset of the clock in relation to any associated data.

Using Gated Clocks

FPGA devices contain dedicated clock networks that can provide large fanout, low skew resources for clocking. Implying fine grained clock gating techniques in the HDL code can disrupt the functionality and mapping to this dedicated resource.

When coding to directly target an FPGA device, do not code clock gating constructs into the clock path. Instead, control clocking by using coding techniques to infer clock enables. This allows you to stop portions of the design for reasons of functionality or power.

If the code already contains clock gating constructs, or if it is intended for a different technology that requires such coding styles, use a synthesis tool that can remap gates placed in the clock path to clock enables in the data path. Doing so allows for a better mapping to the clocking resources, and simplifies the timing analysis of the circuit for data entering and exiting the gated domain.

If larger portions of the clock network might be shut down periods of time, the clock network can be enabled or disabled by using one of the following components:

- BUFGCE
- BUFHCE
- BUFR
- BUFMRCE

When a clock might be slowed down during periods of time, one of the following components can also be used with additional logic to periodically enable the clock net:

- BUFGCE
- BUFHCE
- BUFR

Alternatively, you can use a BUFGMUX to switch the clock source from a faster clock signal to a slower clock.

Reducing Dynamic Power

Any of the techniques discussed above can effectively reduce dynamic power. Depending on the requirements and clock topology, one technique might be more effective than another.

- BUFR
- BUFMRCE
- BUFHCE
- BUFGCE

BUFR

A BUFR is likely to be most effective when it meets the following criteria:

- The BUFR is an externally generated clock.
- The BUFR is under 200 MHz.
- The BUFR is required to source up to a maximum of three clock regions.

BUFMRCE

A BUFMRCE might be required for a Virtex-7 device in order to use this technique with more than one clock region (but only up to three vertically adjacent regions).

BUFHCE

A BUFHCE is better suited for higher speed clocks that can be contained in a single clock region.

BUFGCE

Although a BUFGCE can span the device, and is the most flexible, it is not the best choice for optimal power savings.

Output Clocks

Using an ODDR component is an effective way to forward a clock out of an FPGA device for external clocking devices.

To create a clock that is well controlled with respect to phase relationship and duty cycle:

1. Tie one input high.
2. Tie the other input low.

Use the set/reset and clock enable to stop the clock and hold it at a certain polarity for sustained amounts of time.

If further phase control is necessary for an external clock, use an MMCM or PLL with either or both of the following:

- External feedback compensation
- Phase compensation that is:
  - Coarse or fine grained
  - Fixed or variable

This gives control over clock phase and propagation times to other devices, thereby simplifying external timing requirements from the device.

Clock Domain Crossings

Because of process, voltage, and temperature (PVT) variations, there is more clock and phase uncertainty when going between two clock domains instead of staying in a single clock domain.

Going between two clock domains adds to both the setup and hold calculation, and can make timing more difficult even on slower speed designs.

To control the clock when different clock frequencies are required for different parts of the design:

1. Limit the number of clock domains.
2. Use general clock enables or one of the following components:
   - BUFGCE
   - BUFGMUX
   - BUFHCE
   - BUFR
   - BUFMRCE

If clocks are synchronous, or are not aligned at every edge, be sure that multi-cycle signals are properly transferred. For truly asynchronous signals (no known phase relationships), you must use special techniques to ensure proper data passing from one domain to another.

Synchronous Domain Crossings

Synchronous domains are domains in which:

- There is a known phase relationship between clocks.
- The phase relationship does not change from implementation run to implementation run.

This generally occurs when:

- The domains are derivatives of each other, or
- The domains are provided from the same internal or external source, or
- Both of the above are true.

In these cases, data can be analyzed, and, with some considerations, safely transferred from one domain to another.

Clock Skew

Depending on the common node and distance traveled in the buffers to reach source and destination, clock skew calculations can be significant.

For small data paths (such as register-to-register paths), the clock skew can be longer than the data delay. If this is not corrected, a hold time can result.

If there are several logic levels, the additional skew can make timing difficult. Xilinx recommends that you closely monitor logic levels in such cross clocking, and take into account the effect of too few or too many logic levels.

Synchronous Domain Crossing Examples

- Going from one BUFG network to another driven from the same MMCM, PLL, or device pin in which both BUFG components exist in the same half of the chip (top half or bottom half).
- Going from one BUFH network to another BUFH network that is placed horizontally adjacent.
- Going from one BUFR to another similarly configured BUFR both driven by the same BUFMR.

  If the BUFR components are not in BYPASS mode, they must also be phase aligned by synchronizing the reset on all associated BUFR components.
- Going to or from a BUFIO to or from a BUFR in the same clock region from the same clock source.

Asynchronous Domain Crossings

For asynchronous domain crossings, you must make a special effort to mitigate factors that can affect the data integrity for the design in such paths, including improper bus capture and metastability.

In general, two methods allow data to cross asynchronous clock domains safely.

- If only a single bit is required, or if methods such as gray coding are used to transfer more than one bit of related data, insert register synchronizers to reduce the Mean Time Before Failure (MTBF) of the circuit.
- For multiple bits of data (that is, a bus), use an independent clock (asynchronous) FIFO to transfer data from one domain to another.

Such a FIFO can be inferred if built from soft logic. However, if using the dedicated hard FIFO is desired or pre-characterized and pre-defined FIFO Logic makes the job easier, the FIFO can be directly instantiated in the design from the FIFO primitives or the CORE Generator™ FIFO Generator can be used to construct the FIFO.

Asynchronous Domain Crossing Examples

- Going from one clock network to another clock network that has no phase relationship.
- Going to or from one clock network using an MMCM or PLL to or from another network not using an MMCM or PLL.
- Going to or from a BUFH to any other network except the BUFH network horizontally adjacent to it.
- Going to or from a clock network that is not directly driven by a dedicated clocking resource (such as an external clock pin, MMCM, or PLL)
- Going to or from a BUFG located in the top half of the device to one located in the bottom half.

Controlling and Synchronizing Device Startup

Once the FPGA device completes configuration, the device comes out of the configuration state and into general operation.

In most configuration sequences, one of the final steps is:

1. The de-assertion of the Global Set Reset (GSR), followed by
2. The de-assertion of the Global Enable (GWE) signal.

At the moment of de-assertion, the design is in a known initial state, and is released for operation.

Unknown States

Portions of the design can go into an unknown state if the release point is not synchronized to the given clock domain, or if the clock is operating at a faster time than the GWE can be safely released.

For some designs, going into an unknown state is inconsequential. For other designs, going into an unknown state can cause the design to become unstable, or misprocess the initial data set.

Known States

When the design must come up in a known state, you must control the startup synchronization process.

Single Clock Domains

If the design has a single clock domain, the post-configuration sequence can be synchronized to the system clock and thus released synchronous to the design.

To synchronize the post-configuration sequence to the system clock:

1. Instantiate the STARTUP component.
2. Connect the system clock to the CLK pin.

This allows the configuration sequence to be driven by that common clock, thus easing synchronization of device startup.

Other Cases

In other cases, you can delay all design clocks until a period of time after GWE is asserted.

To delay the design clocks:

1. Instantiate the following components:
   - BUFGCE
   - BUFHCE
   - BUFR
2. Use the enables of those components to delay clocking a few clock cycles post-configuration.

For an MMCM, do so on the output clocks, but not on the feedback clock.

Alternative Methods

Alternative methods include using the following on critical parts of the design (such as a state machine):

- Clock enables
- Local resets

This ensures that the startup of those critical parts is controlled and known.

Using Clock Buffers for Non-Clock Nets

Clock buffers provide a uniform, low skew signal effective for clocking. When clock buffers are not required for general clocking needs, they can serve as additional routing resources for high fanout signals.

Take the following into account when choosing clock buffers for non-clock signals.

- Design Performance
- Using More Than Two BUFG Components or BUFH Components for Non-Clock Signals
- Using BUFG Components for Mixed Polarity Signals
- Using Enables Effectively
- Buffer Selection
- Specifying Buffer Placement

Design Performance

Global clock buffers are designed for low skew, not necessarily short propagation delay. Depending on desired clock frequency, and the nature of the timing path, a BUFG might not allow the design to meet timing.

Using More Than Two BUFG Components or BUFH Components for Non-Clock Signals

When BUFG drives non-clock signals, at certain points the signal must leave the global clock net to reach its destination.

- If *no more than two* (that is, one or two) BUFG components exit the global clock net at a single point, no contention is possible. The design can reliably route in any circumstance.
- If *more than two* (that is, three or more) BUFG components need to go to the same location, a conflict might not allow the design to route.

Xilinx recommends using no more than two (that is, one or two) BUFG components for non-clock signals. There are no routing conflicts to be resolved.

The same restriction applies to:

- Driving two BUFH components in the same clock region with non-clock signals.
- Driving a combination of BUFH components and BUFG components that must drive the same logic in the same clock region.

Using BUFG Components for Mixed Polarity Signals

Do not use BUFG components for:

- Mixed polarity signals
- Signals requiring additional logic for large portions of the net

For example, consider a high fanout reset signal that is:

- Active-High for some portions of the design
- Active-Low for other portions of the design

In this case, inserting a BUFG might not benefit, and might even hurt, the active-Low portions of the signal that require a LUT or other logic to perform the inversion.

Using Enables Effectively

To speed up the propagation delay when using BUFHCE:

1. Tie the input to logic **1**.
2. Use the enable to change the logic value of the net.

To do so, you must:

1. Set the **CE_TYPE** attribute to **ASYNC**.
2. Set **INIT_OUT** to **0**.

To encode an inversion into this path at no additional cost:

1. Ground the input.
2. Set **INIT_OUT** to **1**.

To accomplish the same goal with BUFGMUX, connect to the select (**S**) pin when:

- **CLK_SEL_TYPE** is set to **ASYNC.**
- Constants are placed on the inputs to the MUX.

Buffer Selection

Buffer selection can control logic grouping and placement with beneficial impact.

Using a BUFH or BUFR limits the connected components to a single clock region, and can lead to more optimal placement to meet timing on tight portions of the design.

Specifying Buffer Placement

In some cases, it might be necessary to specify the placement of the buffers in order to achieve the desired results, particularly on higher speed paths.

Clock Resource Selection Summary

This section summarizes clock resource selection for the following components:

- BUFG
- BUFGCE
- BUFGMUX and BUFGCTRL
- BUFH
- BUFG
- BUFHCE
- BUFR
- BUFIO
- BUFMR
- BUFMRCE
- MMCM
- PLL
- IDELAY and IODELAY
- ODDR

BUFG

Use BUFG when a high fanout clock must be provided to several clock regions throughout the device.

Use BUFG when you do not want to instantiate or manually control clocking.

Use BUFG for very high fanout non-clock nets such as a global reset for medium to slower speed clocks in which mixed polarities do not exist. Xilinx recommends that you limit this use to only two in any design.

For SSI devices in which clocks must span more than one SLR, locate them in a center SLR. This more evenly distributes the clocking net across the entire device, thereby minimizing skew.

BUFGCE

Use BUFGCE to stop a large fanout several-region clock domain.

BUFGMUX and BUFGCTRL

Use BUFGMUX and BUFGCTRL to change clock frequencies or clock sources.

BUFH

Use BUFH for smaller clock domains of logic that can be contained in a single clock region.

Use BUFH for very high speed clocking domains.

Use BUFH in clock domains that are less likely to compete for clocking resources with BUFG components.

BUFG

For SSI devices, use BUFG in the upper or lower SLR components. This reduces the chances of competition for resources with the BUFG components in the center SLR.

BUFHCE

Use BUFHCE (when driven by a BUFG) to stop a medium grained portion of the clock network in which can be placed into a single clock region

Use BUFHCE for high fanout non-clock signals (such as a reset) that can be contained in a single clock region.

BUFR

Use BUFR for small to medium sized clock networks that do not require performance higher than 200 MHz.

Use BUFR for externally provided clocks that can be constrained in up to three vertically adjacent clock regions that require clock division.

For SSI devices, use BUFR in the upper or lower SLR components. This reduces the chances of competition for resources with the BUFG components placed into the center SLR components.

BUFIO

Use BUFIO for externally provided high speed I/O clocking generally in source synchronous data capture.

BUFMR

BUFMR applies to Xilinx 7 series FPGA devices only.

Use BUFMR when you need to use BUFR or BUFIO components in more than one vertically adjacent clock regions for a single clock source.

For SSI devices, locate the BUFMR and associated pin into the center clock region in an SLR. This allows access to all three clock regions from the BUFMR in case it is required.

BUFMRCE

BUFMRCE applies to Xilinx 7 series FPGA devices only.

Use BUFMRCE when you need to use BUFR or BUFIO components in more than one vertically adjacent clock regions for a single clock source when the clock is desired to be periodically stopped.

Use BUFMRCE if you are using more than one BUFR in which clock division is used. The BUFMRCE can be used to ensure proper phase startup of all connected BUFR components.

MMCM

Use MMCM to remove the clock insertion delay (phase align the clock to the incoming data) for system synchronous inputs and outputs.

Use MMCM for clock phase control to align source synchronous data to the clock for proper data capture.

Use MMCM to change the clock frequency or duty cycle of an incoming clock.

Use MMCM to filter clock jitter.

PLL

Use PLL for clock phase alignment of high speed incoming clocks

IDELAY and IODELAY

Use IDELAY and IODELAY on an input clock to add small amounts of additional phase offset (delay).

Use IDELAY and IODELAY on input data to add additional delay to data. This effectively reduces clock phase offset in relation to the data.

ODDR

Use ODDR to create an external forwarded clock from the device.

Appendix A

Additional Resources

Xilinx Resources

- **Device User Guides**:
  http://www.xilinx.com/support/documentation/user_guides.htm
- **Xilinx Glossary**: http://www.xilinx.com/company/terms.htm
- ***Xilinx Design Tools: Installation and Licensing Guide*** (UG798):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/iil.pdf
- ***Xilinx Design Tools: Release Notes Guide*** (UG631):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/irn.pdf
- **Product Support and Documentation:** http://www.xilinx.com/support

Hardware Documentation

- **7 Series Device Documentation**:
  http://www.xilinx.com/support/documentation/7_series.htm

ISE Documentation

- **Libraries Guides**:
  http://www.xilinx.com/support/documentation/dt_ise14-3_librariesguides.htm
- **ISE Design Suite Documentation**:
  http://www.xilinx.com/support/documentation/dt_ise14-3.htm
  - ***Command Line Tools User Guide*** (UG628):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/devref.pdf
  - ***Constraints Guide*** (UG625):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/cgd.pdf
  - ***Data2MEM User Guide*** (UG658):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/data2mem.pdf
  - ***ISim User Guide*** (UG660):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/plugin_ism.pdf
  - ***Synthesis and Simulation Design Guide*** (UG626):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/sim.pdf
  - ***Timing Closure User Guide*** (UG612):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug612.pdf

- *Power Methodology Guide* (UG786):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug786_PowerMethodology.pdf
- *Xilinx/Cadence PCB Guide* (UG629):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/cadence_pcb.pdf
- *Xilinx/Mentor Graphics PCB Guide* (UG630):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/mentor_pcb.pdf
- *XPower Estimator User Guide* (UG440):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug440.pdf
- *XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices* (UG627):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/xst.pdf
- *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices* (UG687):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/xst_v6s6.pdf

Partial Reconfiguration Documentation

- **Partial Reconfiguration website**: www.xilinx.com/tools/partial-reconfiguration.htm
- *Partial Reconfiguration User Guide* (UG702):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug702.pdf

PlanAhead Documentation

- **PlanAhead User Guides**:
  http://www.xilinx.com/support/documentation/dt_planahead_planahead/14-3_userguides.htm
  - *Floorplanning Methodology Guide* (UG633):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/Floorplanning_Methodolgy_Guide.pdf
  - *Hierarchical Design Methodology Guide* (UG748):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/Hierarchical_Design_Methodolgy_Guide.pdf
  - *Pin Planning Methodology Guide* (UG792):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug792_pinplan.pdf
  - *PlanAhead Tcl Command Reference Guide* (UG789):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug789_pa_tcl_commands.pdf
  - *PlanAhead User Guide* (UG632):
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/PlanAhead_UserGuide.pdf
  - **PlanAhead Tutorials:**
    http://www.xilinx.com/support/documentation/dt_planahead_planahead14-3_tutorials.htm