UltraScale Architecture Migration

Methodology Guide

UG1026 (v1.5) April 24, 2017

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/24/2017	1.5	Added information on UNIMACROS and XPMs in Introduction to the UltraScale Architecture.
03/30/2016	1.4	Added Xilinx® UltraScale™ information: Added Introduction to UltraScale Architecture. Added note about UltraRAM. Added UltraScale+ information. Added information about SDR registers for tri-state. Added note about wide SDP mode. Added Table 3, I/O Standards that Support Uncalibrated Source Termination in HP I/O Banks Added information about HD I/O banks Added information about UltraScale+ device-specific UNISIM primitive. Added link to UltraScale+ Architecture data sheets.

Table of Contents

Migrating a Design to an UltraScale Device

Introduction to the UltraScale Architecture

The Xilinx® UltraScale™ architecture is the first ASIC-class All Programmable architecture to enable multi-hundred gigabit-per-second levels of system performance with smart processing, while efficiently routing and processing data on-chip. UltraScale architecture-based devices address a vast spectrum of high-bandwidth, high-utilization system requirements by using industry-leading technical innovations, including next-generation routing, ASIC-like clocking, 3D-on-3D ICs, multiprocessor SoC (MPSoC) technologies, and new power reduction features. The devices share many building blocks, providing scalability across process nodes and product families to leverage system-level investment across platforms.

Virtex® UltraScale+™ MPSoC devices provide the highest performance and integration capabilities in a FinFET node, including both the highest serial I/O and signal processing bandwidth, as well as the highest on-chip memory density. As the industry's most capable FPGA family, the Virtex UltraScale+ devices are ideal for applications including 1+Tb/s networking and data center and fully integrated radar/early-warning systems.

Virtex UltraScale+ MPSoC devices provide the greatest performance and integration at 20 nm, including serial I/O bandwidth and logic capacity. As the industry's only high-end FPGA at the 20 nm process node, this family is ideal for applications including 400G networking, large scale ASIC prototyping, and emulation.

Kintex® UltraScale+ devices provide the best price/performance/watt balance in a FinFET node, delivering the most cost-effective solution for high-end capabilities, including transceiver and memory interface line rates as well as 100G connectivity cores. Our newest mid-range family is ideal for both packet processing and DSP-intensive functions and is well suited for applications including wireless MIMO technology, Nx100G networking, and data center.

Kintex UltraScale devices provide the best price/performance/watt at 20 nm and include the highest signal processing bandwidth in a mid-range device, next-generation transceivers, and low-cost packaging for an optimum blend of capability and cost-effectiveness. The family is ideal for packet processing in 100G networking and data centers applications as well as DSP-intensive processing needed in next-generation medical imaging, 8k4k video, and heterogeneous wireless infrastructure.

Zynq® UltraScale+ MPSoC devices provide 64-bit processor scalability while combining real-time control with soft and hard engines for graphics, video, waveform, and packet processing. Integrating an ARM®-based system for advanced analytics and on-chip programmable logic for task acceleration creates unlimited possibilities for applications including 5G Wireless, next generation ADAS, and Industrial Internet-of-Things.

IMPORTANT: *UNIMACROs are not supported in the UltraScale® device architecture. Replace any UNIMACROS with Xilinx Parameterized Macros (XPMs).*

For details on XPMs, see *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 12].

For details on the various XPMs and their parameterization options, see the *UltraScale Architecture Libraries Guide* (UG974) [Ref 7].

Note: In the 2017.1 Vivado release and beyond, XPMs are enabled automatically in project mode, and in non-project mode are used automatically during synthesis/implementation.

Differences Between UltraScale FPGA Families

This document uses the Kintex UltraScale™ and Virtex UltraScale families as the basis for descriptions and examples. The following list defines some of the differences for the Kintex UltraScale+, Virtex UltraScale+, and Zynq UltraScale+ families:

- UltraRAM Support
- High-density I/Os (HD I/Os) with corresponding logic resources for Zynq UltraScale+ MPSoC devices and the Kintex UltraScale+ family
- MIPI D-PHY transmitter and receiver functions are supported in the HP I/Os specific to the Virtex UltraScale+ devices, Kintex UltraScale+ devices, and Zynq UltraScale+ MPSoCs
- Expanded functionality system monitor

For details on supported features in the HD I/O Banks, see the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1]

Targeting Considerations for UltraScale Devices

This section covers topics associated with programmable logic design implementations that are not technology-specific. Consider the concepts described when evaluating existing design source for potential migration to UltraScale devices or when developing new code for use in UltraScale devices.

RECOMMENDED: *See the UltraFast Design Methodology Guide for the Vivado Design Suite (UG949) [Ref 2] before starting on a new design or retargeting an existing design to ensure compliance with best practices in pinout selection, design creation, implementation, and debug.*

UltraScale Device Selection

UltraScale devices and other architectures have a logic cell count as a means of defining the amount of general-purpose logic resources in different size devices. In UltraScale devices, the logic cell count is much higher than in other devices.

Although the number of logic cells is a good indicator of the available programmable LUT resources, it is not always the best indicator of the overall logic capacity of the device. UltraScale devices contain several hardened functions that in earlier FPGAs or other technologies might use LUTs, flip-flops, and other resources. For example, the use of the hardened Interlaken core, PCIe® Gen3 block, integrated FIFO, and other blocks might represent a significant LUT savings over earlier implementations and should be considered a part of the device selection process.

Because logic density is not the only factor in determining device selection, all other resources required for your design should also be evaluated.

RECOMMENDED: *Initially compare all relevant resources in the Feature Summary tables in the UltraScale Architecture and Product Overview (DS890) [Ref 3].*

Additional considerations that help determine the appropriate UltraScale device selection include:

- Block RAM/DSP: For designs requiring heavy block RAM or DSP use, Kintex® UltraScale devices have higher block RAM and DSP content for a given array size.

  If targeting an UltraScale+ device, you might want to consider using UltraRAM for memory arrays larger than 128 Kb. For additional details on the UltraRAM, refer to the *UltraScale Architecture Memory Resources User Guide* (UG573) [Ref 4].

- I/O: The number of I/Os for a given array size, the bank sizes, arrangement, and capabilities differ from the 7 series devices.

When starting a new design, consider whether or not more functionality can be folded into the device to limit the number of I/Os required and also to improve overall system performance, cost, power, and capability.

- Speed Grade: In many cases, similar performance can be seen in UltraScale devices using a lesser speed grade. However, individual block speed as well as overall system performance should be analyzed before making a selection. When possible, implement the prior design, specifying the desired speed grade to determine if timing can be met before finalizing speed grade decisions.

Use of Control Signals

The use of control signals (those signals that control synchronous elements such as clock, set, reset, and clock enable) can impact device density, utilization, and performance.

When compared with prior architectures, the UltraScale architecture provides more flexibility regarding the number and types of control signals specified in the design. However, it is still best to be aware of the use and impact that control signals and control sets might have on design implementation.

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

As with prior generations of FPGAs, Flip-Flops in an UltraScale device 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. This could 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 there is 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 therefore be avoided.

You cannot implement (1) registers that contain both asynchronous reset and asynchronous set signals and/or (2) registers that contain an asynchronous control signal with a dynamic value. The Vivado Design Suite issues an error if either of these behaviors are described in RTL or if there is an instantiated FDCPE or LDCPE that allows this asynchronous behavior.

This construct can be described in RTL or instantiated as an FDCPE in HDL or EDIF formats.

Examples Showing Coding Styles to Avoid and Appropriate Corrections

EXAMPLE 1 - Avoid this Coding Style

Shown below is a simple example 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;
```

Use this Coding Style Instead

If the set and reset signals are considered necessary for proper function, convert one or both to a synchronous specification, as shown in the example below:

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

EXAMPLE 2 - Avoid this Coding Style

The following 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;
```

Use this Coding Style Instead

In this case, make `initc` synchronous to build an improved circuit:

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

Register Initialization

Many engineers use the inherent initialization of registers and latches in the FPGA through 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, consumes less logic than an uninitialized state yields. (An example is a state machine with certain states that must be run at start-up but perhaps not on a subsequent reset.) 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 best to use register initialization on any inferred register, shift register lookup table (SRL), or RAM, when possible.

In the following 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 above, 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 best to specify register initialization so that the simulation start-up state more accurately reflects the initial condition of the FPGA without requiring a reset condition.

Control Signal Polarity

The UltraScale architecture is enhanced over 7 series FPGAs and other prior architectures. Resets now have programmable inversion, allowing either an active-High or active-Low reset without any area or timing penalty. When there is a choice, it is best to use a single polarity reset (either all active-High or all active-Low) to avoid any possible impact on utilization. However, using mixed polarity resets causes less adverse impact on an UltraScale device design than on designs based on earlier-generation devices.

When coding or instantiating components with clock enable, however, use active-High enables. The enable pins do not have programmable inversion, and using an active-Low enable can have a negative impact on area and timing during implementation. Also, to get the best implementation, avoid inverting the enables when creating IP or instantiating UNISIM primitives.

Limit Use of Low Fanout Control Signals

When compared with earlier devices, the UltraScale device configurable logic block (CLB) allows more flexibility in the use of control signals in the architectures. Even so, 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 under utilized CLBs in terms of registers, SRLs, and LUT RAM. These control signals can have negative impacts on placement and timing. In general, do not implement a set, reset, or clock enable in the code unless it is required for the active functionality of the design.

Avoid Unnecessary Use of Sets or Resets

Unnecessary sets and resets in the code can prevent the inference of SRLs, RAM (LUT RAM or block RAM), and other logic structures that are otherwise possible. To maximize the efficiency of the architecture, code sets and resets only when they are necessary for the active functionality of the design. 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.
- A reset is not required 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.
- When the reset is held for multiple clock cycles, 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 DSP48E2 Registers

UltraScale device DSP48E2 registers contain only resets and not sets. The DSP block can perform many different functions, including multiplication, addition/subtraction, comparators, counters, and general logic. To allow flexible use of the DSP48E2, do not describe a set condition on any registers expected to be mapped to this resource. 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 DSP48E2 slice.

Use of Synchronous Sets/Resets

If a set or reset is necessary for proper operation of the circuit, always code a synchronous reset. 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 such as the DSP48E2 and RAMB36E2/RAMB18E2 cannot implement an asynchronous set and reset.

They cannot, therefore, be used with functional equivalence if an asynchronous set or reset exists in the RTL code. Synchronous resets provide enhanced flexibility with respect to register sharing within a CLB. In addition, placement is less affected, or in many cases, improved, so fewer routing resources are required. Additional benefits include overall device density, performance and power improvements, as well as reduced routing delay and capacitance.

High Fanout Signals

It is best to limit fanout of signals as much as possible. In some designs, however, high fanout nets will exist. In general, high fanout signals should not be split manually or replicated but coded as a single signal. If replication becomes necessary for timing or other reasons, the signal can be replicated during synthesis, or later, during physical optimization (`phys_opt_design`). This allows for greater flexibility to control replication many times, providing 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. Controlling this from the synthesis and implementation tools means that you do not have to modify and reverify the original code for such changes.

In prior architectures, placing high fanout signals onto global resources or BUFGs was at times preferable and yielded better results. When targeting UltraScale devices, however, it is generally best not to do this. The Vivado® tools automatically use global resources when available and when determined to be beneficial for timing or routability. Instantiating or using BUFGs is therefore not necessary.

Use of DSP and Other Arithmetic-Intensive Code

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

The DSP48E2 blocks use a signed arithmetic implementation. To best match the resource capabilities and, in general, to get the most efficient mapping, write code using signed values in the HDL source. If you use unsigned bus values in the code, the synthesis tools should still be able to use this resource, but they might not get the full bit precision of the component because of the unsigned-to-signed conversion. The multiplier within the DSP48E2 has an input bit precision of 18 bits by 27 bits signed data. Thus, the bit precision for unsigned data is 17 bits by 26 bits. For Verilog code, data is considered unsigned unless otherwise declared in the code.

If you expect the target design to contain a large number of adders, it is best to evaluate the design and make greater use of the DSP48E2 pre- and post-adders. For example, with FIR filters, use an adder cascade 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 fewer LUTs and flip-flops as well as fewer DSP blocks (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. With awareness of these capabilities, you can decide on the proper trade-offs in advance and account for them in the RTL code, which results in a smoother and more efficient implementation from the start.

RAM Considerations

When retargeting block RAM and LUT RAM by inference, instantiated primitive, or Vivado IP Catalog, there are two methods by which to maximize use of block RAM and LUTs in the UltraScale architecture. Choose the one of the following:

- If a core was used for RAM generation, regenerate the IP for the UltraScale device.
- Recode the RAM for proper synthesis inference.

Either method can provide good results for utilization and performance. However, it is best to infer memory, where possible, to improve understanding of the code, simulation, and portability of the code.

Using FIFOs

Many designs require FIFOs to transfer data to different clock domains or to store data until needed by other portions of the design. When targeting such designs to an UltraScale device, consider using one of the dedicated FIFO structures for this purpose. The integrated FIFOs generally allow higher speed operation with lower power and area requirements when compared to soft solutions, while at the same time improving reliability or Mean Time Between Failures (MTBF). The UltraScale device FIFO can be incorporated into the design by direct instantiation or by using the Vivado IP Catalog FIFO Generator.

Use of XOR Functions (CRC/Error Detection Correction)

Several error detection/correction, encryption, and other types of algorithms make use of wide XOR functions to realize the circuit. In prior-generation devices, this XOR circuitry was often implemented using LUTs and could represent a timing bottleneck and/or consume a disproportionate amount of power in the circuit. In the UltraScale architecture, a wide XOR function is added to the DSP48E2, which can efficiently implement parity, CRC, checksums, and other functions requiring large XOR computations.

RECOMMENDED: *To use the XOR function most efficiently, ensure proper pipelining around the function to use the registers in the DSP block.*

Asynchronous Data Synchronization

Most modern designs contain more than one clock domain. When a clock domain does not have a known phase relationship and/or incompatible frequencies, data transference across such domains must be managed appropriately. For buses of data, the integrated FIFO logic is the safest and most efficient means of bus-wide data transference. However, for a single signal, such as a reset that might not be synchronized to the desired clock domain, it is best to use proper synchronization techniques. Proper techniques include:

- Incorporating one or more synchronization register
- Identifying the synchronization register(s) with the ASYNC_REG property so that the tools can use place and route techniques to improve MTBF and so analysis tools can identify domain crossings of interest for reliability studies

Use of LUTs as Pack-Thrus

To gain access to all the circuitry in the Slice/CLB, LUTs must sometimes be configured as simple buffers to allow data to pass through LUT inputs into the internal circuitry. In prior architectures, configurations were called route-thrus or pack-thrus. Such pack-thrus were often used to gain access to all the registers in the slice, which in register-intensive designs, could consume a portion of the logic resources. The UltraScale architecture is designed so that the LUTs and registers can be accessed independently from one another, eliminating a common use for pack-thrus and allowing for greater logic usage/utilization. Register-intensive designs are therefore simpler to implement in UltraScale architectures.

Clocking Considerations

The clocking architecture for UltraScale devices is designed with many changes and improvements over earlier architectures. This section provides high-level guidance on retargeting existing clocking structures from 7 series FPGAs to UltraScale devices. For more in-depth information about clocking architecture changes, see the *UltraScale Architecture Clocking Resources User Guide* (UG572)[Ref 5].

RECOMMENDED: *Though in most cases, the clocking buffers and clock modifying blocks (that is, MMCM and PLL) automatically retarget to the UltraScale device clocking network, it is a good idea to review the clocking network for potential changes that might better suit the design in this architecture.*

The 7 series FPGA clocking buffers are retargeted to one of three types of UltraScale device clocking buffers, depending on use. These are BUFGCE, BUFGCTRL or BUFCE_DIV. In many cases, if multiple, redundant buffers appear in series or parallel, the Vivado Design Suite logic optimization algorithms (`logic_opt`) attempt to reduce these buffers to the minimum number to improve clock resource efficiency and improve any possible clock domain crossing timing. MMCMs and PLLs from 7 series FPGA designs are retargeted to the UltraScale device MMCME3, which is a superset of functionality for either of those clock-modifying components.

For clocks originating from Gigabit transceivers, a new clocking structure exists in the UltraScale architecture to facilitate the delivery of the clock to the FPGA resources. This involves using the BUFG_GT and BUFG_GT_SYNC components to connect and distribute the clock and cannot be automatically migrated from existing 7 series FPGA designs. For details on the clocking circuitry in UltraScale devices, see the appropriate UltraScale architecture GT transceivers user guide. For more information, the documentation available from the Support page; see Appendix A, Additional Resources and Legal Notices.

Retargeting Considerations from 7 Series to UltraScale Devices

The following section describes additional design implementation and constraints details to consider when retargeting an existing 7 series device design to an UltraScale device.

Use of Existing Soft IP, EDIF, or NGC Netlists

RECOMMENDED: *Regenerate or resynthesize existing soft IP or black box netlists in the design prior to implementation in UltraScale devices.*

Many netlists targeting the 7 series architectures can be implemented without error when targeting UltraScale devices. However, when migrating from earlier architectures to the UltraScale architecture, it is best to regenerate any netlist or core to maximize efficiency and to ensure that new features and capabilities are fully realized.

NGC format netlists are no longer supported in the UltraScale architecture. If your existing design contains NGC netlists, you must convert them to Electronic Data Interchange Format (EDIF) before targeting an UltraScale device. There is a utility in the Xilinx Tcl Store to assist with converting an NGC netlist to EDIF. To gain access to this utility, open the Xilinx Tcl Store and install the Project Utilities (`projutils`) application if it is not installed already. Once installed, the command `convert_ngc` is available to simplify this conversion. Type `convert_ngc -help` for details on running this command.

UltraScale Device IP Migration

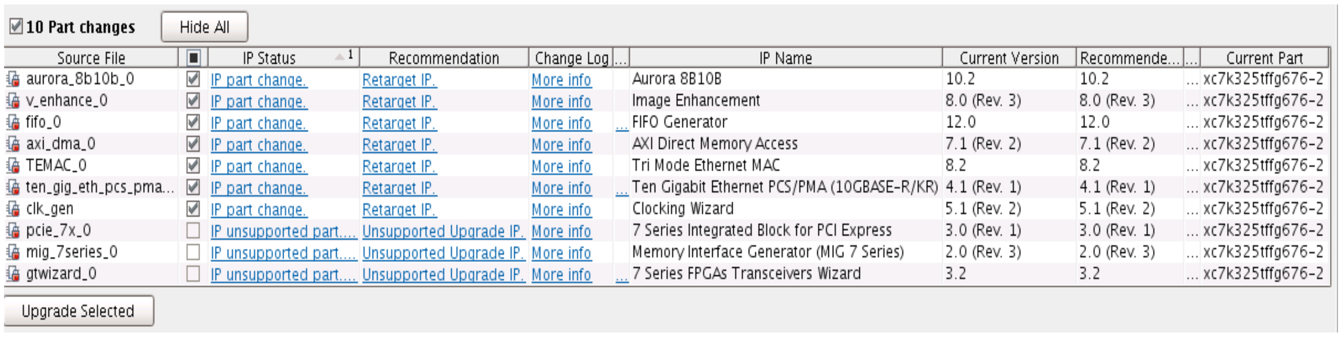
In general, migrating a 7 series FPGA design containing Xilinx IP to an UltraScale device involves a simple upgrade of the IP or the creation of a new UltraScale device-specific version of the IP. Depending on the particular IP, there might be changes required to the design so that it works with the upgraded IP. Review of the change log for the IP to get details on the upgrade changes.

When migrating IP, you must take into account the following conditions:

- The IP has an upgrade path and reports the status as "IP part change" (or "IP board change") when running Report IP Status. The version of the IP does not change. In this case, review the change log. Look for changes specific to use of the IP with the UltraScale architecture that could include port and functionality changes. For example, when updating a transceiver from a 7 series device to an UltraScale device, if you have selected transceiver debug ports, the name and function of these ports will have changed. Refer to the product guide section on transceiver debug ports for the new part in this case.
- The IP does not have an upgrade path to the UltraScale architecture. Depending on the IP, another version might be available. See the Product Guide for the specific IP for more information.

RECOMMENDED: *When migrating a 7 series FPGA design, first create or modify your project to target the UltraScale device. Next, upgrade the IP to the latest available version in the Vivado tools release you are using. Then generate a Report IP Status report, as shown in the figure below. See Table 1 for a list of commonly used IP.*

This report shows whether there is an upgrade path or whether the IP is unsupported in the UltraScale architecture. From this report you can review the change logs for the IP, which contain an upgrade path. Each change log provides a summary of the changes for the specific IP and can contain specific information to determine if changes made for the UltraScale architecture are applicable. See also the related IP Product Guide for UltraScale device-specific details.



Source File	■	IP Status	Recommendation	Change Log	...	IP Name	Current Version	Recommende...	...	Current Part
aurora_8b10b_0	✓	IP part change.	Retarget IP.	More info		Aurora 8B10B	10.2	10.2	...	xc7k325tffg676-2
v_enhance_0	✓	IP part change.	Retarget IP.	More info		Image Enhancement	8.0 (Rev. 3)	8.0 (Rev. 3)	...	xc7k325tffg676-2
fifo_0	✓	IP part change.	Retarget IP.	More info	...	FIFO Generator	12.0	12.0	...	xc7k325tffg676-2
axi_dma_0	✓	IP part change.	Retarget IP.	More info		AXI Direct Memory Access	7.1 (Rev. 2)	7.1 (Rev. 2)	...	xc7k325tffg676-2
TEMAC_0	✓	IP part change.	Retarget IP.	More info		Tri Mode Ethernet MAC	8.2	8.2	...	xc7k325tffg676-2
ten_gig_eth_pcs_pma...	✓	IP part change.	Retarget IP.	More info	...	Ten Gigabit Ethernet PCS/PMA (10GBASE-R/KR)	4.1 (Rev. 1)	4.1 (Rev. 1)	...	xc7k325tffg676-2
clk_gen	✓	IP part change.	Retarget IP.	More info		Clocking Wizard	5.1 (Rev. 2)	5.1 (Rev. 2)	...	xc7k325tffg676-2
pcie_7x_0	☐	IP unsupported part....	Unsupported Upgrade IP.	More info		7 Series Integrated Block for PCI Express	3.0 (Rev. 1)	3.0 (Rev. 1)	...	xc7k325tffg676-2
mig_7series_0	☐	IP unsupported part....	Unsupported Upgrade IP.	More info		Memory Interface Generator (MIG 7 Series)	2.0 (Rev. 3)	2.0 (Rev. 3)	...	xc7k325tffg676-2
gtwizard_0	☐	IP unsupported part....	Unsupported Upgrade IP.	More info	...	7 Series FPGAs Transceivers Wizard	3.2	3.2	...	xc7k325tffg676-2

Figure 1: **Report IP Status After Part Change to an UltraScale Device**

The following table provides a list of the commonly used IP with migration information for each.

Table 1: **UltraScale Device Migration Information for Commonly Used IP**

IP	Notes
Ten Gig Ethernet PCS PMA	Can be upgraded Port changes Reset functional change
7 Series PCIe 2.1 core (except 7 Series Virtex® XT core)	Cannot be upgraded Create PCIe 3.1 for Ultrascale devices.
7 Series Virtex XT PCIe 3.0 core	Minor port differences As of this document release date, not yet auto-upgradable
MIG	Cannot be upgraded Must use MIG 5.0 for UltraScale devices
Transceiver Wizard	Cannot be upgraded Must use an UltraScale FPGA Transceiver Wizard
FIFO	Can be upgraded Port changes Functional changes
Aurora	Can be upgraded Port changes

Software Considerations

UltraScale devices are supported by the Vivado Design Suite tools. Consult the current release notes for system requirements (for example, operating system and memory requirements) as well as device support and other important information about the tool suite.

Baselining the Design

RECOMMENDED: *During the initial stages of design retargeting, "baseline" the design using the method specified in the UltraFast Design Methodology Guide for the Vivado Design Suite (UG949) [Ref 2].*

Start with simple design constraints and a simple design flow and slowly expand upon those to ensure that the proper constraints are set in the design and that the best design flow is established for this new target.

Use of Synthesis Attributes and Physical Constraints

Often, synthesis attributes (`KEEP`, `MAX_FANOUT`, `DONT_TOUCH`, etc.) or physical constraints (`LOC`, `PBLOCK`, `RLOC`, `PROHIBIT`, etc.) that improved the results in earlier architectures might not apply and/or might degrade the results targeting another device. For this reason, it is best to reevaluate and potentially remove all synthesis attributes from the HDL code and/or Vivado synthesis XDC file. Similarly, it is best to evaluate and potentially remove all physical constraints from the implementation XDC file. Ensure that the need for attributes such as `KEEP` or `MARK_DEBUG` still exists and if not, remove them, as they might prohibit optimal results. After initial implementation and analysis, you can determine which, if any, of these attributes should be reintroduced. Before doing so, however, allow the tools to implement the design without such restrictions and see if the desired implementation occurs without intervention.

Specification of Timing Constraints

Vivado synthesis and implementation tools use the SDC (Synopsys Design Constraints) syntax for timing constraint specification. If the Vivado Design Suite was used for the 7 series FPGA design, in most cases, the same timing constraints can be carried forward to the UltraScale device implementation without modification. If the ISE® Design Suite was used previously, the timing constraints specified in UCF format must be translated into the XDC syntax. See the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 6] for details.

One change to note in migrating constraints to an UltraScale device design is that `PRIMITIVE_GROUP` and/or `PRIMITIVE_SUBGROUP` properties for constraint processing might have changed. If you are using these in your design, see the *UltraScale Architecture Libraries Guide* (UG974) [Ref 7] for the correct `PRIMITIVE_GROUP/PRIMITIVE_SUBGROUP` properties for the UNISIM components. This helps you ensure the constraints are bound to the proper cells.

During most retargeting scenarios, modified IP and other design areas can impact the overall timing specification where timing paths change in name, reference points, and/or type, thus potentially invalidating constraints properly specified in a earlier target. Particularly for timing paths that refer to internal nodes of the design, ensure that the referenced node still exists and that the timing specification or exception is still applicable. Use `check_timing` to validate that the specified timing constraints are complete and accurate for the new target.

Simplifying the Design Flow

As a design progresses, the design flow is often augmented with the use of different options or directives, use of different implementation steps (such as `power_opt_design` or `phys_opt_design`, or perhaps even more advanced control (such as pre- or post-Tcl processing). Crafting such a flow often leads to improved results for a given design targeting a given architecture. However, changes in the design and/or target architecture might require different tactics. So, it is not always the best strategy to use the same options and flow used when targeting 7 series device as you would when targeting an UltraScale device.

RECOMMENDED: *After the design and constraints are re-verified, run a default flow through the Vivado tools. Analyze the design for possible design or constraint changes to better facilitate progress towards the desired goals. After that is exhausted, start re-exploring different tool options and directives as well as flows.*

7 Series Device Primitive Retargeting Considerations

The following primitives are identical between 7 series and UltraScale devices. There should be no changes in behavior, connectivity, or general use:

AND2B1L	FDCE	LUT3	RAM128X1D
BUFG	FDPE	LUT4	RAM256X1S
BUFGCE	FDRE	LUT5	RAM32M
BUFGCTRL	FDSE	LUT6	RAM32X1S
BUFGMUX	KEEPER	MUXF7	RAM64M
BUFGMUX_1	LDCE	MUXF8	RAM64X1D
BUFGMUX_CTRL	LDPE	OR2L	RAM64X1S
CFGLUT5	LUT1	PULLDOWN	SRL16E
EFUSE_USR	LUT2	PULLUP	SRLC32E

Some device primitives instantiated for the 7 series architecture are either not automatically retargeted to the UltraScale architecture, or they are retargeted with some notable differences. See the *UltraScale Architecture Libraries Guide* (UG974) [Ref 7], for more details on any UNISIM component and the respective user guide for specific details about any of these blocks when targeting UltraScale devices.

This following sections highlight a few of these older components.

Advanced 7 Series Component Retargeting

GTPE2_CHANNEL, GTPE2_COMMON, GTHE2_CHANNEL, GTHE2_COMMON, GTXE2_CHANNEL, GTXE2_COMMON, IBUFDS_GTE2

The gigabit transceiver primitives, their associated buffers, and common circuitry are not supported in UltraScale and UltraScale+ devices. They are not retargeted because of base functionality differences with the serial transceiver circuitry in UltraScale and UltraScale+ devices. These capabilities have been replaced by the E3 series of transceiver components for UltraScale devices and the E4 series of transceiver components for UltraScale+ devices.

IMPORTANT: *If your design contains these older components, replace them with a Vivado IP Catalog-generated equivalent interface, targeting the appropriate UltraScale device.*

XADC

Some specifications for the UltraScale device `SYSMONE1` and UltraScale+ device `SYSMONE4` differ from the 7 series device XADC component. When using an instantiated XADC in the design, see the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 8] to learn if any of these changes might impact the use of this component for the given application.

RECOMMENDED: *If you determine that the `SYSMONE1` or `SYSMONE4` should be used, run the System Management wizard in the Vivado Design Suite IP catalog to integrate this function into your design.*

7 Series Config/BSCAN Component Retargeting

Note: For additional details on the Configuration components, see the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 9].

CAPTUREE2

In comparison with earlier devices, the capture feature in the UltraScale device has changed. As a result, there is no retargeting of the `CAPTUREE2` primitive, and this causes an error in UltraScale device designs. Use the Vivado® logic analyzer for a more complete solution. See the "Readback Verification and CRC" section of the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 9] for additional information.

DNA_PORT

The `DNA` value is extended from 57 bits to 96 bits, and the primitive now reflects this new behavior.

RECOMMENDED: *Update any design containing a `DNA_PORT` to instantiate and use the `DNA_PORTE2` primitive, which allows proper functional simulation with this function.*

FRAME_ECCE2

7 series devices and UltraScale devices differ with respect to Configuration Frame Error Correction Circuitry or FRAME_ECC pinout and functionality. For this reason, the FRAME_ECCE2 component causes an error in UltraScale device designs. For designs still requiring this functionality, replace the FRAME_ECCE2 with the FRAME_ECCE3 component.

RECOMMENDED: *For details on the changes and use of this new component, see the UltraScale Architecture Configuration User Guide (UG570) [Ref 9].*

ICAPE2

The Internal Configuration Access Port (ICAP) component differs in terms of pinout, functionality, and latency. For this reason, the ICAPE2 component does not automatically retarget, and an error results. For designs requiring ICAP functionality, use the ICAPE3 component.

RECOMMENDED: *See the UltraScale Architecture Configuration User Guide (UG570) [Ref 9] for additional details on this component.*

STARTUPE2

Some additional pins and functionality are added to the STARTUP block, so there is now a STARTUPE3 block. In most cases, this block retargets cleanly. However, it is best to modify code, when convenient, to instantiate and use the new STARTUPE3 block.

RECOMMENDED: *See the UltraScale Architecture Configuration User Guide (UG570) [Ref 9] for additional details on this component.*

7 Series I/O Component Retargeting

IDELAYE2

The IDELAYE2 primitive is replaced with the IDELAYE3 primitive in the UltraScale architecture. The input delay line now consists of 512 taps instead of the previous 32, resulting, potentially, in less maximum delay. Refer to the appropriate UltraScale data sheet [Ref 11] for specifics. You can cascade IDELAYE3 with its associated ODELAYE3 output delay primitive for a longer delay, if required. In FIXED mode, the delay is now given in terms of ps rather than tap value. The delay line uses this value and the associated reference clock (via the IDELAYCTRL) to calculate how many taps are needed to generate the required delay at current PVT (process/voltage/temperature), with no user intervention required. In VAR_LOAD mode, programming the IDELAYE3 now requires a 9-bit value.

The IDELAY can no longer connect directly to a clocking resource such as a BUFG or MMCM/PLL. If the 7 series FPGA design has such connectivity, you must modify it, using an MMCM/PLL for phase alignment and/or by shifting the data to get proper clock and data alignment.

Additionally, when the IDELAY is used in conjunction with an input register, IDDR or ISERDES, the reference clock (C) must be shared with the clock pin (C) of the register or IDDR or the CLK_DIV pin of the ISERDES component. The greater frequency range of the IDELAY reference clock in the UltraScale architecture should allow enough flexibility to accommodate a common clock between these components.

IN_FIFO

The IN_FIFO is replaced by an 8-deep FIFO that is optionally available inside the ISERDESE3 component. To gain access to this function, instantiate the ISERDESE3 and configure it to use this built-in capability.

ISERDESE2

The ISERDESE2 is replaced by the ISERDESE3 primitive. SerDes ratios of 4 and 8 are available when using DDR clocking, and 2 or 4 are available when using SDR clocking. Intermediate ratios require an FPGA logic gearbox. There is no longer a bitslip function available, and if required, this can be simply implemented in the FPGA logic.

There is no longer a SDR register for the tri-state. To support SDR bi-directional signals or outputs with tri-state place a data flip-flop in the IOLOGIC (OFD by using an IOB constraint) and the 3-state flip-flop in fabric and assign it as close as possible to the IOB. If there is a single flip-flop driving multiple tri-states, the flip-flop should be replicated and each individual tri-state flip-flop assigned as close as possible to the IOB.

An 8-deep, bypass-able FIFO is added to the ISERDESE3, allowing parallel data to be read from the primitive using a clock that is a non-integer divide of the sampling clock. To ensure best use of the I/O data capture capabilities in UltraScale devices, instantiate and use the ISERDESE3 in the design.

ODELAYE2

The ODELAYE2 primitive is replaced with the ODELAYE3 primitive in the UltraScale architecture. The output delay line now consists of 512 taps instead of the previous 32, and has a maximum delay of 1.25 ns. No delay cascade is possible. In FIXED mode the delay is now given in terms of ps rather than tap value. The delay line uses this value and the associated reference clock (through the IDELAYCTRL) to calculate how many taps are needed to generate the required delay at current PVT, with no user intervention required. In VAR_LOAD mode, programming the IDELAYE3 now requires a 9-bit value. The attribute value DELAY_SRC="CLKIN" is not supported.

Additionally, when the ODELAY is used in conjunction with an output register, ODDR or OSERDES, the reference clock (C) must be shared with the clock pin (C) of the register or ODDR or the CLK_DIV pin of the OSERDES component. The greater frequency range of the ODELAY reference clock in the UltraScale architecture should allow enough flexibility to accommodate a common clock between these components.

OSERDESE2

The OSERDESE2 is replaced by the OSERDESE3 primitive. SerDes ratios of 4 and 8 are available when using DDR clocking, and 2 or 4 are available when using SDR clocking. Intermediate ratios require an FPGA fabric gearbox. To ensure best use of the capabilities of I/O data capture in UltraScale devices, instantiate and use the OSERDESE3 in the design.

OUT_FIFO

There is no direct equivalent to the OUT_FIFO in the UltraScale architecture. If a similar feature is needed, consider generating a LUT-based FIFO using the IP catalog to replace the function of the OUT_FIFO.

IDDR

The IDDR is replaced by the IDDRE1 in the UltraScale architecture. The functionality remains the same, but the clock enable (CE) and set (S) pins are removed, as their functionality is no longer available. There is a reset available, but it can only operate as an asynchronous reset. If SRTYPE is set to "SYNC" it cannot be retargeted. All three existing capture modes (SAME_EDGE, OPPOSITE_EDGE, and SAME_EDGE_PIPELINED) are supported.

To ensure best use of the capabilities of I/O data capture in UltraScale devices, instantiate and use the IDDRE1 in the design.

ODDR

The ODDR is replaced by the ODDRE1 in the UltraScale architecture. The functionality remains the same, but the clock enable (CE), reset (R) and set (S) pins are removed, as their functionality is no longer available. The R and S pins are replaced with an SR pin that can either set or reset the component asynchronously, depending on programming. The INIT function must match that of the set or reset function meaning if you specify a reset, the INIT must be set to a zero or if you specify a set, it must be set to a one. Only the existing SAME_EDGE mode of operation is supported with the new primitive. Another difference is that when the set/reset function is released, the component cannot capture/transmit data until three clock cycles later.

To ensure best use of the capabilities of I/O data capture in UltraScale devices, instantiate and use the ODDRE1 in the design in order.

7 Series RAM/ROM Retargeting

FIFO18E1/FIFO36E1

The hard-FIFO blocks are updated in the UltraScale architecture with new features and capabilities, as well as more compatibility to soft versions of the FIFO core. Consequently, there are pinout, functionality, and latency changes that do not allow automatic retargeting of this component.

RECOMMENDED: *For designs that contain `FIFOE1` blocks, review the UltraScale Architecture Memory Resources User Guide (UG573) [Ref 4] for details about the changes to this block, and use a new `FIFOE2` block in its place. Alternatively, you can use the FIFO Generator in the Vivado IP Catalog to make it easier to instantiate this block into the design.*

RAMB18E1/RAMB36E1

The block RAM components for the UltraScale architecture are enhanced for greater functionality. However, these components are a superset of the features and functionality of the 7 series FPGA `RAMBE1` blocks. Typically, designs containing these `RAMBE1` blocks cleanly retarget to the UltraScale architecture with the same functionality and latency seen prior. However, it is best to review the use of any such components, as some of the enhancements made to the UltraScale device `RAMBE2` blocks might yield a better solution in terms of area, performance, power, and other metrics.

Note: In UltraScale devices only, if you are in the wide Simple Dual Port (SDP) mode (x72 for RAMB36 or x36 for RAMB18) and are using the output registers, you cannot have one port use the optional clock inversion while the other is not inverted. When targeting an UltraScale+ device, for larger memory arrays requiring four or more Block RAMs, you should consider migrating to one or more UltraRAMs if they are available in your target device.

7 Series Clock Component Retargeting

BUFG

Any `BUFGs` in the design are converted to `BUFGCEs`, with the clock enable (CE) pin constantly enabled. The functionality is the same, and in general, is the optimal resource in UltraScale devices.

BUFH/BUFHCE

The `HROW` buffers or `BUFHs` from 7 series FPGAs are deprecated because of the redesigned clocking network in UltraScale devices. Designs containing `BUFHs` are retargeted to `BUFGs`, which might exist in the final implementation or be combined with other `BUFGs` to form a proper clock network. Similarly, `BUFHCEs` are retargeted to `BUFGCEs`. In most cases, this retargeting provides a sufficient clocking structure.

However, it is a good idea to review the clocking topology with the UltraScale device clocking architecture in mind to see if further improvements or consolidation can occur. For more details about clocking resources, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 5].

BUFIO

The I/O Clock Buffer or `BUFIO` from 7 series FPGAs is deprecated because of the redesigned clocking network in UltraScale devices. Designs containing `BUFIOs` are retargeted to `BUFGs`, which might exist in the final implementation or be combined with other `BUFGs` to form a proper clock network. In most cases, this retargeting provides a sufficient clocking structure for low- to high-speed I/O clocking. However, it is a good idea to review the clocking topology with the UltraScale device clocking architecture in mind. For very high speed I/O interface implementations, the bit slice components have dedicated, low jitter clocking that can also be used in place of the BUFIO, but that must be used in conjunction with the bit slice components. For more details about clocking resources, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 5].

BUFR/BUFMR/BUFMRCE

The regional clock buffers `(BUFR`, `BUFMR`, `BUFMRCE)` from 7 series FPGAs are deprecated because of the redesigned clocking network in UltraScale devices. Designs containing any of these buffers are retargeted to `BUFGCE_DIVs`, which allow for the proper division functionality to be carried forward. In many cases, this retarget provides a sufficient clocking structure. However, it is a good idea to review the clocking topology with the UltraScale device clocking architecture in mind to see if further improvements or consolidation can occur. For more details about clocking resources, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 5].

MMCME2_ADV/MMCME2_BASE

The `MMCM` is improved to add additional functionality in UltraScale devices, so there is now an `MMCME3` component. However, `MMCME3`is a pure superset of the `MMCME2`. Thus, any `MMCME2` is expected to retarget automatically in an UltraScale device. If, however, the `MMCM` was created using the Clocking wizard or other IP, generate it to target the UltraScale device to ensure the most up-to-date representation of this resource. If the `MMCME2` is directly instantiated in the design, updating to the `MMCME3` is still recommended but can be done at your discretion.

PLLE2_ADV/PLLE2_BASE

The PLL in UltraScale devices is changed so that its functionality no longer maps directly to the `PLLE2` found in 7 series FPGAs. For this reason, the software still retargets PLL implementations to an UltraScale device automatically, but it retargets to the `MMCME3` (not the `PLLE3`) because the `MMCME3` is a superset of the `PLLE2` functionality. In most cases, this is adequate.

However, if you want to use PLLE3 with an UltraScale device, you must instantiate it directly into the design. Alternatively, you can use the Clocking wizard to provide the appropriate IP, including the PLLE3.

Retargeting 7 Series Registers and Latches

FDCE, FDPE, FDRE, FDSE

In comparison with 7 series FPGAs, UltraScale devices include a significant enhancement to the registers:

- 7 series device registers only supported an active-High reset.
- UltraScale devices include an optional inversion on the set/reset pin. This allows native implementation of an active-High or active-Low reset on the primitive. This inversion is represented and controlled by an attribute on the FD components:

```
FDCE: IS_CLR_INVERTED
FDPE: IS_PRE_INVERTED
FDRE: IS_R_INVERTED
FDSE: IS_S_INVERTED
```

There is no requirement to set this attribute if an inverter is described prior to the set or reset. The software automatically sets the attribute to yield the correct and optimal results.

RECOMMENDED: *For registered 3-stated output paths, UltraScale devices no longer contain a dedicated register on the 3-state path. Use a CLB register for the 3-state. Refer to Xilinx Answer #62490 (UltraScale I/O - Recommended design methodology for SDR 3-state flip-flops) [Ref 10] for more details.*

Slice/CLB Primitives

CARRY4

Because the size of the UltraScale device CLB doubled in comparison with the 7 series device slice, the carry logic also doubled to form a new component, the CARRY8. Designs that use CARRY4 components are automatically mapped efficiently into the new CARRY8 component.

RECOMMENDED: *Eventually, change any CARRY4s instantiated in the design to the native CARRY8 components. There is no immediate need to do this in most cases, however, the primary exception would be encountered in a design in which several single CARRY4s exist and where dense packing is required. In this case, modify the code to use CARRY8s instead of CARRY4s, where the packing of the two prior CARRY4s are defined in the code. This allows for denser designs that use several small carry chains.*

SelectIO Considerations

The architecture for SelectIO™ interface for UltraScale devices is redesigned and improved. This section provides high-level guidance on retargeting existing designs from 7 series devices to UltraScale devices for the SelectIO interface. For more in-depth information about SelectIO architecture changes, see the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1].

DCI Considerations

DCI Support

For both 7 series and UltraScale devices, DCI (digitally controlled impedance) is supported only in HP I/Os.

For UltraScale devices, only one resistor of 240 Ω to GND is required on the VRP pin to support this feature, regardless of the type and value of DCI termination desired.

Termination Circuit Resistor Requirements

In 7 series FPGAs, DCI calibrates each leg of the split-termination circuit to be directly equal to the external resistor values. For example, a 7 series device with a target parallel termination of 50 Ω to VCCO/2 requires 100 Ω external resistors on the VRN and VRP pins.

In UltraScale devices, only one 240 Ω resistor is required at the VRP pin. The exact values of the split-termination or single-termination resistors are determined by the user-controllable ODT attribute.

Possible ODT values for split-termination DCI standards (HSTL and SSTL)

RTT_40	RTT_48	RTT_60	RTT_NONE

IMPORTANT: *The ODT value represents the desired Thevenin resistance to VCCO/2 for split-termination DCI standards.*

Possible ODT values for single-termination POD standards

RTT_40	RTT_48	RTT_60	RTT_NONE

Possible ODT values for single-termination HSUL standards

RTT_120	RTT_240	RTT_NONE

IMPORTANT: *The ODT value represents the desired resistance to VCCO for single-termination DCI standards.*

Source Driver Termination

Source Driver termination must be specified through the `OUTPUT_IMPEDANCE` attribute for the I/O standards that support the controlled impedance driver.

Controlled impedance driver support is extended to all the following I/O standards:

HSTL_I_DCI	DIFF_HSTL_I_DCI_18	HSLVDCI_18
DIFF_HSTL_I_DCI	LVDCI_15	POD10_DCI
LVDCI_18	POD12_DCI	DIFF_POD10_DCI
HSUL_12_DCI	DIFF_POD12_DCI	SSTL135_DCI
DIFF_HSUL_12_DCI	SSTL15_DCI	DIFF_SSTL135_DCI
SSTL18_I_DCI	DIFF_SSTL15_DCI	HSLVDCI_15
DIFF_SSTL18_I_DCI	HSTL_I_DCI_12	SSTL12_DCI
HSTL_I_DCI_18	DIFF_HSTL_I_DCI_12	DIFF_SSTL12_DCI

See the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1] for details about the `OUTPUT_IMPEDANCE` attribute.

T_DCI Standards

The 7 series architecture supported `T_DCI` standards for bidirectional I/O configurations with 3-state support for internal input split-termination. Those `T_DCI` standards are not supported in UltraScale devices.

Many of the UltraScale architecture DCI standards are, however, capable of supporting similar bidirectional configurations. The table below lists the `T_DCI` standards that are transparently ported or migrated to an equivalent UltraScale architecture standard when designing with the Vivado Design Suite.

Table 2: **T_DCI I/O Standards for Migration Between Architectures**

7 Series Architecture I/O Standard	UltraScale Architecture Equivalent I/O Standard
`DIFF_HSTL_II_T_DCI`	No Migration
`DIFF_HSTL_II_T_DCI_18`	No Migration
`DIFF_SSTL18_II_T_DCI`	No Migration
`DIFF_SSTL15_T_DCI`	`DIFF_SSTL15_DCI`
`DIFF_SSTL135_T_DCI`	`DIFF_SSTL135_DCI`
`DIFF_SSTL12_T_DCI`	`DIFF_SSTL12_DCI`
`HSTL_II_T_DCI`	No Migration
`HSTL_II_T_DCI_18`	No Migration
`SSTL18_II_T_DCI`	No Migration
`SSTL15_T_DCI`	`SSTL15_DCI`
`SSTL135_T_DCI`	`SSTL135_DCI`
`SSTL12_T_DCI`	`SSTL12_DCI`

Uncalibrated Input Termination

`IN_TERM` is not a supported attribute in UltraScale devices. `ODT` must be used instead to specify the input termination value for HR I/Os. HP I/Os also have support for uncalibrated input termination that is accessed with the `ODT` attribute in UltraScale devices.

Uncalibrated Source Termination

Similar to DCI I/O standards, the option to choose the value of driver termination in UltraScale devices is available in non-DCI SSTL, HSTL, HSUL and POD standards. This is supported in HP I/Os only. This feature is accessed through the `OUTPUT_IMPEDANCE` attribute as well.

The following table shows a list of I/O standards that support uncalibrated source termination in HP I/O banks.

Table 3: **I/O Standards that Support Uncalibrated Source Termination in HP I/O Banks**

HSTL_I	DIFF_HSTL_I	SSTL18_I	DIFF_SSTL18_I	POD12	DIFF_POD12
HSTL_I_18	DIFF_HSTL_I_18	SSTL15	DIFF_SSTL15	POD10	DIFF_POD10
HSTL_I_12	DIFF_HSTL_I_12	SSTL135	DIFF_SSTL135	HSUL_12	DIFF_HSUL_12
		SSTL12	DIFF_SSTL12		

I/O Standards No Longer Supported

The following I/O standards are *not* supported in the UltraScale architecture:

`DIFF_HSTL_II_T_DCI`	`DIFF_SSTL12_T_DCI`	`SSTL135_T_DCI`
`DIFF_HSTL_II_T_DCI_18`	`HSTL_II_T_DCI`	`SSTL12_T_DCI`
`DIFF_SSTL18_II_T_DCI`	`HSTL_II_T_DCI_18`	`MOBILE_DDR`
`DIFF_SSTL15_T_DCI`	`SSTL18_II_T_DCI`	`DIFF_MOBILE_DDR`
`DIFF_SSTL135_T_DCI`	`SSTL15_T_DCI`	`PCI33_3`

The following Class II I/O standards are *not* supported in HP I/Os:

`DIFF_HSTL_II`	`DIFF_SSTL18_II`	`HSTL_II_18`
`DIFF_HSTL_II_18`	`HSTL_II`	`SSTL18_II`

VREF-based Inputs Standards

In 7-series devices, VREF-based input standards do not have a VCCO requirement and can be placed in banks with any VCCO, if the on-die input termination feature was not used. However, in the UltraScale architecture, the following VREF based input standards have a VCCO requirement and can only be placed in banks that have a compatible VCCO. This requirement applies even when on-die input termination (ODT) or internal VREF features are not used.

`HSUL_12`	`HSTL_I_DCI_18`	`SSTL12`
`HSUL_12_DCI`	`HSTL_I_12`	`SSTL18_I_DCI`
`HSLVDCI_18`	`HSTL_I_DCI_12`	`SSTL15_DCI`
`HSLVDCI_15`	`SSTL18_I`	`SSTL135_DCI`
`HSTL_I`	`SSTL18_II`	`SSTL12_DCI`
`HSTL_II`	`SSTL15`	`POD10`
`HSTL_I_DCI`	`SSTL15_R`	`POD12`
`HSTL_I_18`	`SSTL135`	`POD10_DCI`
`HSTL_II_18`	`SSTL135_R`	`POD12_DCI`
		`MIPI_DPHY_DCI*`

Note: * Applies only to UltraScale+ .

External VREF

`VREF` is a dedicated pin in UltraScale devices. There is one dedicated `VREF` per bank (or Mini-bank, where applicable).

For HD I/O banks, external VREF is not supported and is instead generated inside the device. The internally generated VREF is sourced from the VCCO supply rail. HD I/O banks are only supported in Zynq UltraScale+ MPSoCs and Kintex UltraScale+ devices.

SelectIO Interface UNISIM Primitives

All 7 series FPGA SelectIO interface UNISIM primitives are supported in UltraScale devices. There are also a few new UltraScale device-specific UNISIM primitives that support new features. These are `IBUFE3`, `IBUFDSE3`, `IOBUFE3`, `IOBUFDSE3` , `HPIO_VREF`, `IBUF_ANALOG`.

In addition, there are new UltraScale+ device-specific UNISIM primitives that support the new MIPI D-PHY feature. These are `IBUFDS_DPHY and OBUFDS_DPHY`.

- `DCITERMDISABLE` and `INTERMDISABLE` ports:
    - Dynamic input termination control port `DCITERMDISABLE` controls both DCI and uncalibrated on-die input terminations in HP I/Os in UltraScale devices.
    - In UltraScale devices (as in 7 series devices), dynamic input termination control port `INTERMDISABLE` controls the uncalibrated termination in HR I/Os.
- `IBUFDISABLE` ports:
    - In UltraScale devices, if `IBUFDISABLE` is asserted in a UNISIM, with `IBUFDISABLE` port enabled (and T is 1, if applicable), and output `O` to the internal logic is logic Low. This is a difference in behavior from 7 series devices, in which `IBUFDISABLE` assertion caused output `O` to the internal logic to be logic High.

V_{CCAUX_IO}

UltraScale devices differ from 7 series FPGAs in that the auxiliary I/O (V_{CCAUX_IO}) voltage supply rail does not support two voltage levels. The only voltage level supported is the nominal level of 1.8 V.

SLEW Attribute

In UltraScale devices, the `SLEW` attribute supports a new value of `"MEDIUM"` in HP I/Os.

Differential Input Termination

Though `DIFF_TERM` attribute is supported, it automatically maps to a new attribute called `DIFF_TERM_ADV` in UltraScale devices.

- `DIFF_TERM = TRUE` automatically maps to `DIFF_TERM_ADV = TERM_100` (100 ohm differential termination)
- `DIFF_TERM = FALSE` automatically maps to `DIFF_TERM_ADV = TERM_NONE` (default, no termination)

The allowed values for `DIFF_TERM_ADV` attribute are:

- `DIFF_TERM_ADV = TERM_NONE` (default)
- `DIFF_TERM_ADV = TERM_100`

New Features and Enhancements

For details about new features and enhancements, see the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1].

New Primitive Considerations

The following section lists a few new primitives available in the UltraScale architecture that do not have direct counterparts in 7 series FPGAs. These are listed in order of priority for addition to a retargeted design, with the goal of achieving better characteristics under certain conditions. More information about using any of these components can be found in the *UltraScale Architecture Libraries Guide* (UG974) [Ref 7].

Slice/CLB Primitives

MUXF9

The doubling of the UltraScale device CLB size (compared to the 7 series device slice) allowed the addition of the `MUXF9`. This MUX allows a single LUT logic level 32-to-1 MUX implementation in a single CLB. Alternatively, you can build certain large condition expressions, a 9-input LUT or a 1-bit wide by 512-bit deep ROM, using this component. The outcome is efficient utilization of all the LUTs in a CLB.

RAM512X1S/RAM256X1D

The larger CLB also facilitates the building of larger distributed or LUT RAM configurations, allowing for asynchronous read, synchronous write operation. Now 512-bit deep, single-port and 256-bit deep, simple dual-port functions can be natively built within a single CLB.

RAM32M16/RAM64M8

Larger configurations of multi-port (one write-port, several read-ports) can also be configured into a single UltraScale device CLB. Investigate using the `RAM32M16` or `RAM64M8` for any multi-port RAM access needs.

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.

References

1. *UltraScale Architecture SelectIO Resources User Guide* (UG571)
2. *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949)
3. *UltraScale Architecture and Product Overview* (DS890)
4. *UltraScale Architecture Memory Resources User Guide* (UG573)
5. *UltraScale Architecture Clocking Resources User Guide* (UG572)
6. *ISE to Vivado Design Suite Migration Guide* (UG911)
7. *UltraScale Architecture Libraries Guide* (UG974)
8. *UltraScale Architecture System Monitor User Guide* (UG580)
9. *UltraScale Architecture Configuration User Guide* (UG570)
10. Xilinx Answer Record 62490
11. UltraScale and UltraScale+ Architecture data sheets available on the Xilinx support site.
12. *Vivado Design Suite User Guide: System-Level Design Entry* (UG895)

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

Please Read: Important Legal Notices