7 Series FPGAs Migration

Methodology Guide

UG429 (v1.2) April 04, 2018

XILINX
ALL PROGRAMMABLE™

Revision History

Date	Version	Revision
04/04/2018	1.2	Updated PCIE_2_0 in Chapter 2. Updated STARTUP_VIRTEX6 in Chapter 2
10/15/14	1.1	Added paragraph on clocking components to the end of the Clocking Considerations section in Chapter 2. Updated description of FIFO18E1/FIFO36E1 in Chapter 2.
03/17/11	1.0	Initial Xilinx release.

Table of Contents

Targeting/Retargeting Considerations for 7 Series Devices

This chapter covers topics associated with prior FPGA design implementations that are not technology specific. The concepts in this chapter should be considered when either evaluating existing design source from prior FPGA or ASIC targets or when developing new code for use in the 7 series devices.

HDL Coding Considerations

Use of Control Signals

The use of control signals (signals that control synchronous elements such as clock, set, reset, and clock enable) can impact device density, utilization, and performance. This is true in almost any FPGA technology. However, the difference in the topology of the 7 series devices should be considered in selecting and using control signals. These considerations are necessary to achieve the best device utilization and performance.

Avoid Use of Both a Set and a Reset on a Register or Latch

As with the Virtex®-6 and Spartan®-6 architectures, 7 series devices do not have a REV pin. As a result, flip-flops cannot natively implement both a set signal and a reset signal without the use of additional logic. Thus, when using both a synchronous set and reset, an additional signal is added to the datapath, which might affect area and timing depending on placement, fanout, and timing considerations. In some cases, that additional signal can be absorbed without increasing logic levels and, in that case, has little net effect on the design. However, for an asynchronous set and reset, the effect on resource utilization and timing is more significant and should be avoided.

Registers that contain both asynchronous reset and asynchronous set signals and/or with an asynchronous control signal with a dynamic value can be implemented. The resulting circuit might consume more resources than desired and might have more significant effect on timing and verification than originally thought.

This configuration can be described in RTL or be instantiated with an FDCPE in HDL, EDIF, or NGC formats. This simple coding example consists of Verilog code describing asynchronous set and reset resulting in additional resources and timing paths:

```
always @(posedge reset, posedge set, posedge clk)
  if (reset)
    a_reg <= 1'b0;
  else if (set)
    a_reg <= 1'b1;
  else
    a_reg <= A;
```

This second coding example consists of VHDL code describing an asynchronous control signal with a dynamic value resulting in additional resources and timing paths:

```
process (clk, initc) begin
   if initc='1' then
      data_reg <= init_signal;
   elsif (clk'event and clk='1') then
      data_reg <= data_in;
   end if;
end process;
```

When the software encounters these configurations, it issues a warning message describing the issue and lists the corresponding registers. When using Xilinx Synthesis Technology (XST) for synthesis, this warning is issued:

```
WARNING:Xst:3001 - This design contains one or more registers or latches with an
active asynchronous set and asynchronous reset. While this circuit can be built, it
creates a sub-optimal implementation in terms of area, power and performance. For a
more optimal implementation Xilinx highly recommends one of these:
   1) Remove either the set or reset from all registers and latches if
   not needed for required functionality
   2) Modify the code in order to produce a synchronous set and/or reset
   (both is preferred)
   3) Use the -async_to_sync option to transform the asynchronous
   set/reset to synchronous operation (timing simulation highly
   recommended when using this option)

Please refer to https://www.xilinx.com search string "Virtex7 asynchronous
set/reset" for more details.

List of register instances with asynchronous set and reset:
My_async_reg in unit <async_set_and_reset_module_or_entity>
```

If a third-party synthesis is used, a similar warning might also be seen. If the design contains a netlist or instantiated FDCPE component, this message might be seen in Map:

```
WARNING: MapLib:1182 - One or more latches or registers which have both an active
asynchronous set and reset have been found in the design. To get this same
functionality in the virtex7 architecture a sub-optimal circuit must be created in
terms of area, power and performance. Therefore it is highly suggested to either
remove one set or reset or make the function synchronous in order to obtain a more
optimal implementation in this architecture.
```

```
List of register instances with set and reset:
  My_async_reg
```

If any of these warnings are encountered, it is suggested to evaluate the code for changes that can eliminate the need for describing both a asynchronous set and reset condition.

Register Initialization

Many engineers use the inherent initialization of registers and latches in the FPGA via the global set/reset (GSR) signal by implicitly specifying initialization of an inferred register, thus creating a more robust and sometimes smaller circuit. With initialization, a different start-up state from reset state is allowed, which in some cases (such as a state machine with certain states that must be run at start-up but perhaps not on a subsequent reset) it consumes less logic than an uninitialized state yields. Initialization also allows for the RTL description to more closely and accurately behave as the actual FPGA, creating a more accurate representation of the circuit. For this reason it is suggested to use register initialization on any inferred register, SRL, or RAM when possible.

In this code example, the reg register is initialized with the value of 1:

```
signal reg: std_logic := '1';
...
process (clk) begin
  if (clk'event and clk='1') then
    if (rst='1') then
      reg <= '0';
    else
      reg <= val;
    end if;
  end if;
end process;
```

In the coding example, the use of initialization eliminates the need to specify a set condition for the sole purpose of creating an initial condition of a logic one. Even if the initial condition matches the reset condition, it is still suggested to specify register initialization to allow the simulation start-up state to more accurately reflect the initial condition of the FPGA without requiring a reset condition.

Limit Use of Active-Low Control Signals

It is not recommended to use inferred or instantiated components with active-Low control signals due to the combination of:

- The 7 series device's coarse slice composition (8 registers per slice).
- The absence of a programmable inversion element on the slice control signals in the 7 series device.
- Hierarchical design methods that do not allow optimization across hierarchical boundaries, such as the use of KEEP_HIERARCHY, partitions, partial reconfiguration, or bottom-up synthesis.

In certain situations, device utilization can increase due to the use of a LUT as an inverter and the additional restrictions of register packing sometimes caused by active-Low control signals.

Timing can also be affected by the use of active-Low control signals. Active-High control signals should be used wherever possible in the HDL code or instantiated components. When a control signal's polarity cannot be controlled within the design (such as when it is driven by an external, non-programmable source), the signal in the top-level hierarchy of the code should be inverted, and active-High control signals driven by the inverter to get the same polarity (functionality) should be described. When described in this manner, the inverter can be absorbed into the I/O logic, without using any additional logic or routing, resulting in better utilization, performance, and power.

Limit Use of Low Fanout Control Signals

The number of unique control signals in the design should be limited to those necessary for design functionality and operation. Low fanout, unique control signals can result in underutilized slices in terms of registers, SRLs, and LUT RAM. These control signals can have negative impacts on placement and timing. In general, a set, reset, or clock enable should not be implemented in the code unless it is required for the active functionality of the design.

Unnecessary Use of Sets or Resets

Unnecessary sets and resets in the code can prevent the inference of SRLs, RAMs (LUT RAMs or block RAMs), and other logic structures that are otherwise possible. To get the most efficiency out of the architecture, sets and resets should only be coded when they are necessary for the active functionality of the design.

Sets and resets should not be coded when they are not required. For example, a reset is not required when it is only used for initialization of the register, because register initialization occurs automatically upon completion of configuration.

Another example where a reset is not required is when a circuit remains idle for long periods. A simple reset on the input registers eventually flushes out the data on the rest of the circuit.

A third example is with inner registers when the reset is held for multiple clock cycles. In this case, the inner registers are flushed during reset, so the reset is not necessary on all registers. By reducing the use of unnecessary sets or resets, greater device utilization, better placement, improved performance, and reduced power can be achieved.

Sets for Multipliers or Adders/Subtractors in DSP48E1 Slice Registers

7 series DSP48E1 slice registers contain only resets and not sets. The 7 series DSP blocks can perform many different functions, including multiplication, addition/subtraction, comparators, counters, and general logic. To allow the flexibility of use of this additional

resource in the design, a set condition cannot exist in the function for it to properly map to this resource. Thus, unless necessary, a set (value equals logic 1 upon an applied signal) should not be coded around multipliers, adders, counters, or other logic that can be implemented within a DSP48E1 slice.

Use of Synchronous Sets/Resets

If a set or reset is necessary for the proper operation of the circuit, a synchronous reset should always be coded. Synchronous sets/resets have improved timing characteristics and stability and can also result in smaller, better utilization within the FPGA.

Synchronous sets/resets can result in less logic (fewer LUTs), fewer restrictions on packing, and, often, faster circuits. The registers within blocks like the DSP48E1 and RAMB36E1/RAMB18E1 cannot implement an asynchronous set and reset. Thus they cannot be used with functional equivalency if an asynchronous set or reset exists in the RTL code. With synchronous resets, far more flexibility is provided to the sharing of registers within a slice. Each slice has a shared reset and registers with incompatible synchronous resets can be remapped to the datapath, allowing them to be placed into the same slice. Asynchronous resets cannot be remapped and retain the same functionality. Thus two registers with different asynchronous resets or a register with an asynchronous reset and one without cannot be mapped into the same slice. This not only affects device density but also can have negative effects on performance and power because placement can become suboptimal and require more routing delay and capacitance.

If you do not want to recode existing asynchronous resets to synchronous resets, the asynchronous resets can be treated as synchronous resets by using the Asynchronous To Synchronous switch, if available, in the synthesis tool. Use of this switch though adds the risk that the RTL hardware description might not act exactly like the implemented design under all reset conditions. Thus extra care and effort might be needed during circuit verification if the switch is used.

If XST is the synthesis tool, the Asynchronous To Synchronous switch is available in the ISE® software Project Navigator Properties for XST, or the **`-async_to_sync`** switch can be used as a synthesis option when used by the command line. This option is not as effective as recoding to use a synchronous set/reset in terms of reducing resources and improving performance. However, it does allow for additional register packing and optimizations, which is not possible otherwise, resulting in a smaller and sometimes faster circuit compared to not using this option at all.

Use of Clock Enables

High fanout clock enables should not be manually split or replicated but coded as a single clock enable. If replication becomes necessary for timing or other reasons, it should be controlled within the synthesis tool. This allows for greater flexibility to control replication many times, giving a better trade-off balance between additional resources and power for improved performance. Also as placement and other factors change, the replication requirements might also change. Having control of this from the synthesis and

implementation tools means that the original code does not need to be modified and reverified for such changes.

Another benefit to retaining high-fanout clock enables to a single net is it allows for simpler remapping to other dedicated resources like a BUFGCE or BUFHCE. These can realize the same functionality with the added benefit of an even greater reduction in both power and consumption of routing resources.

Use of DSP and Other Arithmetic Intensive Code

Many DSP designs are well suited for the 7 series architecture. To obtain best use of the architecture, the underlying features and capabilities need to be understood so that design entry code can take advantage of these resources.

The DSP48E1 blocks use a signed arithmetic implementation. It is suggested to code using signed values in the HDL source to best match the resource capabilities and, in general, get the most efficient mapping. If unsigned bus values are used in the code, the synthesis tools should still be able to use this resource but might not get the full bit precision of the component due to the unsigned-to-signed conversion. The multiplier within the 7 series DSP48E1 slice has an input bit precision of 18 bits by 25 bits signed data. Thus the bit precision for unsigned data is 17 bits by 24 bits. For Verilog code, data is considered unsigned unless otherwise declared in the code.

If the target design is expected to contain a large number of adders, it is suggested to evaluate the design to make greater use of the DSP48E1 slice pre- and post-adders. For example, with FIR filters, the adder cascade can be used to build a systolic filter rather than using multiple successive add functions (adder trees). If the filter is symmetric, you can evaluate using the dedicated pre-adder to further consolidate the function into both fewer LUTs and flip-flops and also fewer DSP slices as well (in most cases, half the resources).

If adder trees are necessary, the 6-input LUT architecture can efficiently create ternary addition (A + B + C = D) using the same amount of resources as a simple 2-input addition. This can help save and conserve carry logic resources, when needed. In many cases, there is no need to use these techniques. By knowing these capabilities, the proper trade-offs can be acknowledged up front and accounted for in the RTL code to allow for a smoother and more efficient implementation from the start.

In most cases, DSP resources should be inferred. If XST is used for synthesis, it is suggested to consult the "XST Hardware Description Language (HDL) Coding Techniques" chapter of UG627, *XST User Guide*. It is also suggested to read UG479, *7 Series FPGAs DSP48E1 Slice User Guide*, for the features and capabilities of the DSP48E1 slice and how to best leverage this resource for one's design needs.

RAM Considerations

To maximize the use of block RAMs and LUTs in the 7 series architecture, certain considerations must be understood when re-targeting block RAM and LUT RAM by inference, instantiated primitive, Unimacro, or Vivado® Design Suite software. If the Vivado tool is used for RAM generation, the IP should be regenerated for the 7 series device, or the RAM should be recoded for proper synthesis inference.

Either method often gives good results for utilization and performance. However, it is recommended to infer memory where possible to improve understanding of the code, simulation, and future portability of the code.

Instantiating RAMs

The recommendations in this section are for cases in which RAM primitives are instantiated in the design or when it is not possible to regenerate Vivado® Design Suite software IP for 7 series devices. These suggestions should also be implemented by code that infers RAM, especially when using synthesis attributes to guide which RAM resources are used (such as syn_ramstyle for Synplicity or RAM_STYLE for XST). The suggestions are divided by RAM depth, the most important factor in determining which RAM resource to use.

Depths Less Than 128 Bits

Due to the large LUTs and deep LUT RAMs in 7 series FPGAs, the criteria for choosing between a block RAM and LUT RAM might have changed from previous FPGA generations. In general, a LUT RAM should be used for all memories that consist of 64 bits or less, unless there is a shortage of logic resources (LUTs) and/or SLICEMs for the target device.

Using LUT RAMs for memory depths of 64 bits or less, regardless of data width, is more efficient in terms of resources, performance, and power. For depths greater than 64 bits but less than or equal to 128 bits, the decision on the best resource to use depends on these factors:

1. The availability of extra block RAMs. If not available, LUT RAM should be used.
2. The latency requirements. If asynchronous read capability is needed, LUT RAMs must be used.
3. The data width. Widths greater than 16 bits should use block RAM, if available.
4. The necessary performance requirements. Registered LUT RAMs generally have shorter clock-to-out timing and fewer placement restrictions than block RAMs. If the design already contains instantiated LUT RAMs with depths greater than 16 bits, the deeper primitive (for example, RAM32X1S or RAM64X1S) should be used.

RAM16X1S components, used in conjunction with MUXF5 components or other logic, are not properly re-targeted to automatically use the greater depth LUT. In such cases, the code should be modified to properly use the deeper primitives.

Depths Greater Than 128 Bits

In most cases, depths greater than 128 bits should target block RAM. There are two types of block RAM in the 7 series devices: an 18 Kb RAM (RAMB18E1) and a 36 Kb RAM (RAMB36E1). The choice between these RAMs is generally dictated by the desired width and depth. Table 1-1 shows the RAMB18 and RAMB36 width/depth combinations in True Dual-Port (TDP) or Simple Dual-Port (SDP) configurations.

Table 1-1: **Guide for Block RAM Primitive Selection Based on Memory Width, Depth, and Type (True-Dual Port vs. Simple-Dual Port)**

	Memory Width (Bits)			
	RAMB18 TDP	**RAMB18 SDP**	**RAMB36 TDP**	**RAMB36 SDP**
512 Depth	18 bits or less	36 bits or less	19 bits to 36 bits	37 bits to 72 bits
1K Depth	18 bits or less	18 bits or less	19 bits to 36 bits	19 bits to 36 bits
2K Depth	9 bits or less	9 bits or less	10 bits to 18 bits	10 bits to 18 bits
4K Depth	4 bits or less	4 bits or less	5 bits to 9 bits	5 bits to 9 bits
8K Depth	2 bits or less	2 bits or less	3 bits to 4 bits	3 bits to 4 bits
16K Depth	1 bit	1 bit	2 bits	2 bits
32K Depth	N/A	N/A	1 bit	1 bit
64K Depth	N/A	N/A	1 bit(1)	1 bit(1)

Notes:
1. Requires two RAMB36 components configured in cascade mode.

Assess Additional RAM Features

Good design practices include determining whether any dedicated RAM features should be used when starting a new design project. Some features to consider are:

- FIFO: The 7 series device block RAM contains dedicated logic to implement synchronous (same clock) or dual-clock FIFO buffers with features such as first word fall-through and programmable threshold almost empty and almost full flags. Designs containing FIFOs created from soft logic should consider using this dedicated logic to improve device utilization, power, and performance as well as ease the overall design of these components.

- ECC Logic: The 7 series block RAM also contains dedicated logic for RAM content error detection and correction. This feature should be considered for designs requiring such data error correction.
- Output Registers: Use of the output register can significantly improve performance (clock to out) of the block RAM, while also improving power and device utilization. If a design is ported into the 7 series architecture from a prior architecture, the code should be re-examined to see if the output register can be incorporated into the design.
- Byte Wide Write Enables: 7 series devices have byte-wide write enables. This feature can be beneficial to the block RAM access and utilization for the device. In some cases, more efficient use of block RAMs and other resources can be seen with the use of this feature.
- Enable/Reset Priority. 7 series FPGAs can change the priority of enables versus resets, allowing for greater consistency of output register control to that of slices and I/O registers.

More information can be obtained from the *7 Series FPGAs Memory Resources User Guide*.

Use of Synthesis and Physical Constraints

Many times, synthesis attributes, constraints, and directives are embedded in the code or synthesis constraints file to create a desired result in a prior implementation or architecture. It is suggested to comment out or remove these elements because they might lead to an inferior result and not be the best choice in future implementations.

Any LOCs, RLOCs, BEL constraints, or other physical constraints, embedded in the code, netlist, or UCF file of the existing design should be removed before re-targeting to a 7 series FPGA. An optimal placement for an older architecture is likely not optimal in a 7 series FPGA due to differences in the functional blocks, device floorplan, and timing. In some cases, errors can occur due to layout and coordinate differences. However, even if no errors occur, timing, density, and power can be suboptimal unless the physical constraints are modified, removed or updated for the new architecture.

Specification of Timing Constraints

First, synthesis timing constraints should be specified that relate realistic timing objectives. The synthesis software can apply area-saving algorithms where performance objectives can still be met in areas with excess timing slack. Timing optimization algorithms can be applied in areas with tight timing slack. Without timing constraints, the synthesis tools must optimize all parts of the design for timing, often at the expense of area. This can also lead to optimization (logic level reduction) in areas that do not need it at the expense of paths that could be further optimized. Timing constraints allow focus on the areas of design that need it and relaxation on those that do not.

Implementation timing constraints should also be applied that reflect both the desired I/O and clock timing but also the timing exceptions, such as multicycle paths and false paths or timing ignores (TIGs). Applying realistic and complete timing constraints can often result not only in improved results but also shorten timing closure and debug as well as reduce run time and memory requirements as well. It is not recommended to over constrain the design because it can lead to longer run times, more memory, and, in some cases, worse results than a precisely constrained design. Time spent to create a good UCF file can save far greater time in the overall timing closure process.

Software Options

Software algorithms for the 7 series FPGAs are designed to deliver a balance between device area (and thus cost), power, and performance. Options in the ISE tools allow designers to improve device area at the cost of performance or improve performance at the cost of device area. There are also options to reduce power that can often result in trade-offs in performance, area, and/or software run time. Options in the software can be specified to achieve design goals when the default balanced approach does not. It is not suggested though to use these options until first it is determined that the design goals can be met with the default algorithms. For this reason, it is suggested that the first design runs are done with the default options used for the software. Then after analysis of those results, specific options can be used to help tune the algorithms to gain the results needed.

Use of LUTs as Route-Thrus

When analyzing 7 series FPGA LUT utilization, the use of LUTs as route-thrus must be considered. LUT route-thrus in the Map report are created when access is needed to internal slice logic or registers when no other input path is available into the slice, most commonly when the bypass inputs (AX, BX, CX, or DX) are not available. A LUT route-thru uses a single input to the LUT to obtain access into the slice. A few situations can cause this:

1. A flip-flop, RAM, or other non-LUT source drives a flip-flop (where bypass lines are occupied, generally because four registers already exist in the slice).
2. A flip-flop, RAM, or other non-LUT source drives the MUXF7/MUXF8 data inputs within the slice.
3. A flip-flop, RAM, or other non-LUT source drives a select or data line of CARRY4 (select line of MUXCY and/or DI of XORCY).

7 series devices have eight registers within a slice, beneficial for both performance and area. For many high-speed, highly pipelined designs, these additional registers can allow for maximum performance without the cost of additional logic. They also allow for improved LUT combinations to utilize the dual output structure of the 6-input LUT without

loss of performance. They also allow for more efficient shift-register implementations when a shift register LUT (SRL) is not possible due to the use of a reset or the need for many tap points out of the shift register.

Having eight registers often allows fewer slices to be consumed for registers driven by non-LUT sources (such as block RAM, another flip-flop, or a DSP slice). The Xilinx software tools are tuned to place these slice registers to yield the best characteristics in terms of performance, area, and power. However, when more than four registers are placed into a single slice, a LUT route-thru is often necessary and thus reported in the resource utilization.

In a given design, more LUTs can be reported as being used in 7 series devices; however, fewer slices are required because of the route-thrus. This can be misleading because it is common to consider LUT usage and not slice usage for utilization comparisons. For designers concerned with how much logic is left, the route-thru consumes one input (generally not the A6 input, but one of the lower inputs to get access to the O5 output) and one output of the LUT, leaving four inputs and one output for any given function. Thus, even when a needed route-thru is used, the logic equivalent of a 4-input LUT remains unused.

In a typical situation, the registers can be distributed over more slices, if necessary, and not consume any route-thrus. Thus, designers get the entire 6-input LUT for logic but might need to consume more slices or possibly see performance penalties for the spreading of these registers. Route-thrus are reported as consuming an entire LUT when in fact they only consume a partial LUT, or in a different placement might not consume a LUT at all. Here is a portion of the resource reporting in the Map (`.mrp`) file that reports route-thrus and their purposes:

```
Number used exclusively as route-thrus:    117
  Number with same-slice register load:    106
  Number with same-slice carry load:        11
  Number with other load:                    0
```

This section of the report indicates the number of LUTs needed for route-thrus and the reason. This is only needed for informational purposes and can be used to further assess the device utilization, if needed.

Figure 1-1 shows an FPGA Editor view of the use of a route-thru for gaining access to all eight registers within a slice.

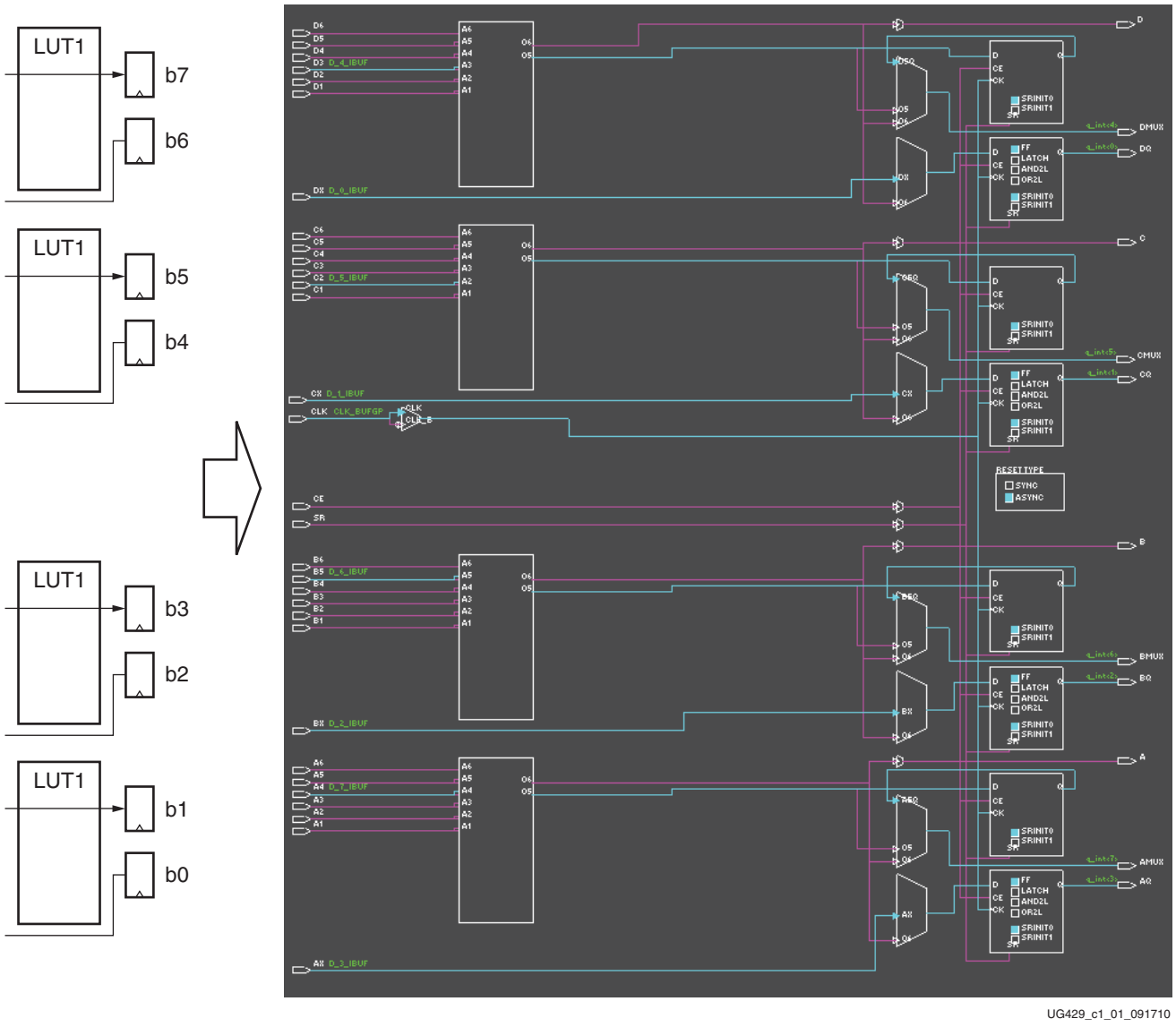

Figure 1-1: **FPGA Editor Representation of a Route-Thru**

Virtex-6 FPGA Re-Targeting Considerations

Introduction

This chapter covers information specific to Virtex®-6 FPGA designs that are migrating to 7 series devices.

7 Series Device Selection

The 7 series architecture is similar to the Virtex-6 family in terms of resources and device size comparisons. The DSP, block RAM, and slice structures have remained relatively the same, meaning the same types of design structures and code should target the same type and amount of resources in Virtex-7 devices. As with Virtex-6 FPGAs, the particular device number refers to the number of thousand logic cells a particular array contains; i.e., the Virtex-6 `XC6VLX550T` device contains approximately 550,000 logic cells. The same metric and convention remains for 7 series devices, meaning a slightly larger size device would be the Virtex-7 `XC7V585T` device, which contains approximately 585,000 logic cells. This simplifies comparison based on logic density between families. For Virtex-6 devices smaller than the `XC6VLX240`, the Kintex®-7 family can be a better fit because Virtex-7 devices start at 285,000 logic cells.

Because logic density is not the only factor in determining device selection, all other needed resources should also be evaluated. It is suggested initially to compare all relevant resources in the Feature Summary tables of *7 Series FPGAs Overview* (DS180) [Ref 3]. Here are some additional considerations to help determine the 7 series device selection:

- Block RAM/DSP. For designs requiring heavy block RAM or DSP use, such as in Virtex-6 SXT devices, Virtex-7 extended capability (`XC7VXT`) devices have higher block RAM and DSP content for a given array size.
- I/O. The number of I/Os for a given array size, the bank sizes, arrangement, and capabilities differ from the Virtex-6 family. At the start of a new design, it is worth considering if more functionality can be folded into the device to not only limit needed I/Os but also improve overall system performance, cost, power, and capability.
- Speed Grade. It is recommended to select the same speed grade as was used to target the Virtex-6 architecture. While additional performance could be seen in some blocks

or areas of the design, overall system performance should be assumed to be relatively the same until the implemented design can be analyzed for its exact performance characteristics.

Use of Existing Soft IP, EDIF, or NGC Netlists

It is highly recommended to regenerate or re-synthesize any old soft IP or black box netlists in the design prior to implementation in the 7 series device. Most netlists targeting the Virtex-6 architectures can be implemented without error when targeting 7 series devices; however, it is suggested to regenerate any netlist or core targeting prior architectures to ensure the 7 series device is most efficiently targeted.

Clocking Considerations

The 7 series architecture has a similar clock structure to the Virtex-6 architecture. Both architectures have 32 global buffers (BUFGs), regional buffers (BUFIOs and BUFHs), and I/O buffers (BUFIO). Both architectures have mixed-mode clock managers (MMCMs), clock gating (BUFGCE and BUFHCE), and clock muxing (BUFGCTRL). In many cases, clocking topologies do not need to be updated to transition from Virtex-6 to 7 series devices. However, some differences might require changes and some additional features in the 7 series architecture should be evaluated to find out if the overall design characteristics could be improved.

Virtex-6 FPGA designs utilizing BUFGs for the clocking connectivity should not need updating. As with Virtex-6 FPGAs, 7 series devices have 32 global buffer components and similar connectivity. The limit of using 12 BUFGs in any clock region is the same; however, the 7 series clock region is bigger than the Virtex-6 FPGA clock region. This is not expected to cause issues but in some cases might pose a different placement within a clock region when re-targeting to the 7 series device. If using the clock-enable or clock-gating capabilities of the global clocking via a BUFGCE, or clock muxing via the BUFGCTRL or BUFGMUX, the functionality and use are the same between architectures. The use of horizontal clocking buffers (BUFHs or BUFHCEs) is also unchanged between architectures.

When using a BUFR, the 7 series buffer functionality is the same as with Virtex-6 FPGAs. The primary difference is the BUFR in Virtex-6 FPGAs can drive adjacent regions, whereas in 7 series devices, it can only drive the region in which it exists. If the BUFR was driving multiple clock regions, it is suggested to re-evaluate this clocking structure. Because the 7 series devices have larger clocking regions than in Virtex-6 FPGAs, it is possible that what drove two clock regions in a Virtex-6 device is a better fit for a single clock region in a 7 series device, in which case, no design changes are necessary. If it is determined that multiple regional clocking is still desired when targeting the 7 series device, then design changes are necessary. It is possible to drive multiple regions with a regional clock;

however, to do this in 7 series devices, a BUFMR can drive BUFRs in up to three vertically adjacent clock regions.

Similar to BUFRs, when using BUFIOs to drive a single bank, the functionality is the same as with Virtex-6 devices. The connectivity of BUFIOs in 7 series devices is limited to a single bank. If it is desired to drive multiple banks, it is possible to drive multiple BUFIOs in adjacent banks through a BUFMR. The BUFMR can drive both BUFIOs and BUFRs to allow for a high-speed I/O clock and a phase-aligned, slower divided clock in multiple adjacent clock regions.

If using an MMCM in the Virtex-6 FPGA design, in most cases, the MMCM can be retargeted and used the same in the 7 series design. If the DRP port is used, the mapping and data to change programming aspects might have changed and thus likely needs modification to perform the same purpose. If the Virtex-6 FPGA design incorporates directly cascaded MMCMs, this connectivity no longer exists and is not directly supported. In all other cases, the MMCM should be able to be automatically re-targeted and used in the same way as in Virtex-6 devices.

Some new clocking capabilities in the 7 series device should be evaluated. There is a new clocking component called the PLLE2. Similar to the MMCM, this functionality is represented by two UNISIM components, PLLE2_BASE and PLLE2_ADV. PLLE2_BASE can be used for most circumstances, and PLLE2_ADV is used in more advanced instances. In most cases, the MMCM is used because it has more capabilities than the PLLE2; however, for high-speed I/O clocking, the PLL might be preferred. The BUFHCE component in 7 series devices allows asynchronous gating of the enable. This can be useful when using a BUFH to drive high-fanout non-clock signals or in cases where the clock can be stopped or lost. A new attribute named CE_TYPE is added to the 7 series BUFH component.

- When this attribute value is SYNC, the behavior is the same as in Virtex-6 devices, where the clock can be stopped synchronously without the potential of clock glitches on the output.
- When the new attribute value is ASYNC, a new behavior is invoked, where the clock is gated or ungated, regardless of clock source input.

The functionality of clocking components in 7 series devices and Virtex-6 devices is similar. The performance parameters, however, are different. Consult the appropriate 7 series datasheet to confirm the performance parameters meet the design requirements

Driving Non-Clock Loads with Global Buffers

It is not uncommon to place high fanout signals onto global buffers (BUFGs) to reduce local routing requirements. This practice can still be done with 7 series devices. As with Virtex-6 FPGAs, it is suggested to limit the number of BUFGs used for non-clock purposes to two or fewer to avoid conflicts.

Other Primitive Retargeting Considerations

These primitives are identical between Virtex-6 and 7 series devices. There should be no changes in behavior, connectivity, or general use:

```
AND2B1L, BUFG, BUFGCE, BUFGCTRL, BUFGMUX, BUFGMUX_1,
BUFGMUX_CTRL, BUFGP, BUFH, BUFHCE, BUFIO, CARRY4, CFGLUT5,
DCIRESET, DNA_PORT, DSP48E1, EFUSE_USR, FDCE, FDPE, FDRE, FDSE,
IBUF, IBUFDS, IBUFDS_DIFF_OUT, IDDR, IDDR_2CLK, IDELAYCTRL,
IOBUF, IOBUFDS, KEEPER, LDCE, LDPE, LUT1, LUT2, LUT3, LUT4, LUT5,
LUT6, MUXF7, MUXF8, OBUF, OBUFDS, OBUFTDS, ODDR, OR2L, PULLDOWN,
PULLUP, RAM128X1D, RAM256X1S, RAM32M, RAM32X1S, RAM64M, RAM64X1D,
RAM64X1S, SRL16E, SRLC32E
```

Some device primitives instantiated for the Virtex-6 architecture are not automatically re-targeted to the 7 series architecture, or they are re-targeted with some notable differences. See *Xilinx 7 Series FPGA Libraries Guide for HDL Designs* (UG768) [Ref 3] for more details on any UNISIM component and the respective User Guide for specific details about any of these blocks when targeting 7 series FPGAs.

This section highlights a few components.

BSCAN_VIRTEX6

The Boundary Scan component for 7 series devices has the same interface and functionality as the Virtex-6 FPGA component. The component has been renamed to BSCANE2 to remove the Virtex-6 FPGA reference; however, it can be treated the same way as the prior named component. If a `BSCAN_VIRTEX6` component exists in the design, it is re-targeted to a `BSCANE2` component automatically; however, it is still suggested to update the code to the new component when convenient.

BUFIODQS

The `BUFIODQS DQSMASK` functionality is not directly supported in the 7 series devices. This functionality was intended for memory interface strobes by the Memory Generation IP (MIG) core and has been moved into the dedicated Phaser circuitry. If a design has a MIG generated core containing this element, it is suggested to regenerate that core for the appropriate 7 series device to properly replace this circuitry. If the design contains an instantiated `BUFIODQS` where the `DQSMASK` pin is tied to ground and the `DQSMASK_ENABLE` attribute is set to `FALSE`, this component can safely be replaced with a `BUFIO` component if driving a single bank. If multiple adjacent banks are to be driven, this component should be replaced with a `BUFMR` followed by two or more `BUFIO`s. If the `DQSMASK` input and `DQSMASK_ENABLE` attribute are used, an alternative circuit needs to be constructed because that circuitry is not supported in the 7 series device.

BUFR

The BUFR primitive in 7 series devices is functionally equivalent to that in Virtex-6 devices. The primary difference is the BUFR can drive multiple adjacent clock regions in Virtex-6 devices whereas in 7 series devices, the BUFRs connectivity is limited to a single clock region. If the use of the BUFR in the Virtex-6 FPGA design was limited to a single clock region, no changes are necessary. It is possible to drive multiple regions with a regional clock; however, to do this in 7 series devices, a BUFMR can drive BUFRs in up to three vertically adjacent clock regions. The SIM_DEVICE attribute in the BUFR primitive is recommended to be changed to 7SERIES to properly reflect the target architecture; however, the same functionality should apply if it is set to VIRTEX6.

RECOMMENDED: *If this attribute is set to an architecture prior to Virtex-6 FPGAs, it is highly suggested to update it to 7SERIES to have the simulation behavior accurately reflect the device circuitry.*

CAPTURE_VIRTEX6

The capture component for 7 series devices has the same interface and functionality as the Virtex-6 FPGA component. The component has been renamed to CAPTUREE2 to remove the Virtex-6 FPGA name. However, CAPTUREE2 can be treated the same way as the prior component. If a CAPTURE_VIRTEX6 component exists in the design, it is re-targeted to a CAPTUREE2 component automatically; however, it is still recommended to update the code to the new component name.

FIFO18E1/FIFO36E1

The hard FIFO components in Virtex-6 devices have the same interface and functionality as the 7 series FIFOs. However, there is a new reset requirement prior to FIFO operation, and the flag de-assertion latency might be different. To ensure the proper circuit initialization, the number of clock cycles for which the reset must be held upon power-up increased from three cycles in Virtex-6 FPGAs to five cycles in 7 series FPGAs. If your circuitry was designed to assert reset for fewer than five clock cycles, or if there is a dependency on the flag de-assertion latency, update the design to ensure these changes are addressed. These components have a SIM_DEVICE property that, by default, observes the Virtex-6 FPGA behavior upon initialization.

RECOMMENDED: *Change the value of SIM_DEVICE to 7SERIES to obtain the proper functionality and latency for 7 series devices. If you choose not to change the value of SIM_DEVICE, perform a timing simulation to ensure the circuit behaves properly in this respect.*

FRAME_ECC_VIRTEX6

The configuration frame error detection and correction circuitry component for 7 series devices has the same interface and functionality as the Virtex-6 FPGA component. The component has been renamed to FRAME_ECCE2 to remove the Virtex-6 FPGA name; however, it can be treated the same way as the prior component. If a FRAME_ECC_VIRTEX6 exists in the design, it is re-targeted to a FRAME_ECCE2 component automatically; however, it is still recommended to update the code to the new component when convenient.

GTHE1_QUAD, GTXE1, IBUFDS_GTHE1, and IBUFDS_GTXE1

The GTHE1_QUAD and GTXE1 primitives and associated buffers and common circuitry are not supported in 7 series devices. They are not re-targeted due to base functionality differences with the serial transceiver circuitry in 7 series devices. These capabilities have been replaced by the GTXE2 and the GTHE2 components. It is recommended that if the design contains these components, they be replaced by a generated equivalent interface targeting the appropriate 7 series device.

ICAP_VIRTEX6

The internal configuration access port for 7 series devices has a similar interface and functionality as the Virtex-6 FPGA component except the data read and data supplied have distinct differences in configuration logic and registers. The component has been renamed to ICAPE2 to remove the Virtex-6 FPGA name, and the BUSY pin that existed on the ICAP_VIRTEX6 component has been removed from the 7 series ICAPE2 component. The ICAPE2 component has deterministic timing; thus it is suggested to modify the ICAP interface to rely on this timing rather than polling the BUSY port. If an ICAP_VIRTEX6 primitive exists in the design, it is suggested to replace it with an ICAPE2 component to ensure proper connectivity and simulation behavior in the design. Refer to *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1],for more details on using the ICAPE2 component.

IODELAYE1

The Virtex-6 FPGA IODELAYE1 component can be automatically re-targeted in certain configurations. The 7 series devices do not have an IODELAYE1 component; instead this same functionality is broken into two separate functions, IDELAYE2 and ODELAYE2. If a design has an IODELAYE1 component configured as an output-only delay (DELAY_SRC = "O"), it can be automatically re-targeted to an ODELAYE1 component. If it is solely an input delay (DELAY_SRC = "I"), it can be re-targeted if the IDELAY_TYPE attribute is not set to DEFAULT. An IODELAY component used as a bidirectional delay cannot be automatically re-targeted. To achieve this same functionality, an IDELAYE2 can be instantiated to the input path and an ODELAYE2 instantiated to the output path.

ISERDESE1

The Virtex-6 FPGA ISERDESE1 component has been replaced with the ISERDESE2 component in 7 series devices. In most cases, the ISERDESE2 component is a superset in terms of capabilities. The primary change is the addition of the Q7 and Q8 pins to allow up to 8-bit data widths using a single ISERDESE2 component and 14-bit data widths with the cascading of two ISERDESE2 components. In most cases, an ISERDESE1 component in the design is re-targeted and does not need to be changed. It is suggested to replace this component with the ISERDESE2 primitive when convenient; however, this is not necessary for targeting 7 series devices.

JTAG_SIM_VIRTEX6, SIM_CONFIG_V6, and SIM_CONFIG_V6_SERIAL

The JTAG_SIM_VIRTEX6 and the SIM_CONFIG_V6 components have been replaced in 7 series devices with JTAG_SIME2 and SIM_CONFIGE2, respectively. The functionality of these components has changed such that it is suggested to replace them with the appropriate component inside the testbench or simulation file to ensure proper functionality. The SIM_CONFIG_V6_SERIAL component does not have an equivalent component for 7 series devices.

MMCM_BASE and MMCM_ADV

The 7 series MMCM components have been renamed to MMCME2_BASE and MMCME2_ADV. From a functionality standpoint, the 7 series MMCM is a superset of the Virtex-6 FPGA MMCM. Thus, in most cases, the MMCM can be automatically re-targeted. The only cases where direct re-targeting might not be possible are when using the DRP port to change operational parameters and if two MMCMs are directly cascaded. In the DRP port case, while the interface has not changed, the address mapping and register data have changed, requiring code changes to load different data to perform the same functions. the cascaded MMCM case, the dedicated connectivity that existed in Virtex-6 FPGAs to allow this configuration does not exist in 7 series FPGAs.

OSERDESE1

In most cases, the Virtex-6 FPGA OSERDESE1 component can be automatically re-targeted to OSERDESE2 for 7 series devices. The changes in the block are due to the removal of the DDR memory interface circuitry to the new Phaser circuitry, which resulted in the removal of pins and attributes associated with the prior circuitry. When using this block with a memory interface, it is recommended to use the MIG IP tool to regenerate code using the new OSERDES and other associated circuitry to realize that circuit. When using the OSERDES without the DDR memory interface pins and options, the component should be automatically re-targeted.

PCIE_2_0

The Virtex-6 FPGA Integrated Block for PCI Express® has been replaced by the PCIE_2_1 block in 7 series devices. This new block has some new capabilities but cannot be directly re-targeted. It is recommended to regenerate any Virtex-6 FPGA design containing this IP with the CORE Generator tool, targeting the appropriate 7 series device. Consult *7 Series FPGAs Integrated Block for PCI Express Product Guide*, (PG023) [Ref 4]for more details about the PCIE_2_1 component.

STARTUP_VIRTEX6

The start-up component for 7 series devices has similar interface and functionality as the Virtex-6 FPGA component. The component has been renamed to STARTUPE2 to remove the Virtex-6 FPGA name; however, in most cases, it can be treated the same way as the prior component. If interfacing to a configuration SPI PROM post-configuration, it is suggested to consult *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1].

If a STARTUP_VIRTEX6 component exists in the design, it is re-targeted to a STARTUPE2 component automatically.

RECOMMENDED: *It is still suggested to update the code to the new component when convenient.*

SYSMON

The Virtex-6 FPGA SYSMON component can be directly re-targeted to 7 series devices. It is remapped to the native XADC component. The XADC component has more precision and capability. It is recommended to re-evaluate if it is advantageous to change the code to incorporate the new XADC capabilities.

TEMAC_SINGLE

The Virtex-6 FPGA TEMAC_SINGLE component is not supported in the 7 series device. It is recommended to use the appropriate soft IP to generate this functionality, if needed.

USR_ACCESS_VIRTEX6

The user access component for 7 series devices has the same interface and functionality as the Virtex-6 FPGA component. The component is renamed to USR_ACCESSE2 to remove the Virtex-6 FPGA name; however, it can be treated the same way as the prior component. If a USR_ACCESS_VIRTEX6 component exists in the design, it is re-targeted to the USR_ACCESSE2 component automatically. It is still recommended to update the code to the new component when convenient to do so.

Using Unimacros

All UNIMACROs in Virtex-6 FPGA designs are supported by 7 series devices. It is recommended but not required to change the DEVICE attribute to 7SERIES.

I/O Considerations

Input Hold Time

For situations where a global clock is used to clock an input register without the use of an MMCM, a delay was automatically inserted to ensure that there is no positive hold time requirements. This capability is not available in 7 series devices so extra care needs to be taken in this situation. To ensure there are no hold time requirements, these steps should be followed:

1. Use an MMCME2 or PLLE2 to remove the clock insertion delay.
2. Insert an IDELAYE2 into the data path to add enough data delay to ensure proper data capture.
3. Place the input register into the slice array rather than the ILOGIC.
4. Ensure external timing can tolerate a positive hold time.

Doing proper timing analysis should alert you to any such issues with I/O timing. It is recommended to timing constrain all I/O paths, even slow ones, to ensure that both setup and hold times are met.

Bank Types

Virtex-7 and Kintex-7 devices have two types of banks supporting different I/O standards and circuitry, while Virtex-6 devices have a more homogeneous bank structure. The two bank types in 7 series devices are referred to as High-Performance and High-Range banks. The most significant difference between High-Performance and High-Range banks is in I/O standard support. The High-Range banks support most I/O standards up to 3.3V V_{CCO} whereas the High-Performance banks support I/O standards at 1.8V V_{CCO} and below. Because all Virtex-6 FPGA banks support I/O standards up to 2.5V, any 2.5V standard in the design must be assigned to a High-Range bank. It is also suggested that any high-speed, low voltage standard (1.8V and below) be assigned to the High-Performance banks to get the best jitter and other performance metrics.

For larger Virtex-7 devices, only High-Performance banks exist in the device, and thus designs requiring I/O standards above 1.8V potentially need external circuitry such as level shifters to accommodate such standards.

Another difference between High-Performance and High-Range banks is that only the High-Performance banks include an `ODELAYE2` component. Thus for any output utilizing the output delay feature of the `IODELAYE1` component for Virtex-6 FPGAs or requiring such delay control on the output path must be assigned to a High-Performance bank.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. UltraFast Design Methodology Training Course
2. Essentials of FPGA Design Training Course
3. Vivado Design Suite QuickTake Video Tutorials

Vivado Documentation

1. *7 Series FPGAs Configuration User Guide* (UG470)
2. 7 Series FPGAs Overview (DS180)
3. *Xilinx 7 Series FPGA Libraries Guide for HDL Designs* (UG768)
4. *7 Series FPGAs Integrated Block for PCI Express Product Guide*, (PG023)

Please Read: Important Legal Notices