UltraScale Architecture Configurable Logic Block

User Guide

UG574 (v1.6) January 22, 2025

AMD Adaptive Computing is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.

Table of Contents

AMD

Chapter 1

Overview

Introduction to the UltraScale Architecture

The AMD UltraScale™ architecture is the first ASIC-class 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 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.

AMD Spartan™ UltraScale+™ devices provide high I/O to logic ratio, integrated memory controllers, and advanced I/O that help Industrial, Vision and Healthcare (IVH), Audio, Video and Broadcast (AVB), Automotive, and other FPGA application developers looking to build cost-optimized solutions. Available in a wide array of packaging options, this family delivers a balance of cost, power, performance, and size.

AMD Artix™ UltraScale+™ devices provide high serial bandwidth and signal compute density in a cost-optimized device for critical networking applications, vision and video processing, and secured connectivity. Coupled with the innovative InFO packaging, which provides excellent thermal and power distribution, Artix UltraScale+ FPGAs are perfectly suited to applications requiring high compute density in a small footprint.

AMD Kintex™ UltraScale+™ devices provide the best price/performance/watt balance in a 16 nm 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.

AMD 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.

AMD Virtex™ UltraScale+™ devices provide the highest performance and integration capabilities in a 16 nm 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 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.

AMD Zynq™ UltraScale+™ 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.

The UltraScale architecture documentation suite is available at docs.amd.com.

CLB Overview

The Configurable Logic Block (CLB) is the main resource for implementing general-purpose combinatorial and sequential circuits. Synthesis tools automatically use the highly efficient logic, arithmetic, and memory features of the UltraScale architecture. These features can also be directly instantiated for greater control over the implementation. The CLB is made up of the logic elements themselves, which are grouped together in a slice, along with the interconnect routing resources to connect the logic elements. Utilization of slices, their placement in the device, and routing between them is best left to the automatic tools, but knowing the UltraScale architecture helps you create more optimal designs.

This user guide provides the necessary CLB details for designers to produce code that uses all of the CLB capabilities. This overview section introduces the most commonly used CLB resources. Later chapters include further detail for optimizing a design to improve density, performance, or power.

The UltraScale architecture CLBs provide advanced, high-performance, low-power programmable logic with:

- Real 6-input look-up table (LUT) capability.
- Dual LUT5 (5-input LUT) option.
- Distributed memory and shift register logic (SRL) ability.
- Dedicated high-speed carry logic for arithmetic functions.

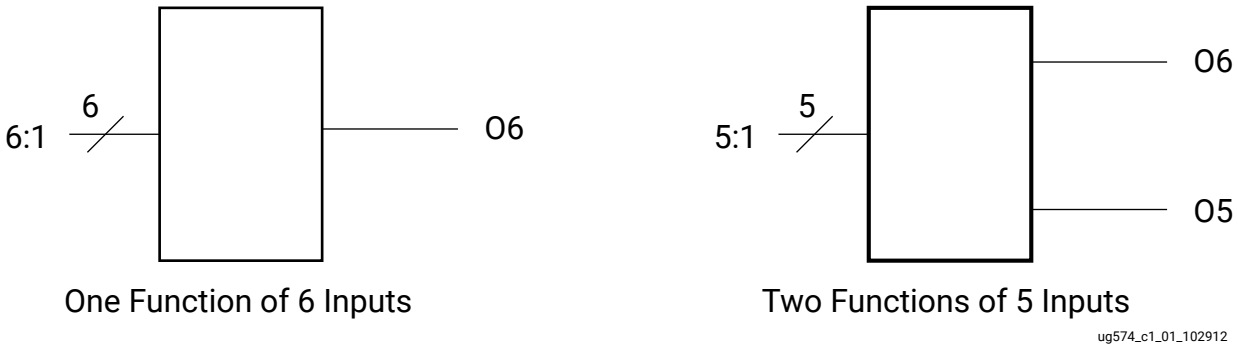
- Wide multiplexers for efficient utilization.
- Dedicated storage elements that can be configured as flip-flops or latches with flexible control signals.

Each UltraScale architecture CLB contains one slice. Each slice provides eight 6-input LUTs and sixteen flip-flops. Slices easily connect to each other to create larger functions. The slices and their CLBs are arranged in columns throughout the device, with the size and number of columns increasing with density.

The UltraScale architecture LUTs can be configured as shown in the following figure.

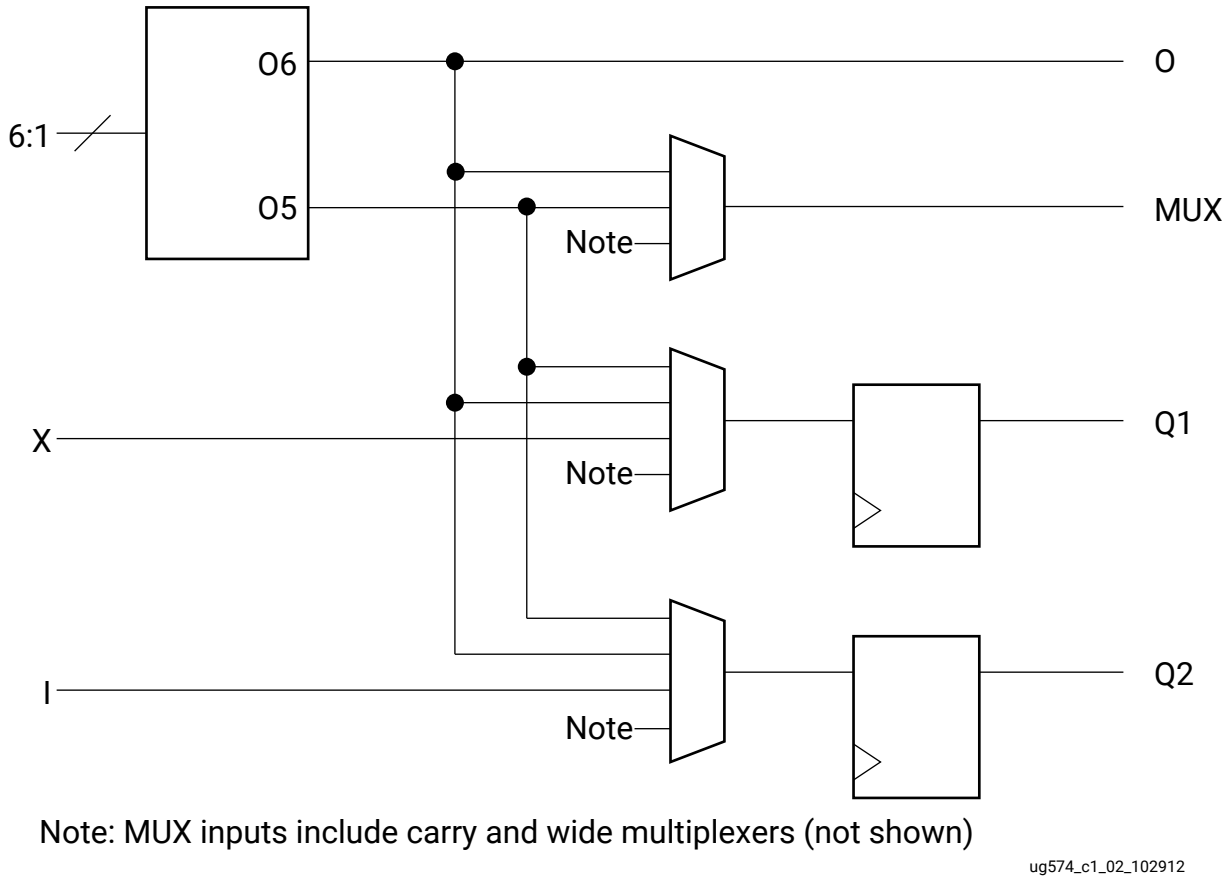
- A 6-input LUT with one output.
- Two 5-input LUTs with separate outputs but common addresses or logic inputs.

Figure 1: **LUT Configurations**



Each LUT output can connect to slice outputs, or optionally be registered in a flip-flop or latch. The storage elements can also be driven by direct inputs to the slice (X and I), or by the results of the internal carry logic or wide multiplexers. See the following figure. The storage elements have a clock enable input, along with an initialization signal that can be programmed as either synchronous or asynchronous, and as set or reset.

Figure 2: **A Simplified View of Slice I/O Connecting to a LUT and Storage Elements**

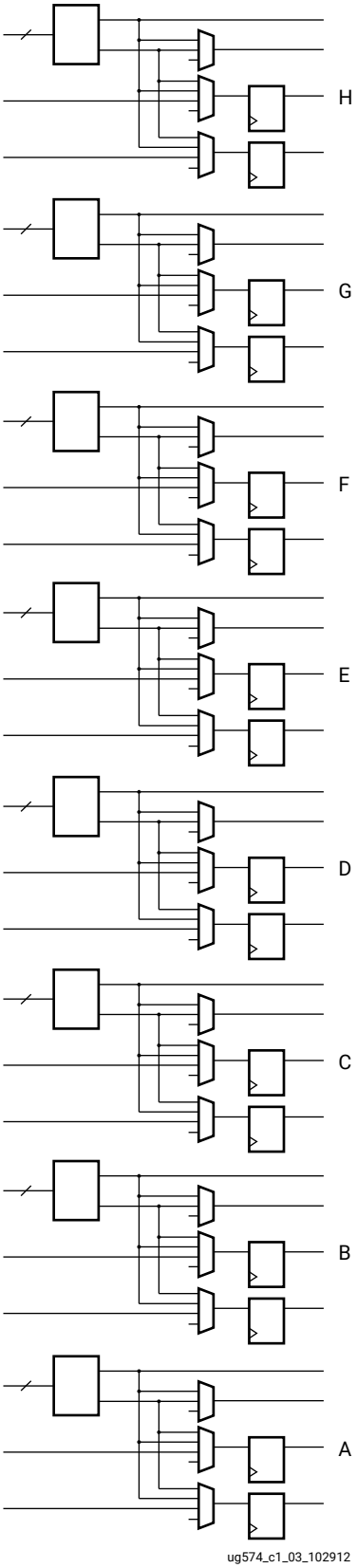


Dedicated carry logic improves the performance of arithmetic functions such as adders, counters, and multipliers. Carry logic consists of dedicated carry-lookahead gates, multiplexers, and routing that are independent of the general-purpose logic resources while providing both higher density and increased performance. Carry logic is often inferred for smaller arithmetic functions. Larger, more complex arithmetic functions can use the dedicated DSP block. See the *UltraScale Architecture DSP Slice User Guide* (UG579).

Each 6-input LUT can implement a 4:1 multiplexer (MUX). Dedicated multiplexers in the slices combine the LUTs together to create even wider functions without having to connect to another slice. All the LUTs in a slice can be combined together as a 32:1 MUX in one level of logic.

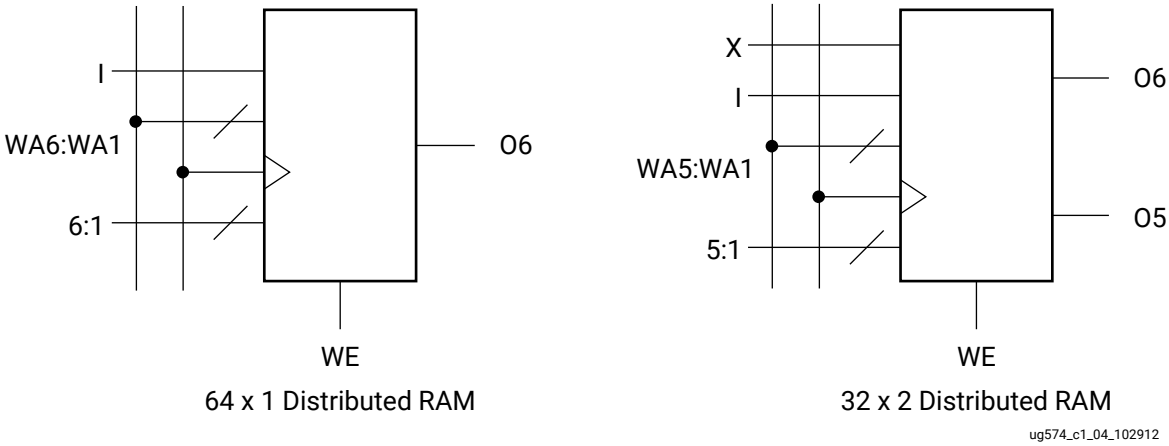
Eight 6-input LUTs and their sixteen storage elements, as well as the multiplexers and arithmetic carry logic, form a slice, as shown in the following figure. Each 6-input LUT and set of associated flip-flops and other logic are labeled A to H from bottom to top.

Figure 3: **LUTs and Storage Elements in One Slice**



There are two types of slices in the UltraScale architecture, with different ratios of the two types by device. The SLICEL (logic) has all the LUT and storage element resources shown, along with the carry logic and wide multiplexers. A SLICEM (memory) can also use the LUTs as distributed 64-bit RAM, by adding a separate write address (WA), write enable (WE), and clock signal. The LUT can be configured as either a 64 x 1 or 32 x 2 memory. The direct inputs X and I serve as the data inputs. See the following figure.

Figure 4: **Distributed RAM Using Look-Up Tables**



The distributed RAM can be combined across the eight LUTs in the SLICEM to create memories of up to 512 bits. The SLICEM shares a common write address and write clock across all 8 LUTs. The SLICEM write enable is also shared but can be used in combination with three other slice inputs for more flexibility. Multiple SLICEM can be combined together to create memories larger than 512 bits. Dedicated block RAM is also available for larger memories. See the *UltraScale Architecture Memory Resources User Guide* (UG573).

Each LUT in a SLICEM can also be used as a 32-bit shift register (SRL32). Combining the LUTs allows up to a 256-bit shift register in one SLICEM, compared to the 16 dedicated flip-flops per slice.

More details on each of these features are provided in Chapter 2: CLB Functionality.

Differences from Previous Generations

The UltraScale architecture CLB is very similar to that of the 7 series devices, with the same basic building blocks of 6-input LUTs, flexible storage elements, and abundant routing. Design techniques used for previous generations will continue to work effectively with the UltraScale architecture-based devices. The differences are primarily to provide more flexibility, which in turn allows for greater automatic optimization of designs.

- The two independent slices of the 7 series CLB are combined into one cohesive UltraScale architecture slice for greater logic and routing efficiency. Top and bottom halves of UltraScale architecture slices are each similar to a 7 series slice. With one slice per CLB in the UltraScale architecture, the total resources per CLB are similar.
- More control sets available for flexibility and clock gating:
  - Four clock enables per CLB slice.
  - Optional invert on set/reset signal.
- More flexibility for distributed RAM control:
  - Dedicated clock for distributed RAM.
  - Write enable independent of flip-flop clock enable.
  - Write enable input can be combined with three direct slice inputs to create up to eight independent write enables within the slice.
- All flip-flop outputs always available for enhanced packing and routing, with four outputs per LUT:
  - Direct LUT output.
  - Multiplexed combinatorial output.
  - Two equivalent flip-flop outputs.
- Flip-flops can all be edge-triggered D-type flops or level-sensitive latches. Selectable by top or bottom half.
- Carry logic expanded from 4 to 8 bits per CLB for faster arithmetic. One carry chain per CLB.

The common features in the CLB structure simplify design migration from the 7 series to the UltraScale architecture-based devices. For greater control signal flexibility and to benefit from the lack of slice-specific restrictions, remove mapping and placement constraints before implementing designs originally targeted to earlier devices.

The CLBs are arranged in columns. This columnar architecture enables different devices to have a mix of varying features that are optimized for different application domains. Through this innovation, AMD offers a larger selection of devices and enables designers to choose a device with the correct features and capabilities needed for a specific design implementation.

The columnar architecture eliminates traditional design barriers by:

- Eliminating geometric layout constraints such as dependencies between I/O count and logic array size.
- Enhancing on-chip power and ground distribution by allowing power and GND to be placed anywhere on the device.
- Allowing disparate silicon-based dedicated IP blocks to be scaled independent of each other and surrounding resources.

The architecture is subdivided into segmented clock regions (CRs). CRs differ from previous families because they are arranged in tiles and do not span half the width of a device. A CR contains CLBs, DSP slices, block RAMs, interconnect, and associated clocking. The height of a CR is 60 CLBs, 24 DSP slices, and 12 block RAMs with a horizontal clock spine (HCS) at its center. The HCS contains the horizontal routing and distribution resources, leaf clock buffers, clock network interconnections, and the root of the clock network. Clock buffers drive directly into the HCS. There are 52 I/Os per bank and four gigabit transceivers that are pitch matched to the CRs. A core column contains configuration, System Monitor, and PCIe® blocks to complete a basic device.

Device Resources

The CLB resources are scalable across all UltraScale architecture-based devices, and provide a common architecture that improves tool efficiency, IP implementation, and design migration. Migration between UltraScale architecture-based devices requires no CLB-based design changes.

Device capacity is often measured in terms of logic cells, which are the logical equivalent of a classic four-input LUT and a flip-flop. The UltraScale architecture CLB increases the effective capacity with its six-input LUT, abundant flip-flops and latches, carry logic, and the ability to create distributed RAM or shift registers in the SLICEM.

CLB Slice Resources

Refer to the feature summary tables in the *UltraScale Architecture and Product Data Sheet: Overview* (DS890) for the count of CLB slice resources, including CLB flip-flops, CLB LUTs, and distributed RAM, that are available in each UltraScale architecture-based device. The amount of distributed RAM can be used to calculate related resources. The maximum number of shift register bits is the number of distributed RAM bits divided by two, while the number of SLICEM is the distributed RAM bits divided by 512.

Recommended Design Flow

CLB resources are inferred for generic design logic and do not require instantiation. HDL or high-level synthesis (HLS) is sufficient to achieve an efficient implementation.

RECOMMENDED: *These coding practices should be considered when targeting UltraScale architecture CLBs.*

- CLB flip-flops have either a set or a reset. Do not use both set and reset on the same element.
- Flip-flops are abundant. Consider pipelining to improve performance.
- Control inputs are shared across multiple resources in a CLB. Minimize the number of unique control inputs required for a design. Control inputs include clock, clock enable, set/reset, and write enable.
- To efficiently implement shift registers in the LUTs, avoid resets on the shift registers.
- For small storage requirements, a 6-input LUT can be used as 64 x 1 memory.
- Standard arithmetic functions are effectively implemented using dedicated carry logic.

The recommended design flow:

1. Implement the design using preferred methodologies (HDL, HLS, IP, etc.).
2. Evaluate utilization reports to determine resources used. Check to ensure that arithmetic logic, distributed RAM, and SRL are used, when helpful.
3. Consider flip-flop usage.
    a. Pipeline for performance
    b. Use dedicated flip-flops at the outputs of dedicated resources (block RAM, DSP)
    c. Allow shift registers to use SRLs (avoid set/resets)
4. Minimize the use of set/resets. The flip-flops are automatically initialized every time the device is powered up.

Pin-out Planning

CLB usage has little effect on pin-outs because CLBs are distributed throughout the device. The column-based architecture provides maximum flexibility with CLBs on both sides of most I/Os. The best design practice is to let the tools choose the I/O locations based on the device requirements. Results can be adjusted, when necessary, for board layout considerations. Set the timing constraints to allow the tools to choose optimal placement based on the design requirements. Because the carry logic cascades vertically up a column, which is the only directionality in the CLB structure, the wide arithmetic buses might drive a vertical orientation to other logic, including I/O.

AMD

Chapter 2

CLB Functionality

Overview

This chapter provides a detailed view of the AMD UltraScale™ architecture CLB. These details are useful for design optimization and verification, but are not necessary for initiating a design. This chapter includes:

- CLB Resources: An overview of CLB slice features
- Look-Up Table: A description of the logical function generators.
- Storage Elements: A description of and controls for the latches and flip-flops.
- Multiplexers: Dedicated gates for combining LUTs into wide functions.
- Carry Logic: Dedicated gates and cascading to implement efficient arithmetic functions.
- Distributed RAM (SLICEM Only): Using the SLICEM LUTs as writable memory.
- Shift Registers (SLICEM Only): Using the SLICEM LUTs as shift registers.

CLB Resources

Every CLB contains one slice with eight 6-input LUTs and sixteen storage elements. The LUTs are organized as a column with an 8-bit carry chain per CLB, called CARRY8. Wide-function multiplexers combine LUTs to create any function of 7, 8, or 9 inputs, or some functions of up to 55 inputs. SLICEL is the name used to describe CLB slices that support these functions, where the L is for logic. The LUT in a SLICEM, where the M is for memory, can be configured as a look-up table, 64-bit distributed RAM, or a 32-bit shift register. The CLB for a SLICEL is referred to as a CLEL tile, and the CLB for the SLICEM is referred to as a CLE_M tile. The following table summarizes the resources in one CLB.

Table 1: **Logic Resources in One CLB Slice**

CLB Slice	LUTs	Flip-Flops	Arithmetic and Carry Chains	Wide Multiplexers	Distributed RAM	Shift Registers
SLICEL	8	16	1	F7, F8, F9	N/A	N/A

Table 1: **Logic Resources in One CLB Slice** *(cont'd)*

CLB Slice	LUTs	Flip-Flops	Arithmetic and Carry Chains	Wide Multiplexers	Distributed RAM	Shift Registers
SLICEM	8	16	1	F7, F8, F9	512 bits	256 bits

The LUTs are labeled A, B, C, D, E, F, G, and H from bottom to top. These designations are used to differentiate the signals around each LUT and the LUT's pair of storage elements. The letter designation is put in front of the signal or component name to differentiate it from the seven others in the CLB slice. For example, the LUT inputs are designated A1-A6, B1-B6, etc., the storage element outputs are AQ and AQ2, BQ and BQ2, etc.

Look-Up Table

The function generators are implemented as six-input look-up tables (LUTs). There are six independent inputs (1 to 6) and two independent outputs (O5 and O6) for each of the eight function generators in a CLB slice.

The function generators can implement:

- Any arbitrarily defined six-input Boolean function.
- Two arbitrarily defined five-input Boolean functions, as long as these two functions share common inputs.
- Two arbitrarily defined Boolean functions of three and two inputs or less.

A 6-input function uses:

- Inputs 1-6
- O6 output

Two 5-input or less functions use:

- Inputs 1-5
- The sixth input is driven High
- O5 and O6 outputs

The propagation delay through a LUT is independent of the function implemented.

Signals from function generators can connect to slice outputs or to other resources within the slice:

- Exit the slice through the direct O output for O6, or through the MUX multiplexed output for O5 or O6.

- Feed the D input of either of the two storage elements.
- Go to the carry logic.
  - Enter the XOR gate from an O6 output.
  - Enter the cascade chain from an O5 output.
  - Enter the select line of the cascade multiplexer from the O6 output.
- Go to F7MUX wide multiplexers from the O6 output.

Each slice contains seven multiplexers to create wider functions. These multiplexers are used to combine up to eight function generators to provide any function with up to nine inputs in a slice.

- F7MUX_AB, F7MUX_CD, F7MUX_EF, and F7MUX_GH: Used to generate any seven-input function from two adjacent LUTs.
- F8MUX_BOT and F8MUX_TOP: Used to generate any eight-input function from two adjacent F7MUXs.
- F9MUX: Used to generate any nine-input function from the two F8MUXs.

Functions with more than nine inputs can also be implemented using multiple CLBs.

Storage Elements

There are 16 storage elements per CLB slice (two per LUT). All can be configured as either edge-triggered D-type flip-flops or level-sensitive latches. The latch option is by top or bottom half of the device (A to D, and E to H). If the latch option is selected on a storage element, all eight storage elements in that half must be either used as latches or left unused. When configured as a latch, the latch is transparent when the CLB clock input (CLK) is High.

Design flexibility is enhanced because both LUT storage elements Q1 and Q2 have the same D input options:

- The LUT O6 output.
- The LUT O5 output.
- A direct input from the CLB inputs that bypasses the LUT (BYP). CLB X input for Q1, CLB I input for Q2.
- The carry XOR result.
- The carry cascade output (CO) at that bit.
- The local one of the seven wide multiplexers (FMUX). This is not available on the bottom LUT A.

Control Signals

There are two clock inputs (CLK) and two set/reset inputs (SR) to every CLB for the storage elements. Each clock or set/reset input is dedicated to eight of the sixteen storage elements, split by top and bottom halves (A to D, and E to H). For a given LUT, the two storage elements (Q1 and Q2) share the same clock and set/reset signals. The clock and set/reset signals have programmable polarity at their slice inputs, allowing any inversion to be automatically absorbed into the CLB.

Clock Enables

There are four clock enables (CE) per slice. The clock enables are split both by top and bottom halves, and by the two storage elements per LUT. Thus, the CEs are independent for:

- AQ1, BQ1, CQ1, and DQ1
- AQ2, BQ2, CQ2, and DQ2
- EQ1, FQ1, GQ1, and HQ1
- EQ2, FQ2, GQ2, and HQ2

This gives the most flexibility for clock gating for design efficiency and power reduction. When one storage element has CE enabled, the other three storage elements in the group must also have CE enabled. The CE is always active-High at the slice, but can be inverted in the source logic.

Initialization

The two SR set/reset inputs to a slice can be programmed to be synchronous or asynchronous. The set/reset signal can be programmed to be a set or reset, but not both, for any individual storage element.

The configuration options for the SR set and reset functionality of a register or latch are:

- No set or reset
- Synchronous set (FDSE primitive)
- Synchronous reset (FDRE primitive)
- Asynchronous set (preset) (FDPE primitive)
- Asynchronous reset (clear) (FDCE primitive)

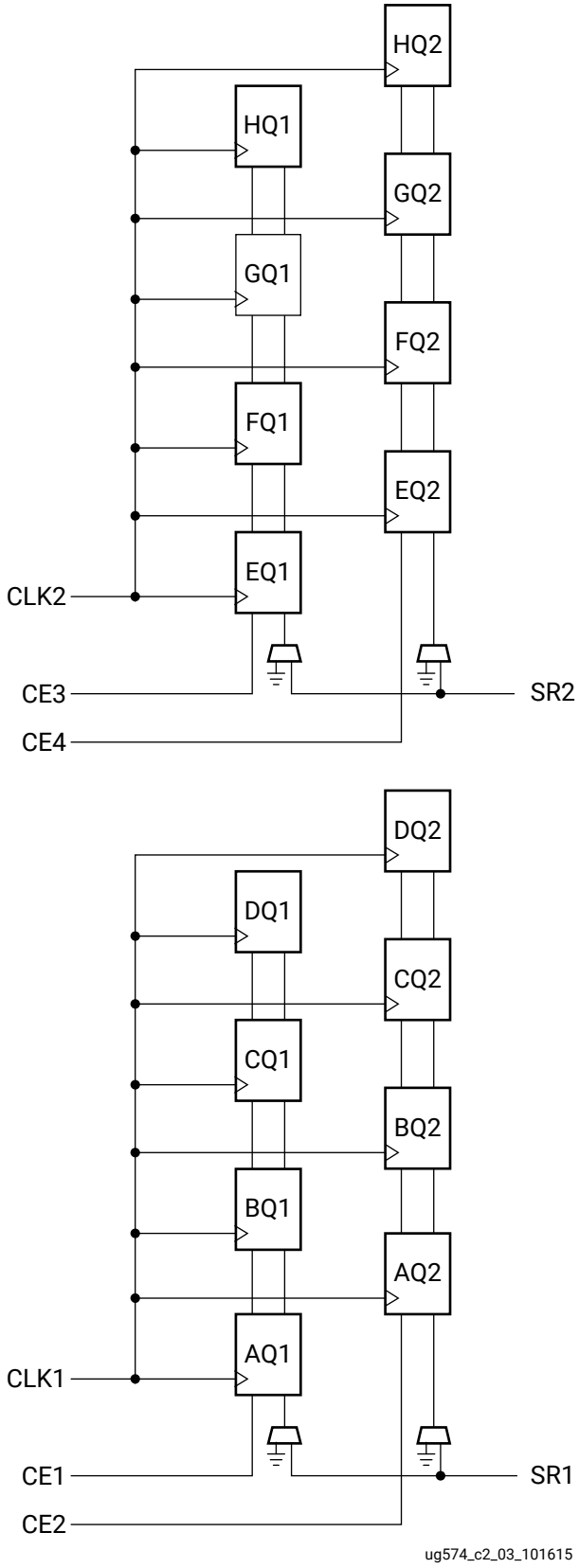
The SR set/reset input can be ignored for groups of four flip-flops (the same groups as controlled by the CE inputs). When one storage element has SR enabled, the other three storage elements in the group must also have SR enabled.

The choice of set or reset can be controlled individually for each storage element in a slice. The choice of synchronous (SYNC) or asynchronous (ASYNC) set/reset (SYNC_ATTR) is controlled in groups of eight flip-flops, individually for the two separate SR inputs.

The initial state after configuration is defined with a separate INIT attribute that can be specified as 0 or 1. By default, defining the SR as a set defines INIT=1, and defining the SR as a reset defines INIT=0. INIT can be defined independent of the SR function.

The following figure shows how the control signals connect to the storage elements in a slice.

Figure 5: **Control Signal Connections to the Storage Elements in a Slice**



Multiplexers

Function generators and associated multiplexers in the UltraScale architecture slice can implement:

- 4:1 multiplexers using one LUT, providing eight 4:1 multiplexers per CLB.
- 8:1 multiplexers using two LUTs, providing four 8:1 multiplexers per CLB.
- 16:1 multiplexers using four LUTs, providing two 16:1 multiplexers per CLB.
- 32:1 multiplexers using eight LUTs, providing one 32:1 multiplexer per CLB.

The wide-input multiplexers are implemented in one level of logic (or LUT) using dedicated multiplexers:

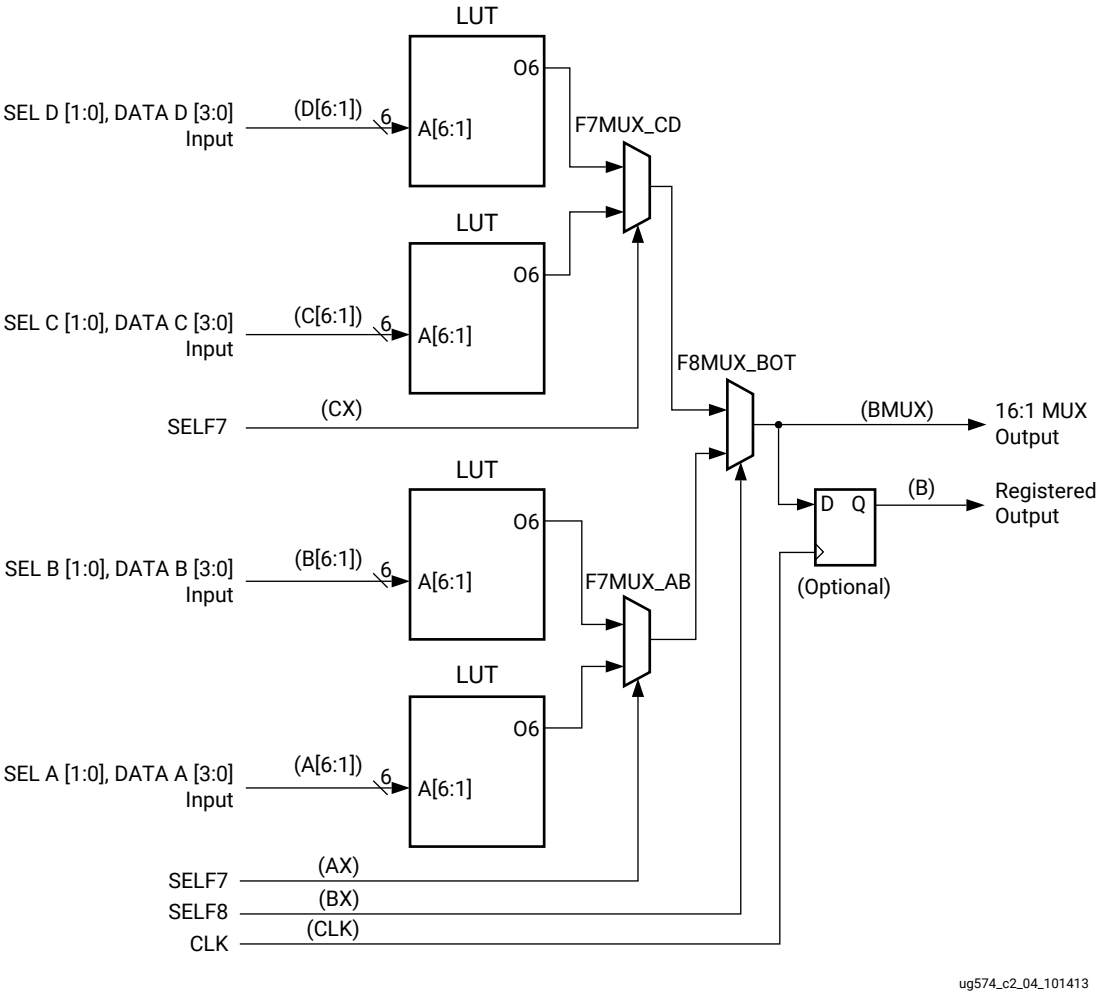
- F7MUX_AB, F7MUX_CD, F7MUX_EF, and F7MUX_GH combine adjacent LUTs.
- F8MUX_BOT and F8MUX_TOP combine F7MUX outputs.
- F9MUX combines the two F8MUX outputs.

These multiplexers allow LUT combinations of up to eight LUTs in a CLB. The wide multiplexers can also be used to create general-purpose functions of up to 13 inputs in two LUTs or 27 inputs in four LUTs. Although the multiplexer outputs are combinatorial, they can be registered in the CLB storage elements.

16:1 Multiplexer

The F8MUX combines the outputs of F7MUX_AB and F7MUX_CD, or F7MUX_EF and F7MUX_GH to form a combinatorial function up to 27 inputs (or a 16:1 MUX). A 16:1 MUX requires only half of a CLB slice, as shown in the following figure.

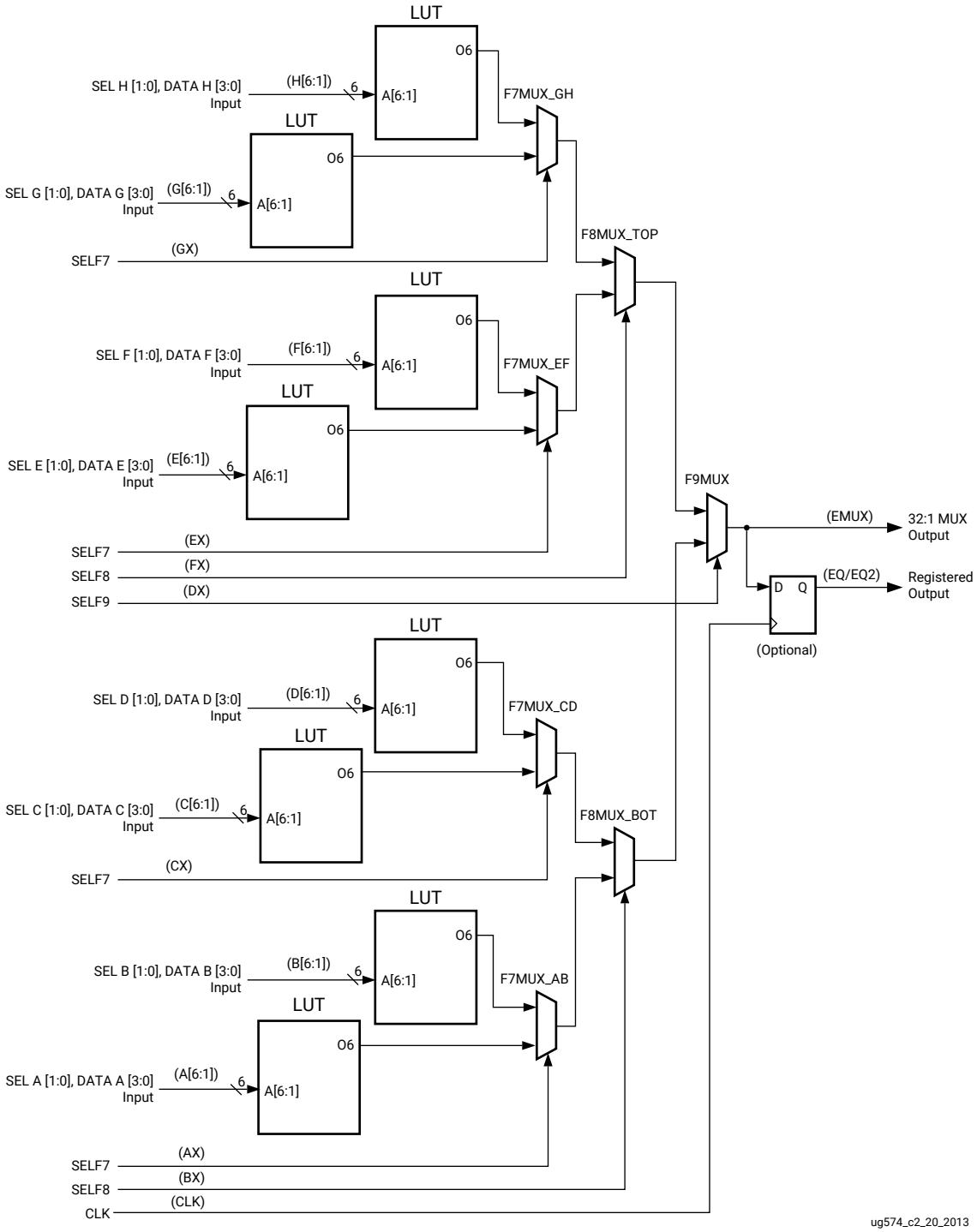
Figure 6: **16:1 Multiplexer in Half a CLB Slice**



32:1 Multiplexer

The F9MUX combines the outputs of F8MUX_BOT and F8MUX_TOP to form a 32:1 MUX in one CLB slice (see the following figure).

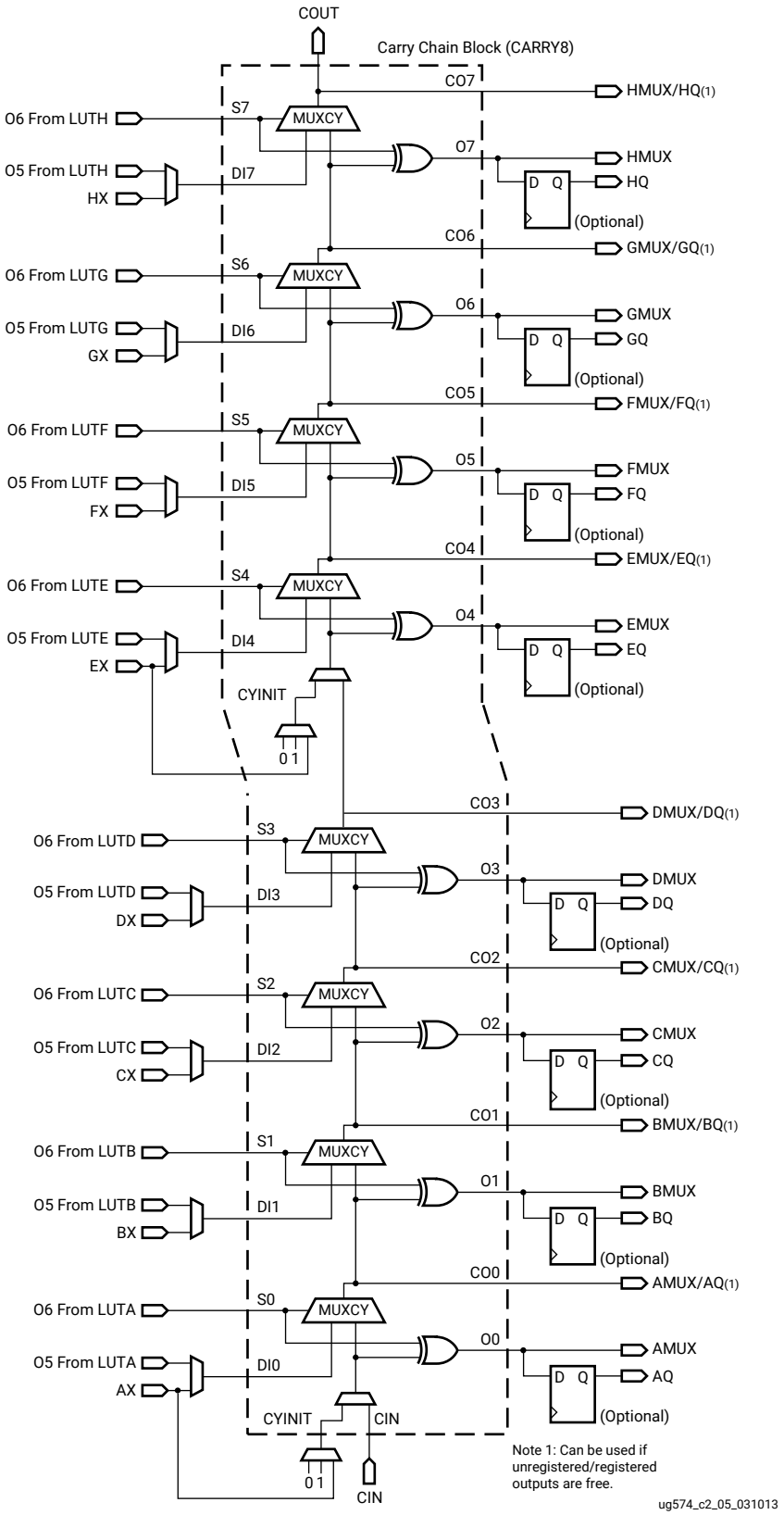
Figure 7: **32:1 Multiplexer in One CLB Slice**



Carry Logic

In addition to function generators, dedicated fast lookahead carry logic is provided to perform fast arithmetic addition and subtraction. A CLB slice has a single carry chain, as shown in the following figure. The carry chains are cascaded to form wider add/subtract logic.

Figure 8: **Fast Carry Logic Path and Associated Elements**



The carry chain runs upward and has a height of eight bits per CLB slice. The carry initialize input CYINIT is used to select the first bit in a carry chain. The value for this input is either 0 (for add), 1 (for subtract), or AX input (for the dynamic first carry bit). The carry initialization function is available both at the beginning of the chain at the bottom of the CLB slice, and at the mid-point, for splitting the slice into two 4-bit carry blocks. Dedicated connections cascade a carry from the COUT pin of one slice to the CIN pin of the slice above. For each bit, there is a carry multiplexer (MUXCY) and a dedicated XOR gate for adding/subtracting the operands with selected carry bits. The dedicated carry path and carry multiplexer (MUXCY) can also be used to cascade function generators for implementing wide logic functions.

Distributed RAM (SLICEM Only)

The function generators (LUTs) in SLICEM can be implemented as a synchronous RAM resource also described as distributed RAM. Multiple LUTs in a SLICEM can be combined in various ways to store larger amounts of data up to 512 bits per SLICEM. Multiple SLICEMs can be combined to create larger memories. RAM elements are configurable within a SLICEM to implement these configurations:

- Single-port 32 x (1 to 16)-bit RAM
- Dual-port 32 x (1 to 8)-bit RAM
- Quad-port 32 x (1 to 4)-bit RAM
- Simple dual-port 32 x (1 to 14)-bit RAM
- Single-port 64 x (1 to 8)-bit RAM
- Dual-port 64 x (1 to 4)-bit RAM
- Quad-port 64 x (1 to 2)-bit RAM
- Octal-port 64 x 1-bit RAM
- Simple dual-port 64 x (1 to 7)-bit RAM
- Single-port 128 x (1 to 4)-bit RAM
- Dual-port 128 x 2-bit RAM
- Quad-port 128 x 1-bit RAM
- Single-port 256 x (1 to 2)-bit RAM
- Dual-port 256 x 1-bit RAM
- Single-port 512 x 1-bit RAM

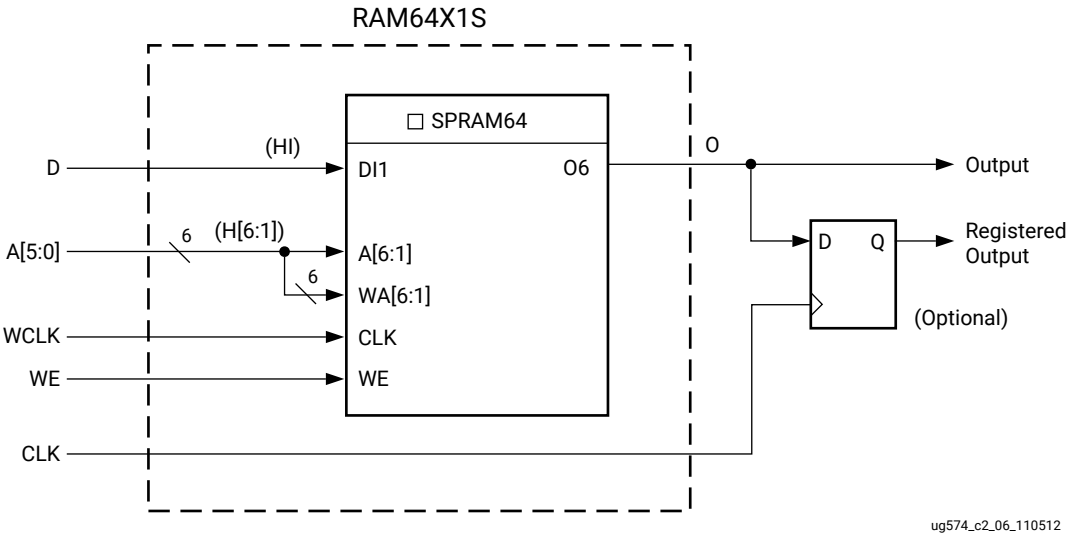
Distributed RAM modules are synchronous (write) resources. The write clock is from a dedicated SLICEM input LCLK that is independent of the two clocks for the storage elements. For a write operation, the write enable (WE) input must be set High. Read is asynchronous by default. A synchronous read can be implemented with a flip-flop in the same SLICEM. By using this flip-flop, the distributed RAM performance is improved because the clock-to-out delay is decreased. However, an additional clock latency is added.

Distributed RAM configurations include:

- **Single-port:** A common address port for synchronous writes and asynchronous reads. Read and write addresses share the same address bus. A single-port RAM is defined by the LUT configuration of SPRAM32 for a 32x1 RAM with five address inputs, or SPRAM64 for a 64x1 RAM with six address inputs. Two SPRAM32 with common address inputs can be combined in the same LUT by tying the sixth address line High and using the O5 and O6 outputs independently.
- **Dual-port:**
  - One port for synchronous writes and asynchronous reads. One function generator is connected with the shared read and write port address.
  - One port for asynchronous reads. Second function generator has the address inputs connected to a second read-only port address, and the write address (WA) inputs are shared with the first read/write port address. Quad-port and octal-port configurations add additional function generators as asynchronous read ports.
- **Simple dual-port:**
  - One port for synchronous writes (no data out/read port from the write port).
  - One port for asynchronous reads.
  - A dual-port RAM is defined by the LUT configuration of DPRAM32 for a 32x1 RAM with five address inputs, or DPRAM64 for a 64x1 RAM with 6 address inputs.

An example 64 x 1 distributed RAM using a single H LUT is illustrated in the following figure. As shown in the figure, the common WA port is always physically driven by the inputs to the H LUT using H6-H1. The read ports are independent of each other for each of the eight LUTs with the H LUT using a common set of inputs for both write and read. Therefore, the H LUT is always effectively a single-port memory, even if DPRAM64 is selected for the LUT configuration. The other LUTs are always effectively dual-port memories, although SPRAM32 can be selected when the read and write addresses are connected together.

Figure 9: **Distributed RAM (RAM64X1S)**

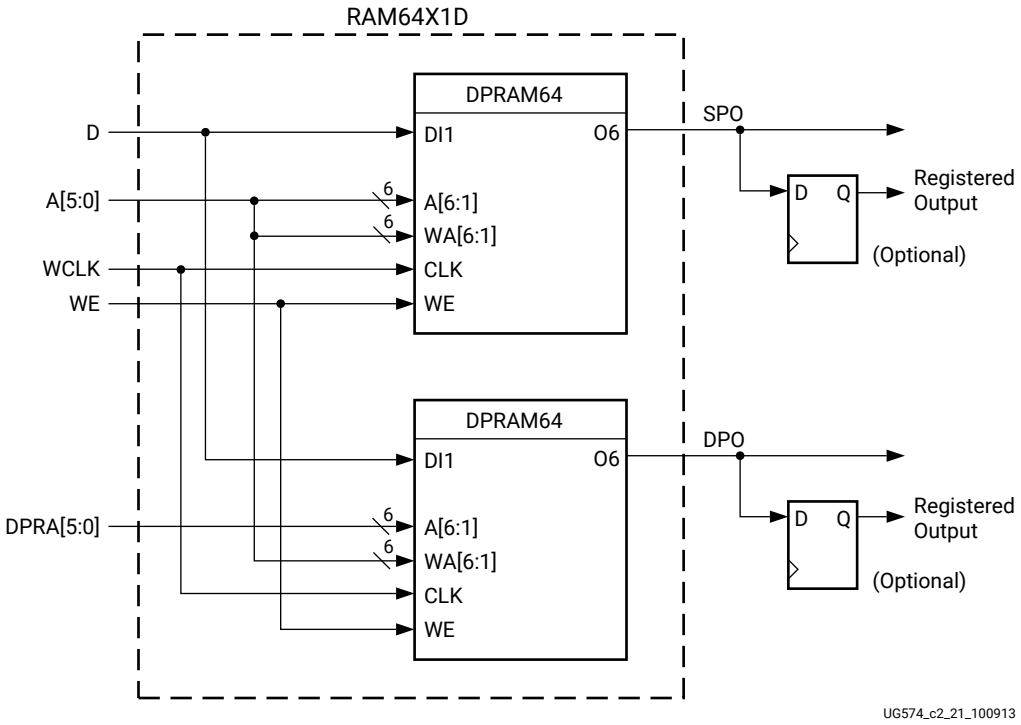


If eight single-port 64 x 1-bit modules are each built as shown in the previous figure (and the eight share the same clock, write enable, and shared read and write port address inputs), then the eight RAM64X1S primitives can occupy a single SLICEM. This configuration equates to a 64 x 8-bit single-port distributed RAM.

However, if any of the LUTs are configured as distributed RAM within a SLICEM, then the other LUTs cannot be used as SRLs. Because the write address port always uses the H LUT, distributed RAM will be placed starting at the top of the slice.

If four dual-port 64 x 1-bit modules are each built as shown in the following figure, the four RAM64X1D primitives can occupy a SLICEM, as long as they share the same clock and shared read and write port address inputs. This configuration equates to a 64 x 4-bit dual-port distributed RAM.

Figure 10: **64x1 Dual-Port Distributed RAM (RAM64X1D)**



With eight LUTs per slice, the dual-port configuration can be extended up an octal-port 64x1 memory, as shown in the following figure. Alternatively, a simple dual-port configuration can be extended up to a 64x7 memory in one slice, with one dedicated write port and seven read ports, as shown in Figure 12: 64x7 Simple Dual-Port Distributed RAM.

Figure 11: **64x1 Octal Port Distributed RAM**

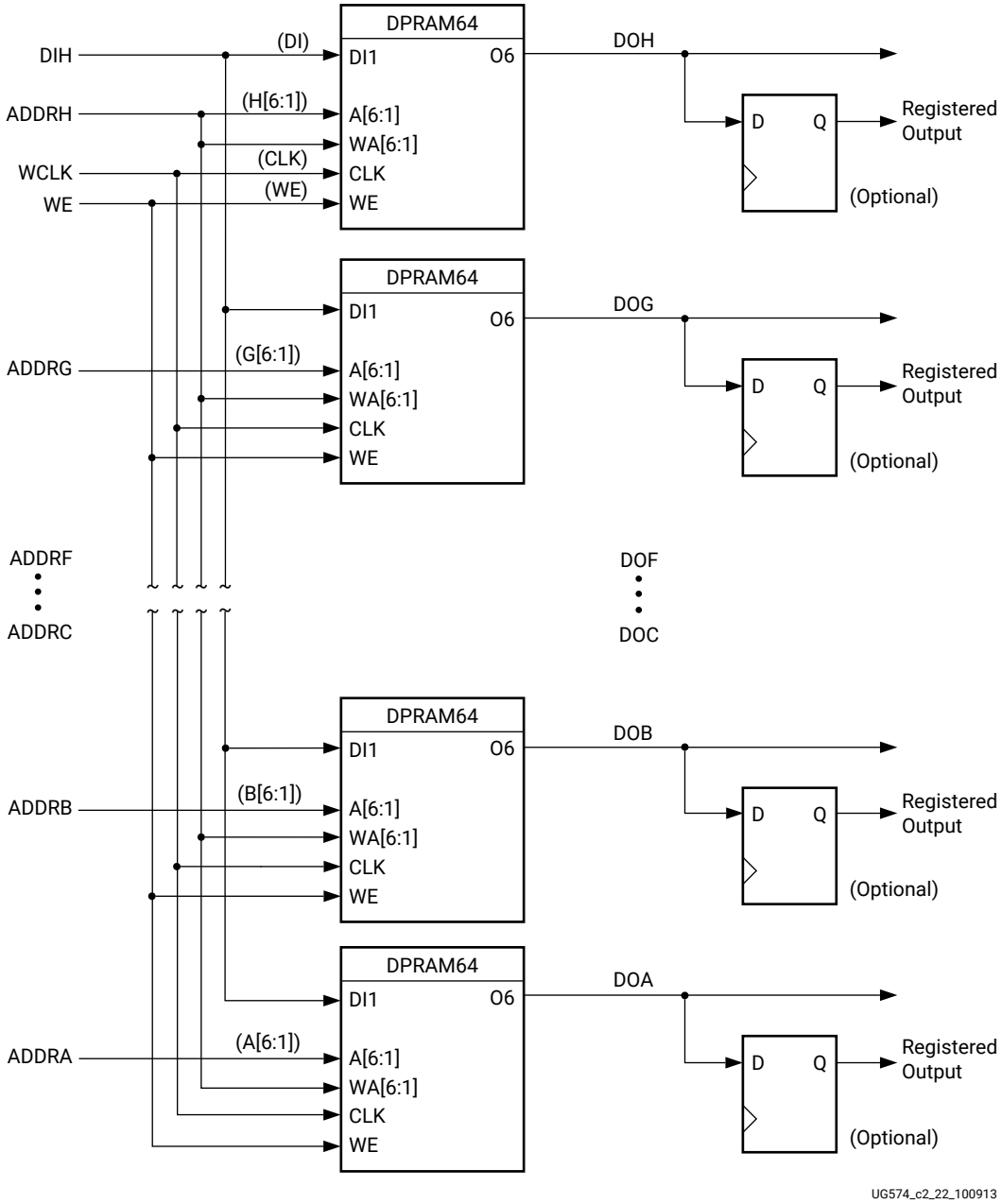
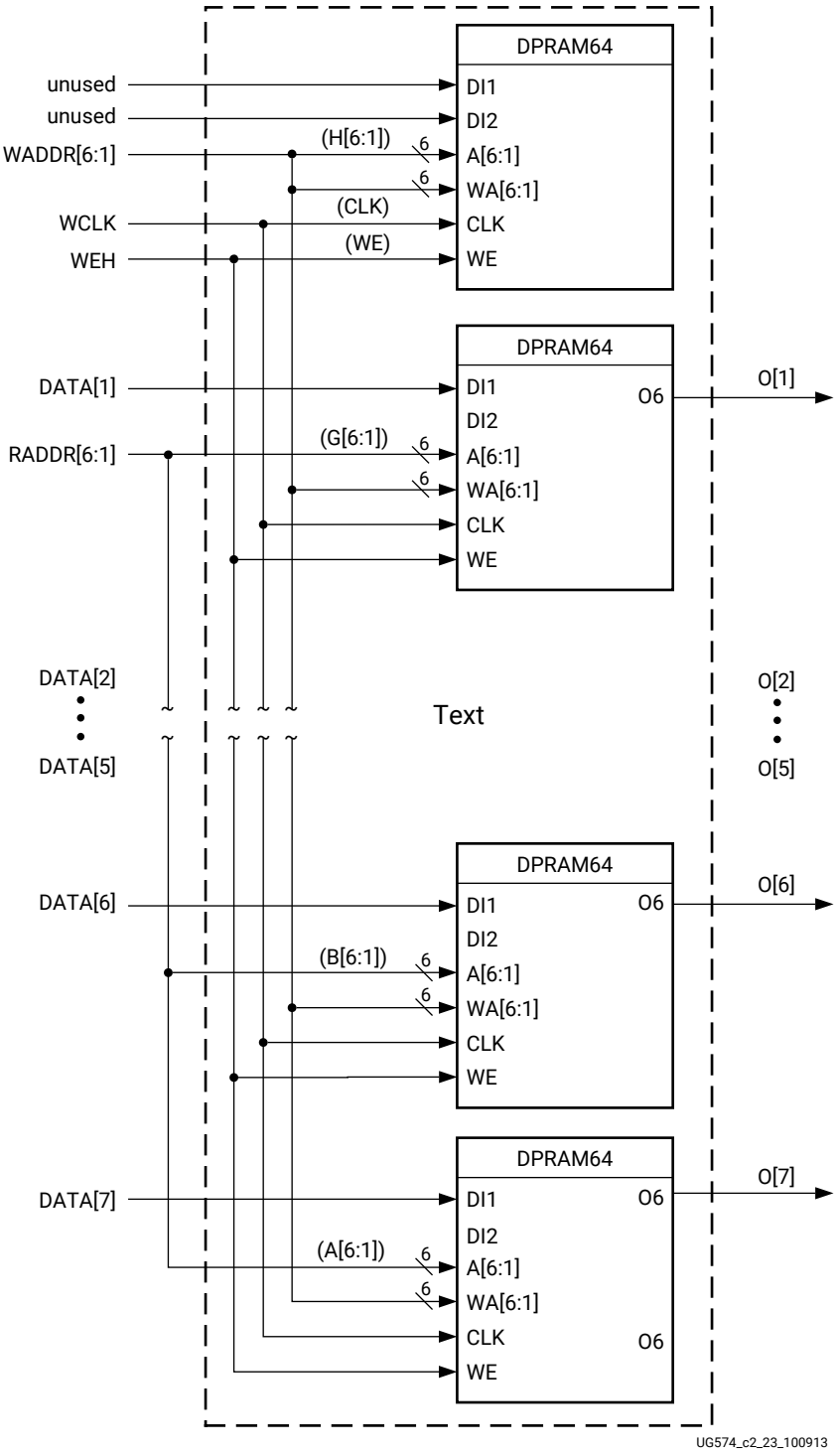
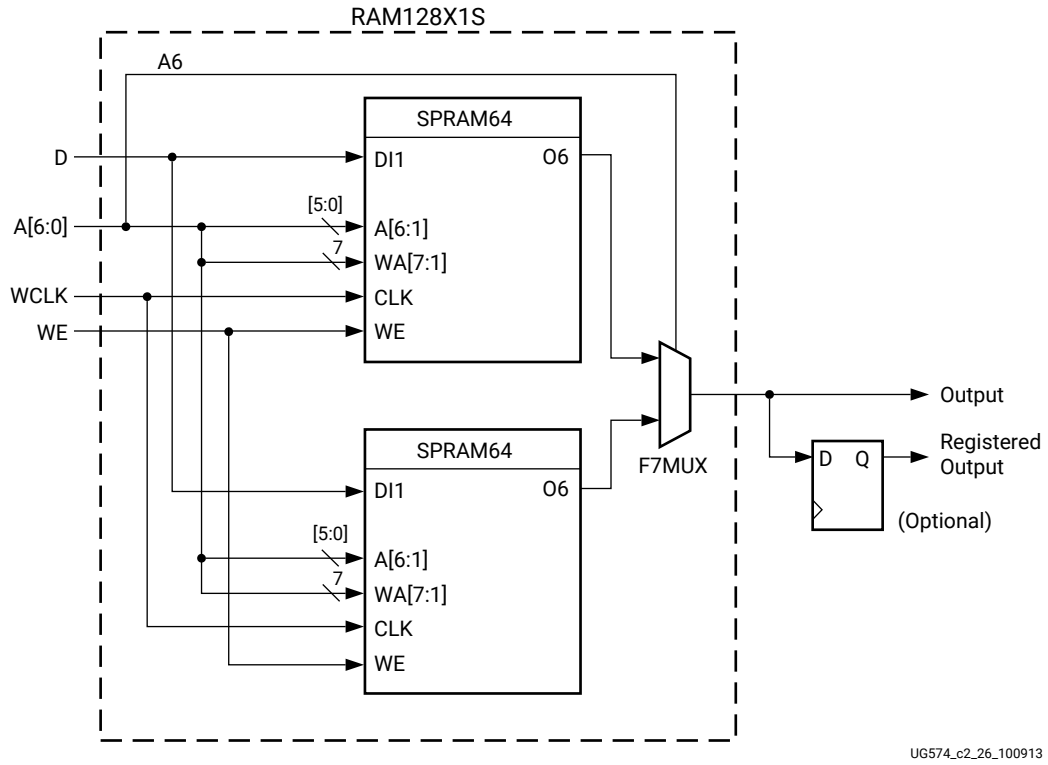


Figure 12: **64x7 Simple Dual-Port Distributed RAM**



Implementation of distributed RAM configurations with depth greater than 64 requires the usage of wide-function multiplexers, as shown in the following figure. If four single-port 128 x 1-bit modules are each built as shown in the figure, the four RAM128X1S primitives can occupy a SLICEM, as long as they share the same clock and shared read and write port address inputs. This configuration equates to 128 x 4-bit single-port distributed RAM.

Figure 13: **128x1 Single-Port Distributed RAM (RAM128X1S)**



Distributed RAM Data Flow

Synchronous Write Operation

The synchronous write operation is a single clock-edge operation with an active-High write-enable (WE) feature. When WE is High, the input (D) is loaded into the memory location at address A.

Asynchronous Read Operation

The output is determined by the address A for the single-port mode output SPO of dual-port mode, or address DPRA for the DPO output of dual-port mode. Each time a new address is applied to the address pins, the data value in the memory location of that address is available on the output after the time delay to access the LUT. This operation is asynchronous and independent of the clock signal.

Distributed RAM Summary

Distributed RAM features include:

- Single-port and dual-port modes are available in the SLICEM. Each LUT provides 64 bits of RAM, and each slice provides up to 256 bits of dual-port RAM or up to 512 bits of single-port RAM.
- A write operation requires one clock edge, using a separate LCLK input from the flip-flops.. The data input has a setup-to-clock timing specification.
- Read operations are asynchronous. Outputs can be registered within the SLICEM using the standard flip-flop.

Read Only Memory (ROM)

Each function generator in both SLICEM and SLICEL can implement a 64 x 1-bit ROM. Four configurations are available: ROM64x1, ROM128x1, ROM256x1, and ROM512x1. ROM contents are loaded at each device configuration. the following table shows the number of LUTs occupied by each ROM configuration size.

Table 2: **ROM Configuration**

ROM	Number of LUTs
64 x 1	1
128 x 1	2
256 x 1	4
512 x 1	8

Address Collision

Unlike dual-ported synchronous RAMs, where there can be non-deterministic result at the synchronously captured read output from a collision between two writes or a read and write to the same address during the same cycle, distributed RAM read outputs are of combinatorial logic and thus do not have an synchronous “address collision” mechanism. A LUTRAM read port combinatorically decodes the value at the address. If a different value is written to the read address, the read output propagates that written value to the read output within normal propagation delay time.

Shift Registers (SLICEM Only)

A SLICEM function generator can also be configured as a 32-bit shift register without using the flip-flops. When used in this manner, each LUT can delay serial data from one to 32 clock cycles. The shiftin D (DI1 LUT pin) and shiftout Q31 (MC31 LUT pin) lines cascade LUTs to form larger shift registers. The eight LUTs in a SLICEM are cascaded to produce delays of up to 256 clock cycles. It is also possible to combine shift registers across more than one SLICEM. The resulting programmable delays can be used to balance the timing of data pipelines.

Applications for shift registers include:

- Delay or latency compensation.
- Synchronous FIFO and content addressable memory (CAM).

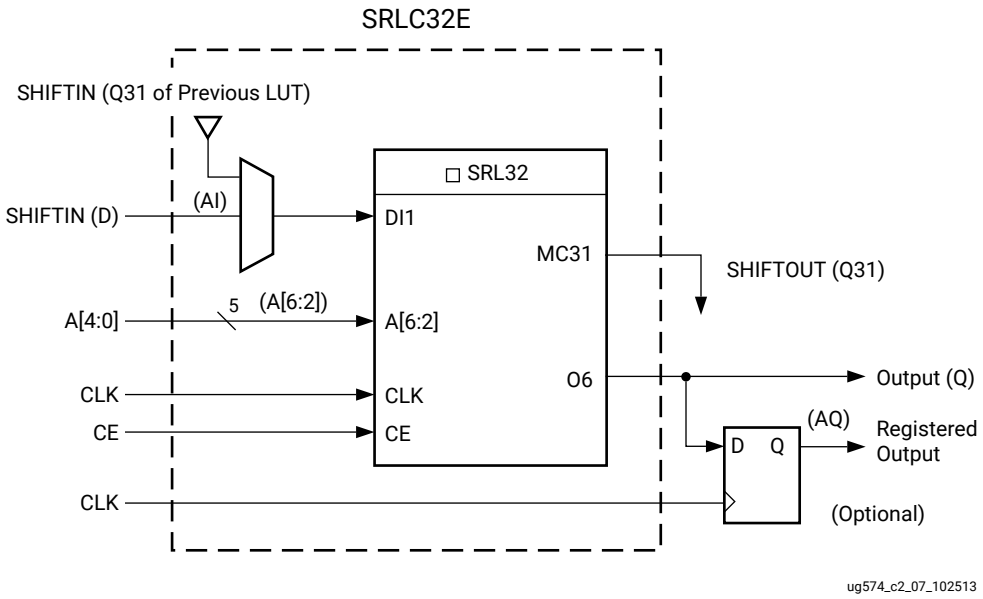
Shift register functions include:

- Write operation: synchronous with a clock input and an optional clock enable.
- Fixed read access to Q31 for cascading to the LUT below.
  - Q31 of bottom LUT A connects to SLICEM output for direct use or cascading to next SLICEM.
- Dynamic read access:
  - Performed through the 5-bit address bus, `A[4:0]`. The LSB of the LUT address is unused and the tools automatically tie it to a logic High.
  - Any of the 32 bits can be read out asynchronously (at the O6 LUT output) by varying the address.
    - This capability is useful in creating smaller shift registers (less than 32 bits).
    - For example, when building a 13-bit shift register, simply set the address to the 13th bit.

- A storage element or flip-flop is available to implement a synchronous read.
  - The clock-to-out of the flip-flop determines the overall delay and improves performance.
  - One additional cycle of clock latency is added.
- Set or reset of the shift register is not supported, but it can be initialized to any value after configuration.

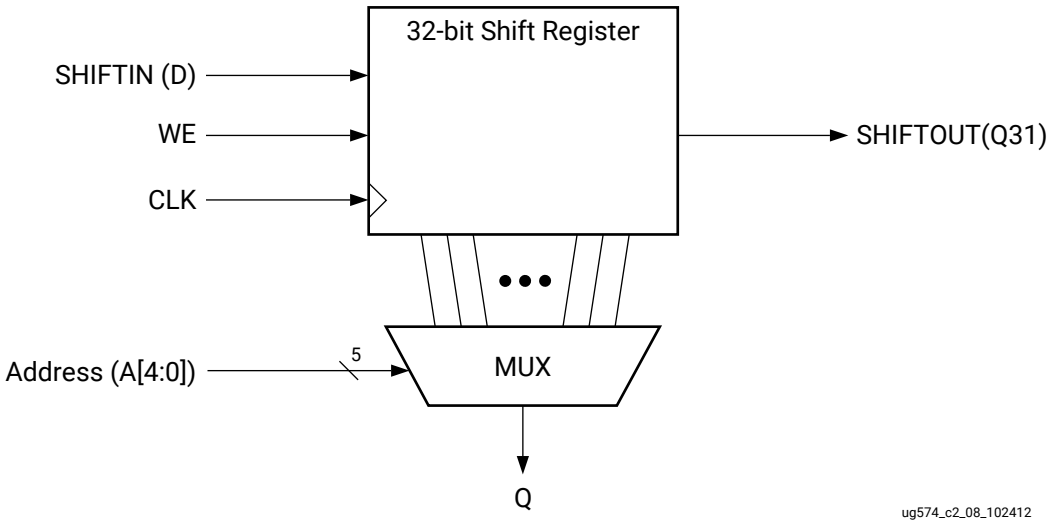
The following figure is a logic block diagram of a 32-bit shift register.

Figure 14: **32-bit Shift Register Configuration**



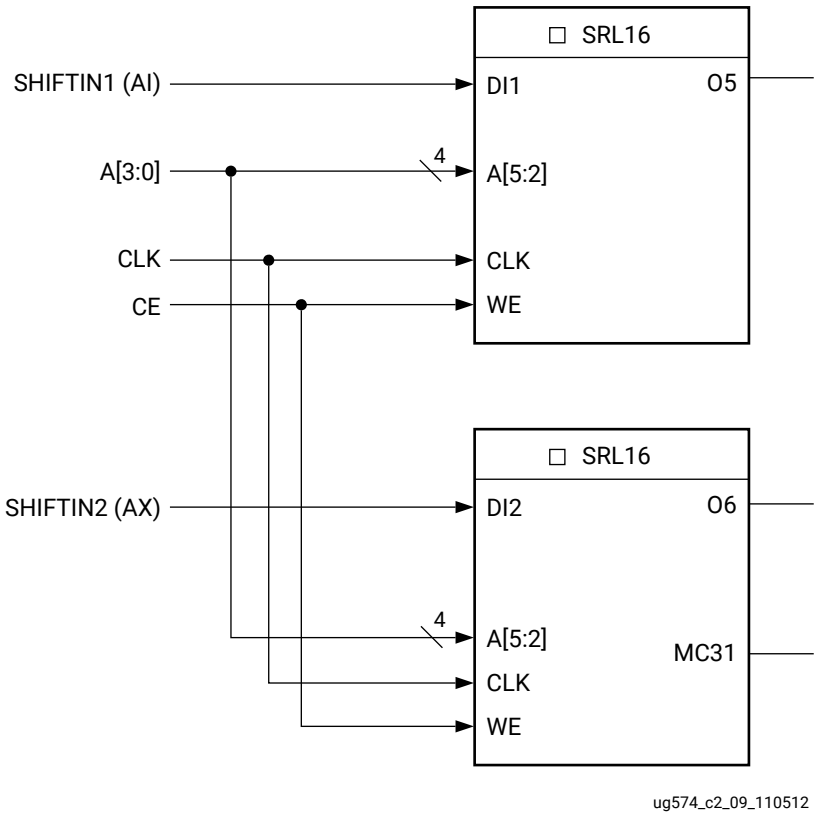
The following figure illustrates an example shift register configuration occupying one function generator.

Figure 15: **Representation of a Shift Register**



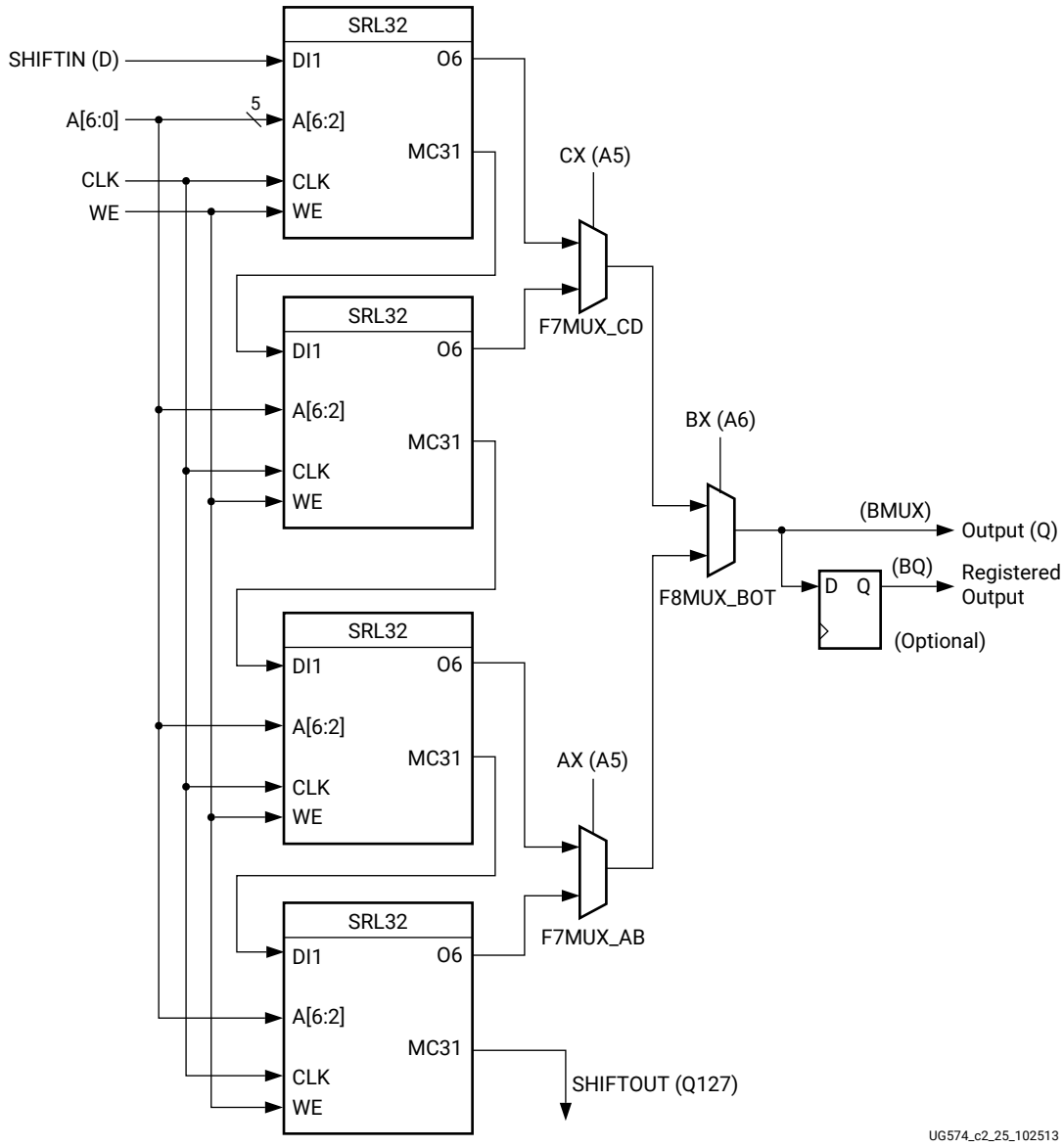
The following figure shows two 16-bit shift registers. This example can be implemented in a single LUT.

Figure 16: **Dual 16-bit Shift Register Configuration**



The MC31 output and a dedicated connection between shift registers allows connecting the last bit of one shift register to the first bit of the next, without using the LUT O6 output. Longer shift registers can be built with dynamic access to any bit in the chain. The shift register chaining and the wide multiplexers allow up to a 256-bit shift register with addressable access to be implemented in one SLICEM. The following figure illustrates a 128-bit shift register than can occupy one half of a SLICEM.

Figure 17: **128-bit Shift Register Configuration**



Shift Register Summary

- A shift operation requires one clock edge.
- Dynamic-length read operations to the Q output of the LUT are asynchronous.
- Static-length read operations to the Q output of the LUT are synchronous.
- The data input has a setup-to-clock timing specification.

- In a cascaded configuration, the Q31 output always contains the last bit value.

Design Entry

Introduction

CLB slice resources are used automatically and efficiently by synthesis tools without any special architecture-specific coding required. Some HDL coding suggestions and techniques can help optimize designs for maximum efficiency.

Design Checklist

These guidelines are provided as a quick checklist of design suggestions for effective use of the AMD UltraScale™ architecture CLB slices:

- **Resource Utilization:**
  - Use generic HDL code and allow the synthesis and mapping tools to choose the specific CLB resources.
  - Consider instantiation of specific resources only when necessary to meet density or performance requirements.
  - Compare the results to an estimated slice count to verify design efficiency.
  - If a design is running out of resources in the target device, examine which resource is the limiting factor and consider using alternative resources, such as moving registers to SRLs or distributed RAM, or distributed RAM to block RAM, or carry logic to DSP slices.
  - When migrating a design from another architecture, remove resource instantiation and any mapping and floorplanning constraints, refer to the *UltraScale Architecture Migration: Methodology Guide* (UG1026)
  - Refer to the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).
- **Pipelining:** Use sequential design techniques and pipelining to take advantage of abundant flip-flops for performance.
- **Control Signals:**
  - Use control signals only as necessary.

  - Avoid using a routed global reset signal and minimize use of local resets.
  - Use active-High control signals.
  - Avoid having both set and reset on the same flip-flop.
  - Avoid control signals on small shift registers and storage arrays to use LUTs instead of flip-flops, to maximize utilization and minimize power.
- **Implementation Options:** To improve performance automatically, use timing constraints and trade off longer implementation run times through tool options.

Using the CLB Slice Resources

AMD recommends using generic HDL code and allowing the tools to infer the usage of CLB resources. Using IP solutions designed for the UltraScale architecture can help make full use of the CLB resources. Although any feature in the CLB can be instantiated directly, including the LUTs, carry logic, and sequential elements, instantiation should be reserved primarily for specifying when resources outside the CLB should be used, such as the DSP slice. Instantiation might also be needed if the synthesis tool does not infer a desired special CLB resource, such as the wide multiplexers, distributed RAM, or SRL feature.

Primitives

This section provides an overview of the most commonly used CLB primitives. For instantiation templates, see the Language Templates in the AMD Vivado™ Design Suite. For more details on all primitives, see the *UltraScale Architecture Libraries Guide* (UG974).

LUT Primitives

The LUT primitives allow direct specification of the Look-Up Table configurations and locations in either the SLICEL or the SLICEM. The superset primitive is the LUT6_2, which provides access to all six LUT inputs and both of the O5 and O6 outputs (the following figure). The LUT6_2 can act as a dual asynchronous 32-bit ROM (with 5-bit addressing), implement any two 5-input logic functions with shared inputs, or implement a 6-input logic function and a 5-input logic function with shared inputs and shared logic values.

Figure 18: **LUT6_2 Primitive**

Other LUT primitives includes the LUT6, providing just the O6 output, LUT5, providing a 5-input LUT, and LUT4, LUT3, LUT2, and LUT1 providing the designated number of inputs in a single LUT.

Specify an INIT attribute consisting of a hexadecimal value to indicate the LUT's logical function. The LSB of the INIT value corresponds to the output when all zeroes are applied, and the MSB for all ones. For example, a LUT6_2 or LUT6 requires a 64-bit hex value. A Verilog INIT value of `64'hffffffffffffffe (X"FFFFFFFFFFFFFFFE"` for VHDL) makes the O6 output 1 unless all zeroes are on the `I[5:0]` inputs (a 6-input OR gate). For the LUT6_2, the lower half bits 31:0 of the INIT value apply to the logic function of the O5 output. In the same example, the O5 output is a 1 unless `I[4:0]` are all zeroes (a 5-input OR gate).

Multiplexer Primitives

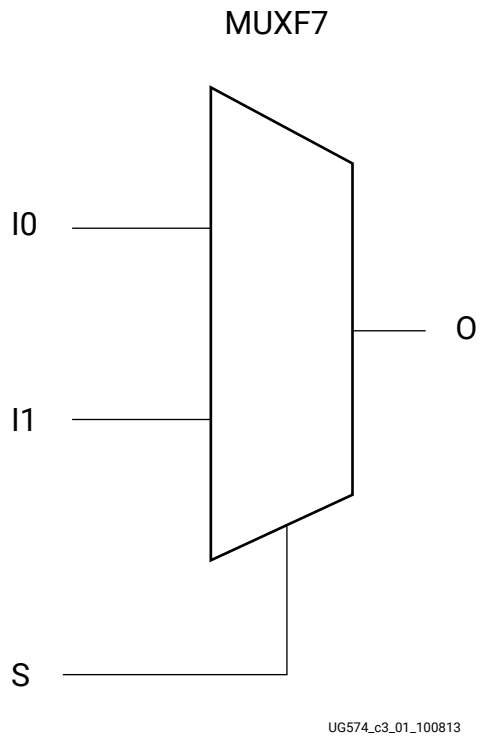
The multiplexer primitives provide direct instantiation of the dedicated multiplexers in each slice, allowing wider multiplexers to be built. The follliwng table describes the primitives.

Table 3: **Multiplexer Primitives**

Primitive	Inputs	Resource	Output Function
MUXF7	LUT outputs (4:1 Mux)	F7MUX_AB, F7MUX_CD, F7MUX_EF, F7MUX_GH	8:1 Mux
MUXF8	F7MUX_AB and F7MUX_CD, or F7MUX_EF and F7MUX_GH outputs (8:1 mux)	F8MUX_BOT, F8MUX_TOP	16:1 Mux
MUXF9	F8MUX_BOT and F8MUX_TOP (16:1 Mux)	F9MUX	32:1 Mux

The signals are the same for all multiplexer primitives. The following figure shows MUXF7.

Figure 19: **MUXF7 Primitive**



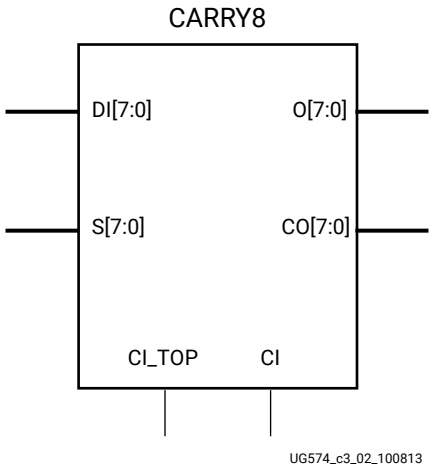
Pin Signals

- **Data In - I0, I1:** The data input provides the data to be selected by the select signal (S).
- **Control In - S:** The select input signal determines the data input signal to be connected to the output O. Logic 0 selects the I0 input, while logic 1 selects the I1 input.
- **Data Out - O:** The data output O provides the data value (one bit) selected by the control inputs.

Carry Chain Primitive

The CARRY8 primitive instantiates the fast carry logic available in each slice. This primitive works with LUTs to build adders and multipliers. the following figure shows the CARRY8 primitive. Synthesis tools generally infer this logic from arithmetic HDL code, automatically connecting this function properly.

Figure 20: **CARRY8 Primitive**



Pin Signals

- **Sum Outputs - O[7:0]:** The sum outputs provide the final result of the addition/subtraction. They connect to the slice combinatorial and registered outputs.
- **Carry Outputs - CO[7:0]:** The carry outputs provide the carry out for each bit. CO[7] on the CARRY8 primitive is equivalent to COUT in the slice (see Figure 8: Fast Carry Logic Path and Associated Elements). A longer carry chain can be created if CO[7] is connected through COUT to the CI input of another CARRY8 primitive, and dedicated routing connects the carry chain up a column of slices. The carry outputs also optionally connect to the slice combinatorial and registered outputs.
- **Carry In - CI:** The carry in input, also called CIN, is used to cascade slices to form longer carry chains.
- **Data Inputs - DI[7:0]:** The data inputs are used as “generate” signals to the carry lookahead logic. The “generate” signals are sourced from LUT outputs.
- **Select Inputs - S[7:0]:** The select inputs are used as “propagate” signals to the carry lookahead logic. The “propagate” signals are sourced from LUT outputs.
- **Split Carry In - CI_TOP:** The CARRY8 has a default CARRY_TYPE of SINGLE_CY8, but it can be split into two independent CARRY4 equivalents with CARRY_TYPE = DUAL_CY4. CI_TOP becomes the external carry input to the upper CARRY4. For the default CARRY8, CI_TOP should be connected to GND.

SLICEM Distributed RAM Primitives

Eight example primitives are shown in the following table. Three primitives are single-port RAM, three primitives are dual-port RAM, and two primitives are octal-port RAM.

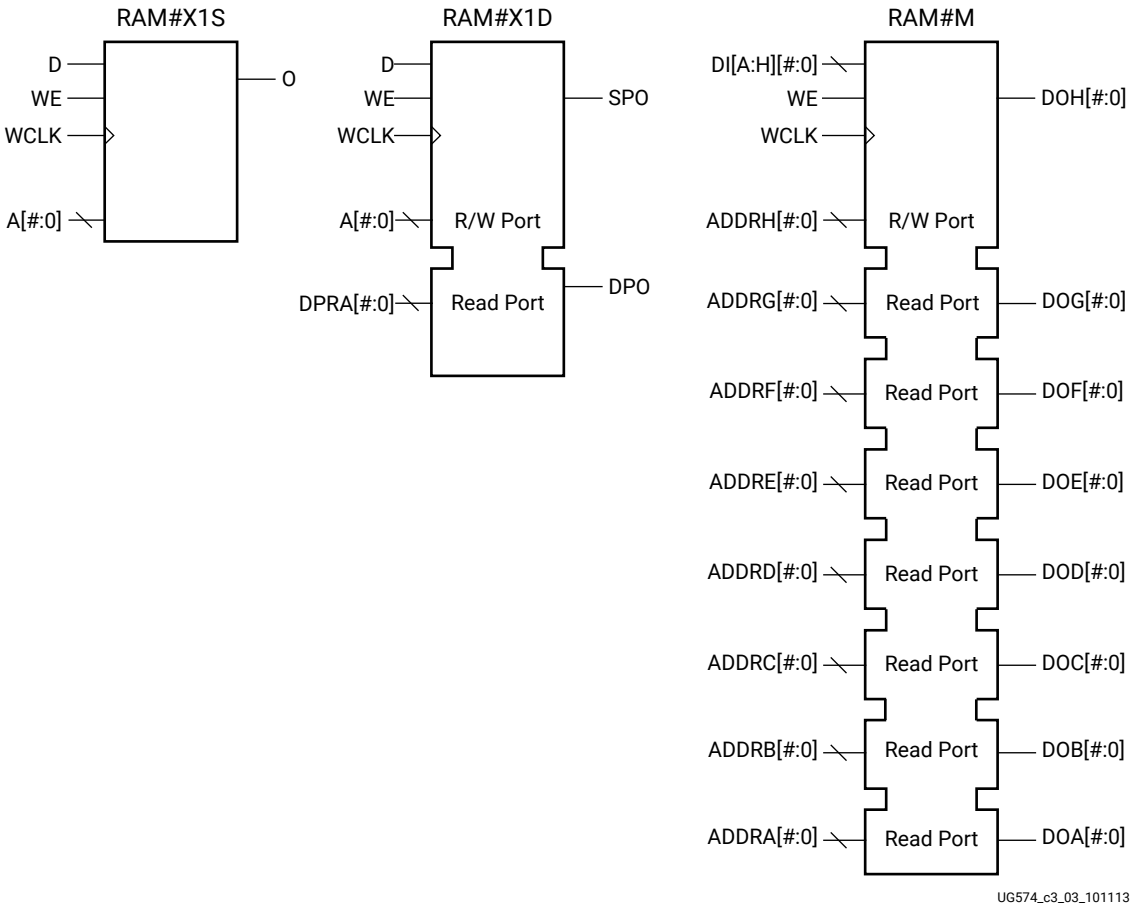
Table 4: **Single-Port, Dual-Port, and Octal-Port Distributed RAM**

Primitive	RAM Size	Type	Address Inputs
RAM32X1S	32-bit	Single-port	A[4:0] (read/write)
RAM32M16	32-bit	Octal 2-bit port	ADDRA[4:0] (read) ADDRB[4:0] (read) ADDRC[4:0] (read) ADDRD[4:0] (read) ADDRE[4:0] (read) ADDRF[4:0] (read) ADDRG[4:0] (read) ADDRH[4:0] (read/write)
RAM64X1S	64-bit	Single-port	A[5:0] (read/write)
RAM64X1D	64-bit	Dual-port	A[5:0] (read/write) DPRA[5:0] (read)
RAM64M8	64-bit	Octal-port	ADDRA[5:0] (read) ADDRB[5:0] (read) ADDRC[5:0] (read) ADDRD[5:0] (read) ADDRE[5:0] (read) ADDRF[5:0] (read) ADDRG[5:0] (read) ADDRH[5:0] (read/write)
RAM128X1S	128-bit	Single-port	A[6:0] (read/write)
RAM256X1D	256-bit	Dual-port	A[7:0], (read/write) DPRA[7:0] (read)
RAM512X1S	512-bit	Single-port	A[8:0] (read/write)

The input and output data are one bit wide (with the exception of the octal-port RAM).

The following figure shows generic single-port, dual-port, and octal-port distributed RAM primitives. The A, ADDR, and DPRA signals are address buses.

Figure 21: **Single-Port, Dual-Port, and Octal-Port Distributed RAM Primitives**



Instantiating several distributed RAM primitives can be used to implement wide memory blocks.

Pin Signals

Each distributed RAM port operates independently of the other while reading the same set of memory cells.

- **Clock – WCLK:** The clock is used for the synchronous write. The data and the address input pins have setup times referenced to the WCLK pin.

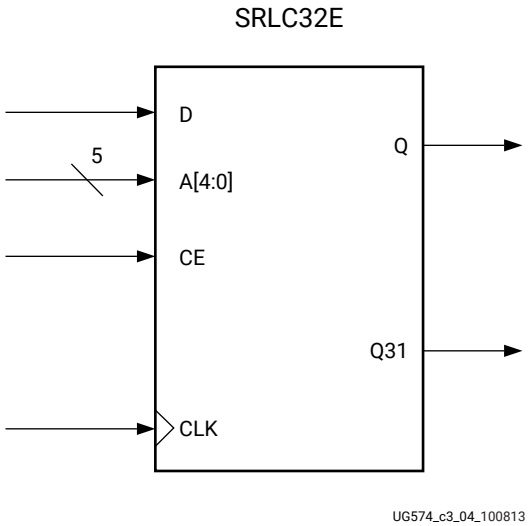
  The clock pin (WCLK) has an inversion option at the slice level. The clock signal can be active at the negative edge of the clock or the positive edge of the clock without requiring other logic resources. The default is at the positive clock edge.

- **Enable – WE:** The enable pin affects the write functionality of the port. An inactive write enable prevents any writing to memory cells. An active write enable causes the clock edge to write the data input signal to the memory location pointed to by the address inputs.
- **Address – A[#:0], DPRA[#:0], and ADDRA[#:0] – ADDRH[#:0]:** The address inputs A[#:0] (for single-port and dual-port), DPRA[#:0] (for dual-port), and ADDRA[#:0] – ADDRH[#:0] (for octal-port) select the memory cells for read or write. The width of the port determines the required address inputs. Some of the address inputs are not buses in VHDL or Verilog instantiations. Table 4: Single-Port, Dual-Port, and Octal-Port Distributed RAM summarizes the function of each address pin.
- **Data In – D, DIH[#:0]:** The data input D (for single-port and dual-port) and DIH[#:0] (for octal-port) provide the new data value to be written into the RAM.
- **Data Out – O, SPO, DPO and DOA[#:0] – DOH[#:0]:** The data out O (single-port or SPO), DPO (dual-port), and DOA[#:0] – DOH[#:0] (octal-port) reflects the contents of the memory cells referenced by the address inputs. Following an active write clock edge, the data out (O, SPO, or DOH[#:0]) reflects the newly written data.

SLICEM SRL Shift Register Primitive

One primitive is available for the 32-bit shift register (SRLC32E), which uses one LUT in a SLICEM. The following figure shows the 32-bit shift register primitive.

Figure 22: **32-bit Shift Register**



Pin Signals

- **Clock – CLK:** Either the rising edge or the falling edge of the clock is used for the synchronous shift operation. The data and clock enable input pins have setup times referenced to the chosen edge of CLK.

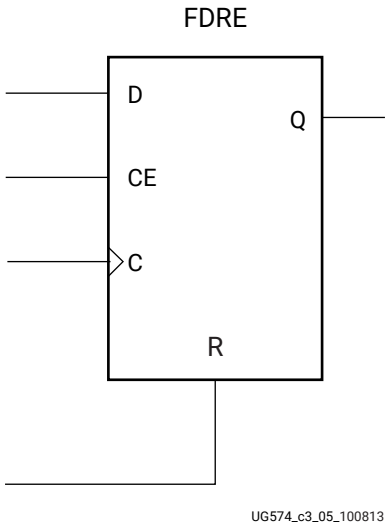
  The clock pin (CLK) has an inversion option at the slice level. The clock signal can be active at the negative or positive edge without requiring other logic resources. The default is positive clock edge.

- **Data In – D:** The data input provides new data (one bit) to be shifted into the shift register.
- **Clock Enable - CE:** The clock enable pin affects shift functionality. An inactive clock enable pin does not shift data into the shift register and does not write new data. Activating the clock enable allows the data in (D) to be written to the first location and all data to be shifted by one location. When available, new data appears on output pins (Q) and the cascadable output pin (Q31).
- **Address – A[4:0]:** The address input selects the bit (range 0 to 31) to be read. The nth bit is available on the output pin (Q). Address inputs have no effect on the cascadable output pin (Q31). Q31 is always the last bit of the shift register (bit 31).
- **Data Out – Q:** The data output Q provides the data value (one bit) selected by the address inputs.
- **Data Out – Q31:** The data output Q31 provides the last bit value of the 32-bit shift register. New data becomes available after each shift-in operation. Q31 is named MC31 on the slice (see Figure 14: 32-bit Shift Register Configuration).

Flip-Flop Primitives

There are several primitives for the CLB slice storage elements, including both flip-flops and latches, with different combinations of control signals available. The FDRE primitive is shown in the following figure for an example. For more information on the flip-flop and latch primitives, see the *UltraScale Architecture Libraries Guide* (UG974).

Figure 23: **FDRE Primitive**



Pin Signals

- **Data In - D:** The data input provides new data (one bit) to be clocked into the flip-flop.
- **Data Out - Q:** Q is the one-bit registered data output from the flip-flop.
- **Clock - C:** Either the rising edge or the falling edge of the clock is used to capture data and toggle the output. The data and clock enable input pins have setup times referenced to the chosen edge of the clock. The clock pin (C) has an inversion option at the slice level. The clock signal can be active at the negative or positive edge of the clock without requiring other logic resources. The default is the positive clock edge.
- **Clock Enable - CE:** When CE is Low, clock transitions are ignored with respect to the D input. The set and reset inputs have priority over the clock enable.
- **Synchronous Reset - R:** When R is High, all other inputs are overridden and the data output (Q) is driven Low on the active clock transition. This signal is available in the FDRE component. The FDRE flip-flop is also cleared by default on power-up.
- **Synchronous Set - S:** When S is High, all other inputs are overridden and the data output (Q) is driven High on the active clock transition. This signal is available in the FDSE component. The FDSE flip-flop is also preset by default on power-up.
- **Asynchronous Clear - CLR:** When CLR is High, all other inputs are overridden and the data output (Q) is driven Low. This signal is available in the FDCE component. The FDCE flip-flop is also cleared by default on power-up.

- **Asynchronous Preset - PRE:** When PRE is High, all other inputs are overridden and the data output (Q) is driven High. This signal is available in the FDPE component. The FDPE flip-flop is also preset by default on power-up.

IMPORTANT! *Using both asynchronous clear and preset on the same flip-flop requires additional resources and timing paths.*

Chapter 4

Applications

Introduction

This chapter provides guidance regarding using the CLB slice resources as parts of larger functions. Trade-offs are described between alternative methods of implementing these functions.

Distributed RAM Applications

Distributed RAM provides a trade-off between using storage elements for very small arrays and block RAM for larger arrays. It is recommended to infer memory where possible to provide the greatest flexibility. Distributed RAM can also be targeted by instantiation or through the use of AMD IP.

In general, distributed RAM should be used for all memories that consist of 64 bits or less, unless there is a shortage of SLICEM or logic resources for the target device. Distributed RAM 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:

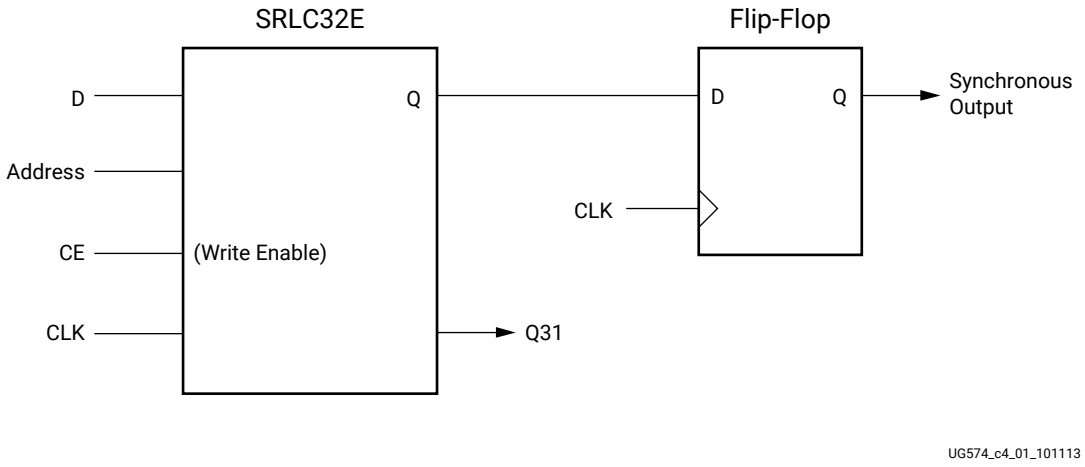
1. The availability of extra block RAMs. If not available, distributed RAM should be used.
2. The latency requirements. If asynchronous read capability is needed, distributed RAMs must be used.
3. The data width. Widths greater than 16 bits should use block RAM, if available.
4. The necessary performance requirements. Registered distributed RAMs generally have shorter clock-to-out timing and fewer placement restrictions than block RAMs.

Shift Register Applications

Synchronous Shift Registers

The shift-register primitive does not use the register available in the same slice. To implement a fully synchronous read and write shift register, output pin Q must be connected to a flip-flop. The shift register and the flip-flop have independent clocks as shown in the following figure.

Figure 24: **Synchronous Shift Register**

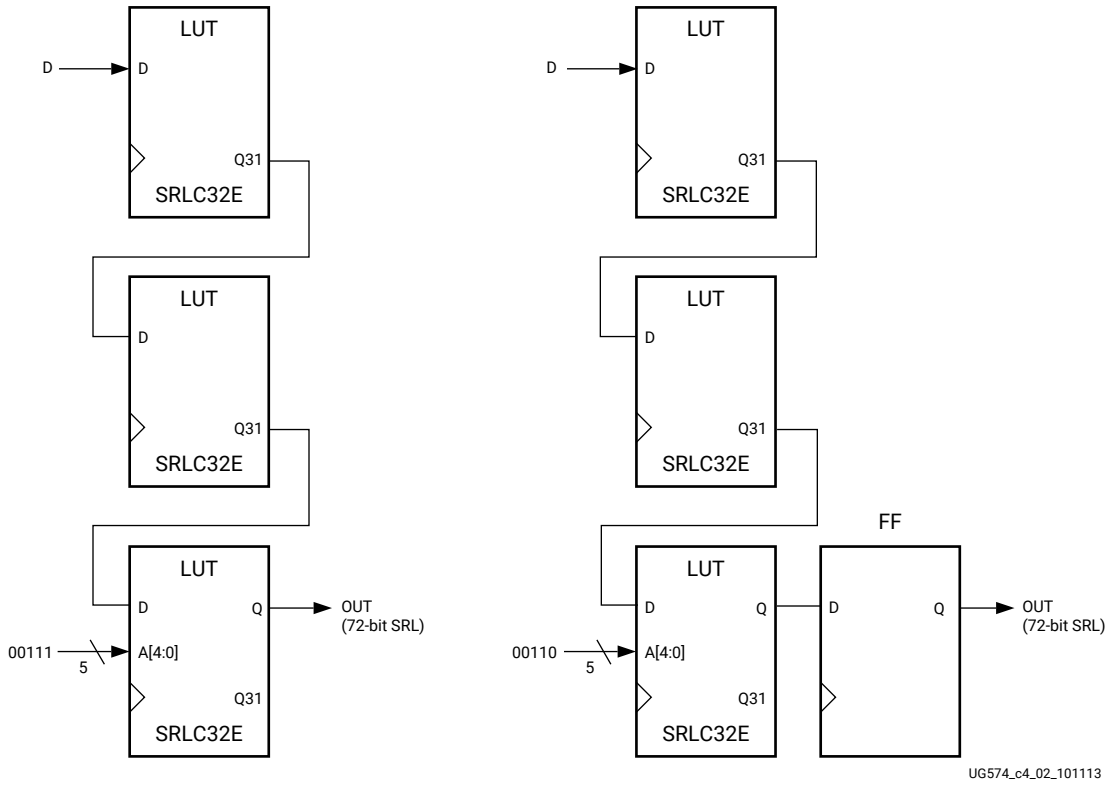


This configuration provides a better timing solution and simplifies the design. Because the flip-flop must be considered to be the last register in the shift-register chain, the static or dynamic address should point to the desired length minus one. If needed, the cascadable output can also be registered in a flip-flop.

Static-Length Shift Registers

The cascadable 32-bit shift register implements any static length mode shift register without the dedicated multiplexers. The following figure illustrates a 72-bit shift register. Only the last SRLC32E primitive needs to have its address inputs tied to `0b00111`. Alternatively, shift register length can be limited to 71 bits (address tied to `0b00110`) and a flip-flop can be used as the last register. (In an SRLC32E primitive, the shift register length is the address input + 1).

Figure 25: **Example Static-Length Shift Register**



Carry Logic Applications

Dedicated carry logic in the CLB improves the performance of arithmetic functions such as adders, counters, and comparators. Designs that include simple counters or adder/subtractors automatically infer the carry logic. More complex functions including multipliers can be implemented using the separate DSP slice.

The DSP slices in every device support many independent functions including multiply, multiply accumulate, multiply add, three-input add, barrel shift, wide-bus multiplexing, magnitude comparator, bitwise logic functions, pattern detection, and wide counter. The architecture also supports cascading multiple DSP slices to form wide math functions, DSP filters, and complex arithmetic. For more details on the DSP slice, see the *UltraScale Architecture DSP Slice User Guide* (UG579).

The choice between using CLB carry logic and the DSP slice depends on the application. A small arithmetic function can be faster and lower power using the CLB carry logic rather than using an entire DSP slice. When the DSP slices are fully utilized, the CLB slices and carry logic provide an efficient alternative.

Using Carry Logic

Carry logic can be inferred or instantiated. Using macros designed for the AMD UltraScale™ architecture can provide the most flexibility and efficiency, especially for more complex functions.

Unimacros include adders, counters, comparators, multipliers, and multiplier/accumulators. The unimacros use the DSP slice.

The AMD IP includes a similar set of functions. When defining these functions, you can specify whether the implementation should be in the CLB carry logic or in the DSP slice.

The carry logic runs vertically up every column of slices. The AMD tools automatically place logic in a column when the carry logic is used. When floorplanning, carry logic based functions should always be specified as a group to avoid unnecessary breaks in the carry chain. Carry logic cannot be cascaded across super logic regions (SLRs) in devices using stacked silicon interconnect (SSI) technology.

Chapter 5

Advanced Topics

Asynchronous Clock Domain Crossing

Signals transferring from one clock domain to another can be categorized into either the preferred synchronous crossing, in which there is a known and predictable phase relationship between domains, or an asynchronous crossing. The asynchronous clock domain crossing paths must be reviewed carefully to ensure that they use proper synchronization circuitry that does not rely on timing correctness and that minimizes the chance for metastability to occur. Special steps must be taken to mitigate improper bus capture, metastability, and other occurrences that can affect the data integrity in such paths.

In general, there are two popular methods to allow data to cross asynchronous clock domains safely. If only a single bit is needed or if methods such as grey-coding are used to transfer more than one bit of related data, register synchronizers can be inserted to improve the Mean Time Between Failures (MTBF) of the circuit. For multiple bits of data (that is, a bus), the generally recommended practice is to use an independent clock (asynchronous) FIFO to safely transfer data from one domain to another. Such a FIFO can be built from the dedicated logic in the block RAM. The dual-clock FIFO design avoids ambiguity, glitches, or metastability problems, and provides a convenient way to pass data between differing clock domains. See the Independent-Clock/Dual-Clock FIFO section in the *UltraScale Architecture Memory Resources User Guide* (UG573).

Register Synchronizers

The flip-flops in the CLB slices of the AMD UltraScale™ architecture are designed to maximize the MTBF of asynchronous signals. Using standard slice flip-flops for register synchronizers makes efficient use of this flexible and abundant resource. Use the ASYNC_REG attribute in your HDL code to identify all synchronizing registers. By doing so, the AMD Vivado™ Design Suite can better understand and use special algorithms to improve synthesis, optimization, simulation, placement, and routing to improve MTBF, by reducing metastability occurrences. If ASYNC_REG is applied, the placer will ensure the flip-flops on a synchronization chain are placed closely to maximize MTBF. Registers with ASYNC_REG that are directly connected will be grouped and placed together into a single slice, assuming they have a compatible control set and the number of registers does not exceed the available resources of the slice.

HARD_SYNC Dedicated Synchronizers

For most designs, creating synchronizers from CLB slice flip-flops will be sufficient. For designs with very high MTBF or when synchronizing very high speed signals, the HARD_SYNC block is available. The HARD_SYNC blocks contain registers with very fast metastable recovery that are ideally suited to synchronize high-speed external signals, for asynchronous handshaking, and synchronizing a limited amount of signals that cross clock domains within the device. The HARD_SYNC synchronizers can be used in place of slice registers for applications with very long MTBF requirements, very high-speed domain crossing, or a combination of those with consistent low temperature operation.

The HARD_SYNC synchronizer block can be configured with two or three stages of latency. Similar to the CLB slice registers, the HARD_SYNC registers can be initialized during configuration and have an optional inverter on the clock input.

The HARD_SYNC primitive (the following figure) must be instantiated to be used in a design. Two or three registers can be in a single HARD_SYNC synchronizer chain (Figure 27: HARD_SYNC Synchronizer Logic). Three stages will exhibit better MTBF characteristics at the expense of an additional clock cycle of latency. Subsequent registers should use regular slice registers.

Figure 26: **HARD_SYNC Primitive**

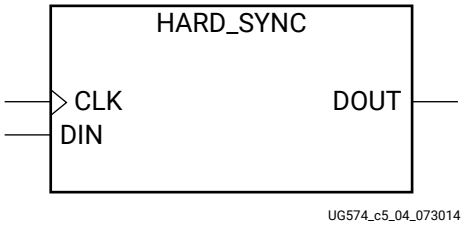
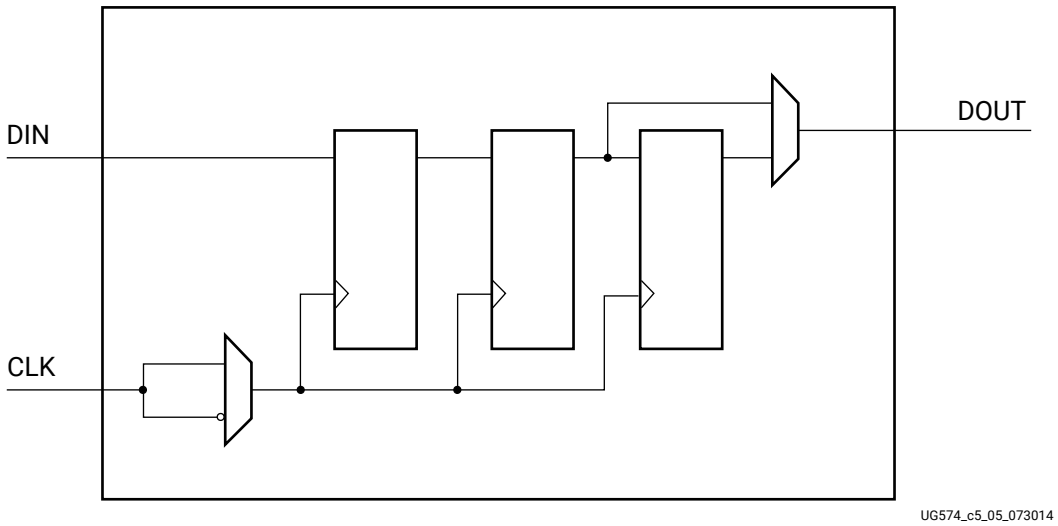


Figure 27: **HARD_SYNC Synchronizer Logic**



Pin Signals

- **Data In - DIN:** The data input provides the asynchronous data (one bit) to be clocked into the first register.
- **Clock - CLK:** Either the rising edge or the falling edge of the clock is used to capture input data and toggle the synchronizer output. The clock pin has an inversion option, with the default being the positive clock edge. All registers in the synchronizer use the same clock and same clock polarity.
- **Data Output - DOUT:** DOUT is the one-bit synchronized data output from the second or third register in the synchronizer.

The latency, initial value, and clock polarity are controlled by attributes (see the following table).

Table 5: **HARD_SYNC Attributes**

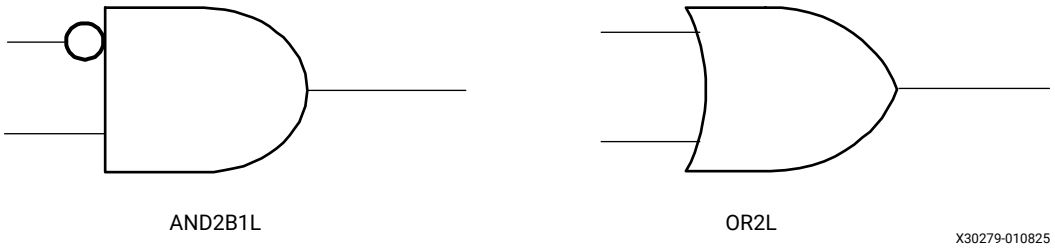
Attribute Name	Values	Default	Type	Description
LATENCY	2, 3	2	Decimal	Number of registers in the synchronizer
INIT	0, 1	0	1-bit Binary	Initialization value for all registers in the synchronizer
IS_CLK_INVERTED	0, 1	0	1-bit Binary	Optional inversion of the clock to active-Low

Four synchronizers are located in the horizontal clock spine in the middle of each of the block RAM columns in a clock region (see *UltraScale Architecture Clocking Resources User Guide* (UG572) for information on the clock regions).

Using the Latch Function as Logic

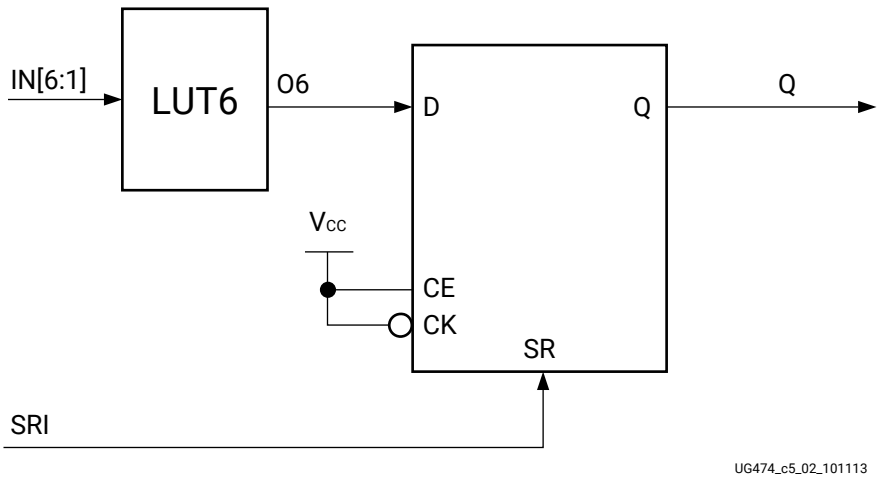
Because the latch function is level-sensitive, it can be used as the equivalent of a logic gate. The primitives to specify this function are AND2B1L (a 2-input AND gate with one input inverted) and OR2L (a 2-input OR gate), as shown in the following figure.

Figure 28: **AND2B1L and OR2L Components**



As shown in the following figure, the data and SR inputs and Q output of the latch are used when the AND2B1L and OR2L primitives are instantiated, and the CK gate and CE gate enables are held active-High. The AND2B1L combines the latch data input (the inverted input on the gate, DI) with the asynchronous clear input (SRI). The OR2L combines the latch data input with an asynchronous preset. Generally, the latch data input comes from the output of a LUT within the same slice, extending the logic capability to another external input. Because there is only one SR input per top or bottom half of the slice, using more than one AND2B1L or OR2L per half slice requires a shared common external input.

Figure 29: **Implementation of OR2L (Q = D or SRI)**



The AND2B1L and OR2L two-input gates save LUT resources and are initialized to a known state on power-up. Using these primitives can reduce logic levels and increase logic density of the device by trading register/latch resources for logic. However, due to the static inputs required on the clock and clock enable inputs, specifying one or more AND2B1L or OR2L primitives can cause register packing and density issues in a slice disallowing the use of the remaining registers and latches.

Interconnect Resources

Interconnect is the programmable network of signal pathways between the inputs and outputs of functional elements within the device, such as IOBs, CLB slices, DSP slices, and block RAM. Interconnect, also called routing, is segmented for optimal connectivity. The AMD placement and routing tools exploit the rich interconnect array to deliver optimal system performance and the fastest compile times.

The CLB slices are arranged in a regular array. Each connects to a switch matrix for access to the general-routing resources, which run vertically and horizontally between the rows and columns. A similar switch matrix connects other resources, such as the DSP slices and block RAM resources.

Most of the interconnect features are transparent to designers. Knowledge of the interconnect details can be used to guide design techniques but is not necessary for efficient design. Only selected types of interconnect are under user control. These include the clock routing resources, which are selected by using clock buffers and discussed in more detail in the *UltraScale Architecture Clocking Resources User Guide* (UG572).

Interconnect Optimization

Interconnect delays vary according to the specific implementation and loading in a design. The type of interconnect, distance required to travel in the device, and number of switch matrices to traverse factor into the total delay. Most timing issues are addressed by making sure that the required timing is in the constraints file, or by examining the block delays and determining the impact of using fewer levels or faster paths. If interconnect delays seem too long, increase implementation effort strategies or iterations to improve performance.

Nets with critical timing or that are heavily loaded can often be improved by replicating the source of the net. The dual 5-input LUT configuration of the slice simplifies the replication of logic in the same slice, which minimizes any additional loads on the inputs to the source function. Replicating logic in multiple slices gives the tools more flexibility to place the sources independently.

Floorplanning is the process of specifying user-placement constraints. Floorplanning can be done either before or after implementation, but automatic place and route is always recommended first before specifying user floorplanning. The AMD tool provides a graphical view of placement. It helps you to choose between RTL coding and synthesis and implementation, with extensive design exploration and analysis features. See the *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949) for best general design practices.

3D IC Devices using Stacked Silicon Interconnect (SSI) Technology

Some of the UltraScale architecture-based devices use stacked silicon interconnect (SSI) technology. These devices provide special routing resources between super logic regions (SLRs). These special routing resources are known as super long lines (SLLs). Combining multiple homogeneous SLR die on a single interposer effectively increases the height of the columns and increases the overall capacity of the device. The tools can be allowed to take best advantage of this configuration, or users can apply floorplanning to control placement within and between SLRs.

***Note*:** Carry logic cascading is limited to within an SLR and does not continue through the SLL connections. The same is true of other types of cascading including block RAM, DSP, and DCI. Boundaries are visible in floorplanning tools and reported in timing reports.

Optimization for devices using SSI technology can start with the same techniques as for any standard monolithic device, including using proper design techniques, timing constraints or options to automatically find the best implementation. Floorplanning for these devices can use the same graphical tools.

For more information on devices using SSI technology, see the following:

- All Programmable 3D ICs
- *Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency* (WP380)

Second Generation 3D IC Interconnect

The UltraScale architecture uses the second generation 3D IC interconnect to connect together multiple SLRs in a single device. A unique feature of this interconnect is the flip-flops at the boundaries between SLRs, providing a predictable, high-speed interface from one SLR to the next in an SSI device. These dedicated flip-flops exist on both SLRs so that the transmitting (TX) and/or receiving (RX) SLR can provide the storage element for a unidirectional connection across the SLL. Combinatorial connections can also be made.

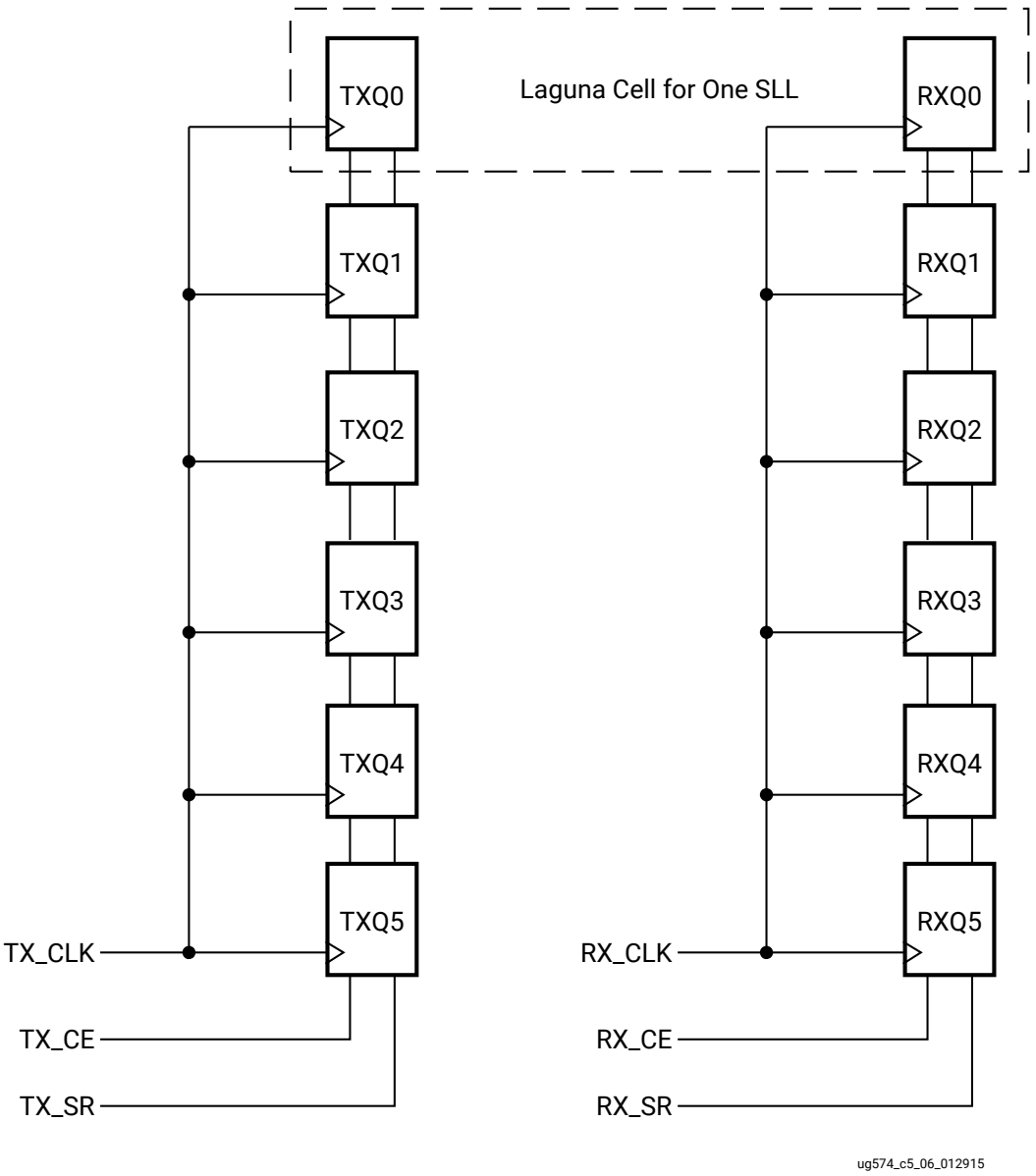
Flip-flops

The flip-flops in the 3D IC interconnect can be used by the tools when a design spans multiple SLRs and has a registered signal at the boundary. The interconnect flip-flops are similar to the CLB slice flip-flops, allowing placement flexibility. The interconnect flip-flops do not allow a latch mode, but have clock enable and initialization functions similar to slice flip-flops. Interconnect flip-flops can be used by the tools whenever a standard flip-flop is instantiated or inferred within a design, or when a flip-flop is replicated by the implementation tools. An interconnect flip-flop cannot be specifically instantiated in a design, but floorplanning can be used to place selected flip-flops in the 3D IC interconnect resources, which are referred to as Laguna cells.

The flip-flops are paired in cells consisting of a TX flip-flop and an RX flip-flop, one of which can be used for the associated connection. The Laguna cell on either or both sides of the SLR boundary can be used to make the SLL connection. Therefore, there are four flip-flops for every connection, but a maximum of two could be used in a given application. In some situations, TX and RX flip-flops cannot be used at the same time for the same path due to potential hold time violations, however the tools will only use both registers when a hold time will not occur.

A Laguna site contains six Laguna cells, and the control signals operate at the site level (see the following figure). A Laguna tile contains four Laguna sites, and is the same height as a CLB slice. The Laguna tiles span the entire height of the clock region, 60 tiles tall. There are two Laguna columns per clock region, but only the top and bottom clock regions in an SLR contain Laguna columns if it is adjacent to another SLR.

Figure 30: **Interconnect Flip-Flops in a Laguna Site**



The total number of available interconnect flip-flops in a device depends on the number of clock regions in a row and the number of SLRs. The KU115, for example, has six clock regions per row and consists of two SLRs, so there are 69,120 interconnect flip-flops available to support the 17,280 SLL connections across the SLR boundary. The number of SLLs for a given SLR can be reported using the Tcl command `get_property NUM_SLLS [get_slrs SLR0]`. Refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) for more information on Tcl commands.

The interconnect flip-flops have initialization and clock enable controls similar to the CLB slice flip-flops. Each group of six TX flip-flops within a site has its own clock and optional clock enable and set/reset control inputs, as do each set of six RX flip-flops in a site. The clock is sourced from the leaf clocks and has programmable polarity at the site. The set/reset also has programmable polarity at the site level.

Similar to the slice flip-flops, asserting the SR signal can force a logic High or Low at the flip-flop output, with the SR function selectable as synchronous or asynchronous. Both options apply to all of the flip-flops on a given SR signal, the six TX flip-flops or the six RX flip-flops in a site.

Similar to the slice flip-flops, the initial state after configuration is defined with INIT set to 0 or 1. By default, when the SR forces a logic High then the INIT state is also High, and SR Low defaults to INIT Low. The INIT definition is controlled at the cell level for a TX/RX pair.

Appendix A

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD 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 DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

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

References

These documents provide supplemental material useful with this guide:

1. *UltraScale Architecture DSP Slice User Guide* (UG579)
2. *UltraScale Architecture Memory Resources User Guide* (UG573)
3. *UltraScale Architecture and Product Data Sheet: Overview* (DS890)
4. *UltraScale Architecture Migration: Methodology Guide* (UG1026)
5. *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949)
6. *UltraScale Architecture Libraries Guide* (UG974)
7. *UltraScale Architecture Clocking Resources User Guide* (UG572)
8. All Programmable 3D ICs page
9. *Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency* (WP380)
10. *Vivado Design Suite Tcl Command Reference Guide* (UG835)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
01/22/2025 Version 1.6	
Address Collision	Added topic.
Pin Signals	Removed flip-flop instruction.
02/28/2017 Version 1.5	
Distributed RAM (SLICEM Only)	Changed “Dual-port 32 x (1 to 4)-bit RAM” to “Dual-port 32 x (1 to 8)-bit RAM” and “Quad-port 32 x (1 to 8)-bit RAM” to “Quad-port 32 x (1 to 4)-bit RAM."
Flip-flops	Clarified second paragraph following Figure 30.
Second Generation 3D IC Interconnect	Deleted the word “interposer” preceding “boundaries” and “connections.”

Section	Revision Summary
11/24/2015 Version 1.4	
General updates	Added AMD UltraScale+™ device information.
10/29/2015 Version 1.3	
Differences from Previous Generations	Deleted last bullet.
Figure 5	Updated figure.
Multiplexers	Updated last paragraph.
32:1 Multiplexer	Updated first paragraph.
Distributed RAM (SLICEM Only)	Updated second paragraph.
02/20/2015 Version 1.2	
Chapter 1: Overview	Changed title of Chapter 1, from “Introduction” to “Overview.”
CLB Slice Resources	Added last two sentences.
CLB Resources	Clarified first paragraph.
Distributed RAM (SLICEM Only)	Changed “Quad-port 32 x (1 to 4)-bit RAM” to “Quad-port 32 x (1 to 8)-bit RAM" and "Dual-port 128 x 1-bit RAM" to "Dual-port 128 x 2-bit RAM."
Chapter 3: Design Entry	Minor editorial changes.
Second Generation 3D IC Interconnect	Changed title of “Laguna Flip-Flops” to “Second Generation 3D IC Interconnect.”
Chapter 5: Advanced Topics	Clarified use of the term “Laguna” with respect to the interconnect flip-flops.
General updates	Deleted Table: “Total Synchronizers Available per Device”.
3D IC Devices using Stacked Silicon Interconnect (SSI) Technology	Minor editorial changes and added Figure 30.
09/08/2014 Version 1.1	
Differences from Previous Generations	Deleted "Differences in Devices Based on SSI Technology" sub-section and added last paragraph.
General updates	Combined Table 1-1 and Table 1-2.
Initialization	Updated and clarified.
Distributed RAM (SLICEM Only)	Clarified sixth paragraph.
Design Checklist	Added reference to *UltraFast Design Methodology Guide for FPGAs and SoCs* (UG949).
Chapter 5: Advanced Topics	Added Asynchronous Clock Domain Crossing.
3D IC Devices using Stacked Silicon Interconnect (SSI) Technology	• Updated references. • Added Second Generation 3D IC Interconnect.
Appendix A: Additional Resources and Legal Notices	Updated references.
12/10/2013 Version 1.0	
Initial AMD release.	N/A

Please Read: Important Legal Notices

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes. THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright