Versal ACAP Configurable Logic Block

Architecture Manual

AM005 (v1.0) July 16, 2020

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
07/16/2020 Version 1.0	
Initial release.	N/A

Table of Contents

XILINX

Chapter 1

Overview

Introduction to Versal ACAP

Versal™ adaptive compute acceleration platforms (ACAPs) combine Scalar Engines, Adaptable Engines, and Intelligent Engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. Most importantly, Versal ACAP hardware and software are targeted for programming and optimization by data scientists and software and hardware developers. Versal ACAPs are enabled by a host of tools, software, libraries, IP, middleware, and frameworks to enable all industry-standard design flows.

Built on the TSMC 7 nm FinFET process technology, the Versal portfolio is the first platform to combine software programmability and domain-specific hardware acceleration with the adaptability necessary to meet today's rapid pace of innovation. The portfolio includes six series of devices uniquely architected to deliver scalability and AI inference capabilities for a host of applications across different markets—from cloud—to networking—to wireless communications—to edge computing and endpoints.

The Versal architecture combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC) to enable seamless memory-mapped access to the full height and width of the device. Intelligent Engines are SIMD VLIW AI Engines for adaptive inference and advanced signal processing compute, and DSP Engines for fixed point, floating point, and complex MAC operations. Adaptable Engines are a combination of programmable logic blocks and memory, architected for high-compute density. Scalar Engines, including Arm® Cortex™-A72 and Cortex-R5F processors, allow for intensive compute tasks.

The Versal AI Core series delivers breakthrough AI inference acceleration with AI Engines that deliver over 100x greater compute performance than current server-class of CPUs. This series is designed for a breadth of applications, including cloud for dynamic workloads and network for massive bandwidth, all while delivering advanced safety and security features. AI and data scientists, as well as software and hardware developers, can all take advantage of the high-compute density to accelerate the performance of any application.

The Versal Prime series is the foundation and the mid-range of the Versal platform, serving the broadest range of uses across multiple markets. These applications include 100G to 200G networking equipment, network and storage acceleration in the Data Center, communications test equipment, broadcast, and aerospace & defense. The series integrates mainstream 58G transceivers and optimized I/O and DDR connectivity, achieving low-latency acceleration and performance across diverse workloads.

The Versal Premium series provides breakthrough heterogeneous integration, very high-performance compute, connectivity, and security in an adaptable platform with a minimized power and area footprint. The series is designed to exceed the demands of high-bandwidth, compute-intensive applications in wired communications, data center, test & measurement, and other applications. Versal Premium series ACAPs include 112G PAM4 transceivers and integrated blocks for 600G Ethernet, 600G Interlaken, PCI Express® Gen5, and high-speed cryptography.

The Versal architecture documentation suite is available at: https://www.xilinx.com/versal.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - CLB Resources
  - Look-Up Table
  - Storage Elements
  - Carry Logic
  - Primitives

CLB Features

The configurable logic block (CLB) provides the most basic, flexible logic functionality in Versal™ adaptable computing acceleration platforms (ACAPs). It can map any arbitrary function into programmable resources. Features include:

- Implementation of any arbitrary programmable logic function into functional units (LUTs)
- Flip-flops and latches for state retention and pipelining

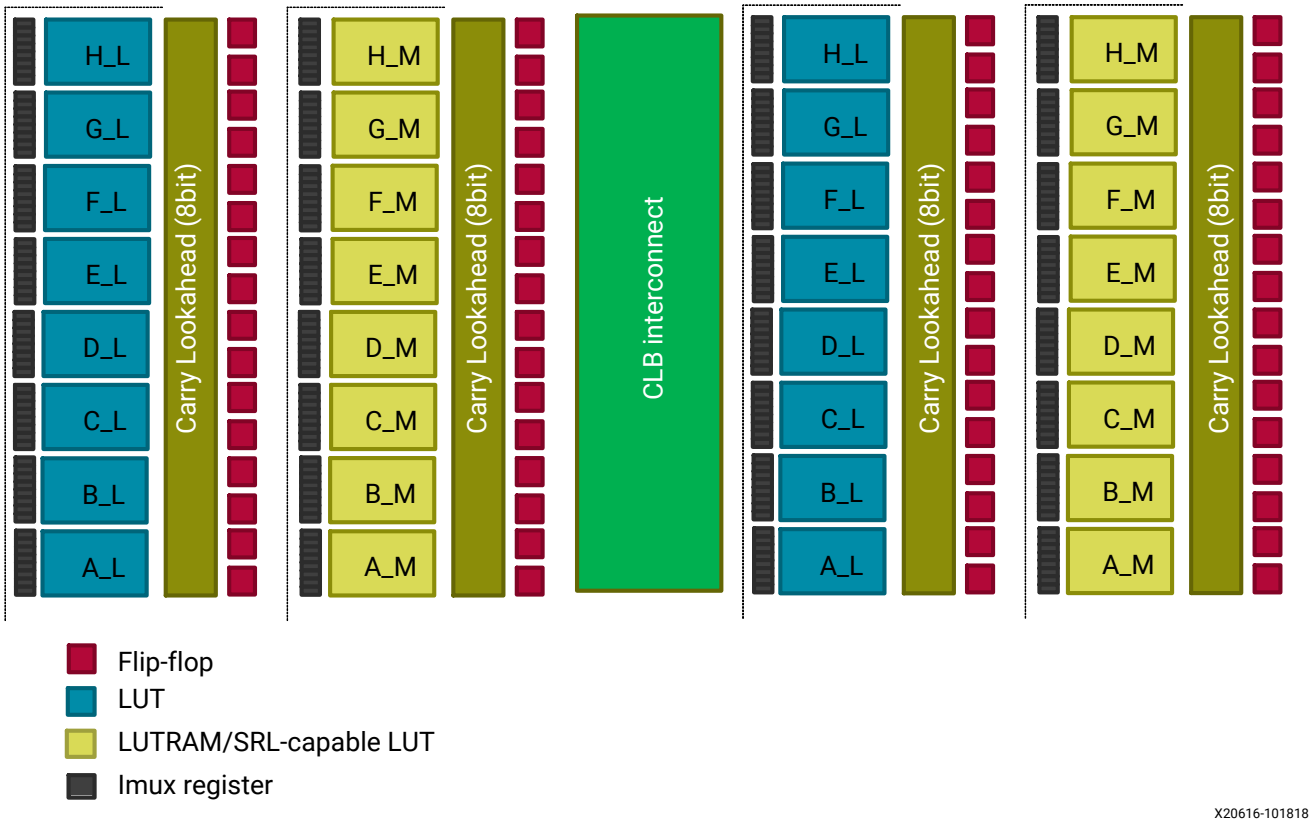
- Acceleration of wide arithmetic/logic functions
- Shift registers
- Small (64-bit) distributed RAM

CLB Architecture

The CLB is the main resource in each Versal™ device and implements programmable combinational logic, sequential logic, and logic paths. These features enable high functionality and routability.

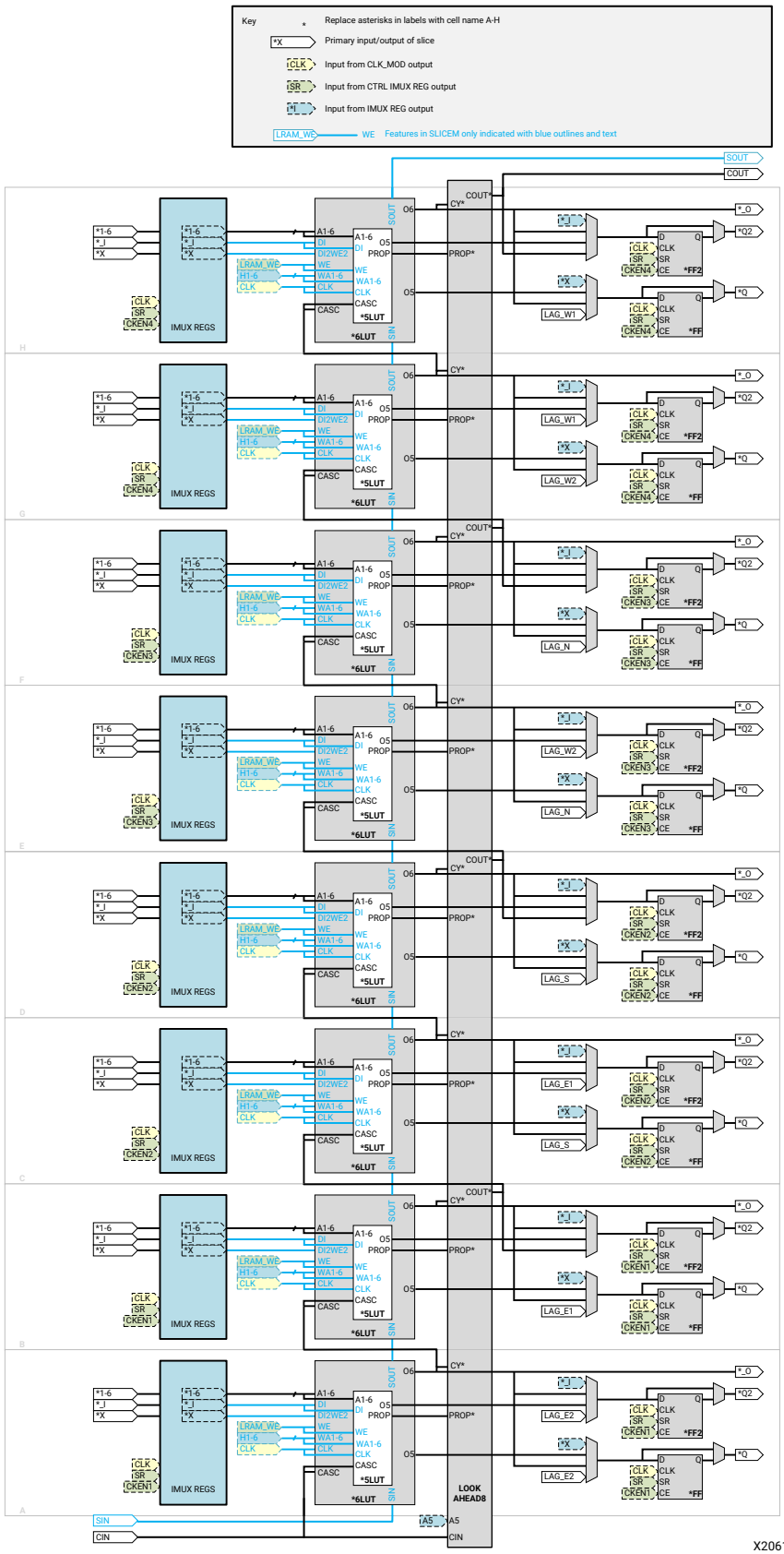
The following figure shows a high-level block diagram of the CLB. There are two CLB types, one with super long line (SLL) connections, and one without. Each CLB contains equal numbers of LUTRAM and SRL-capable LUTs. Only one LUT type can be used in a SLICEM.

Figure 1: **CLB Block Diagram**



The following figure shows a Versal device SLICEL/SLICEM. Note the Imux registers, the carry lookahead logic which now contain fast lookahead multiplexers, and input and output multiplexers before and after the flip-flops. The multiplexers after the flip-flops are new to Versal devices. Some of the inputs to the input multiplexer are from the SLL connections.

Figure 2: **SLICEL/SLICEM Block Diagram**



Differences from Previous Generations

Differences between Versal™ adaptable computing acceleration platform (ACAP) configurable logic blocks (CLBs) and previous generations of UltraScale™ device CLBs are as follows:

- The CLB tile has been completely redesigned. The CLB has four times more logic capacity (32 LUTs/64 slice flip-flops as opposed to 8 LUTs/16 slice flip-flops in UltraScale devices). This results in more local routing for better performance and less general routing congestion.
- Dedicated LUT-LUT cascade paths now exist inside the CLB to reduce delays on multi-logic level paths as well as reduce external routing demands. In addition, the LUT-LUT cascade paths are leveraged to reduce cost and enable a more flexible carry logic structure.
- Super long line (SLL) connections are now part of the CLB (as opposed to being a dedicated column in previous architectures). There are no dedicated registers as the SLL connections rely on registering signals in the CLB.
- Wide function multiplexers (MUXF7, MUXF8, and MUXF9) are no longer implemented. Other LUTs are now used to implement wide multiplexing.
- There are now three outputs per LUT/FF pair instead of four. This enhances routability by increasing fanout per output.
- Dual LUT mode now supports 2 functions of up to 6 independent inputs.
- New cascade multiplexers enable new carry chains to start at bits 0 and 4.
- There is only one CLB type. One half of the LUTs in a CLB are capable of supporting LUTRAM and SRL configurations.
- LUTRAMs are simplified, having dedicated hardware to support 32 and 64 bit depths. Deeper LUTRAMs can be implemented using additional logic.
- Control sets for CLK and SR are at a coarser granularity, but CE stays the same.
- Output multiplexing in CLBs is new to Versal architecture. Each flip-flop is bypassable and can select one of several inputs. O6 comes straight out to interconnect and in carry mode also acts as carry_out. Both flip-flops receive O6 but each flip-flop only receives one of the O5 signals (O5_1 and O5_2).
- Additional registers (Imux registers) are embedded into the CLB and also exist in the local interconnect block for all hard blocks connected to programmable logic routing. They allow additional pipelining by breaking critical paths into smaller pieces to increase F_{MAX}. They are also used to fix hold time violations by gating data for a half cycle. This frees up routing resources previously used for hold time fixing.

XILINX

Chapter 2

CLB Resources

Overview

This chapter provides a detailed view of the Versal™ adaptable computing acceleration platform (ACAP) configurable logic block (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.
- Carry Logic: Dedicated gates and cascading to implement efficient arithmetic functions.
- Primitives: Overview of the most commonly used CLB primitives.

CLB Resources

Every configurable logic block (CLB) contains four slices totaling 32 6-input look-up tables (LUTs) and 64 slice storage elements. The LUTs in the CLB are grouped in groups of eight. Unlike previous generations, there is only one CLB type. Groups of four SLICEM LUTs are supported. Four logic LUTs and either four LUTRAMs or four SRLs can be placed in a SLICEM. Each CLB contains exactly 50% LUTRAM/SRL capable LUTs. There is a carry block next to each group of eight LUTs that can be used together with the LUTs to implement various arithmetic functions. Multiplexers and slice flip-flops are located next to the carry block. This allows the outputs of the LUTs and the carry logic to interface with the interconnect block directly, or through programmable flip-flops.

The CLBs are arranged in columns. The local interconnect and super long line (SLL) connections, located in the middle of the CLB, provide additional CLB interfaces. An interconnect block is located on the left and right side of each CLB column. The left and right halves of the CLB are identical. Therefore, many of the descriptions in this architecture manual only focus on one half of the CLB.

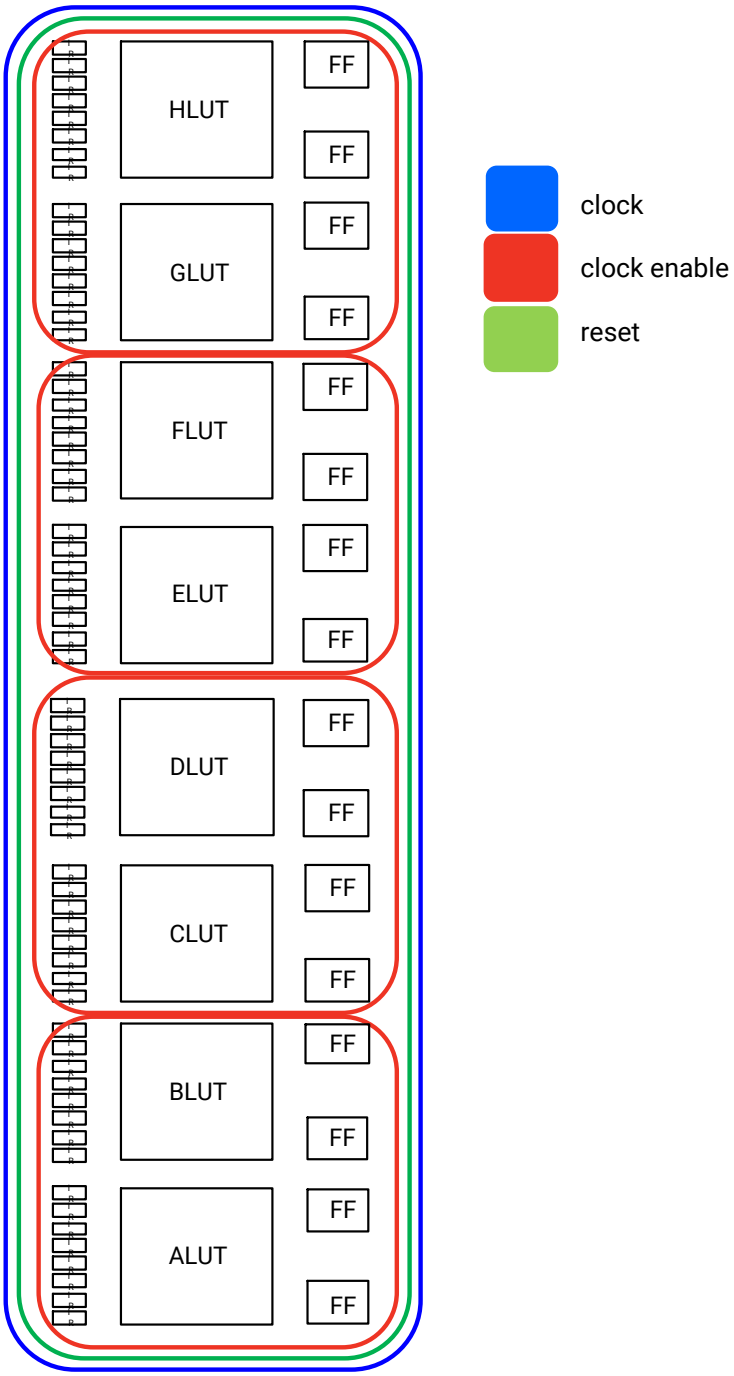
Imux Register

Inverse multiplexer (Imux) registers are embedded into the CLB. Imux registers exist on the following CLB inputs: 192 Imuxs, 64 bypasses, 16 clock enable (CE) pins, 4 set/reset (SR) pins, and 2 write enable (WE) pins. The Imux registers are located near the interconnect/CLB boundary and can be bypassed. CE, SR, and WE Imux control registers are located in the interior of the CLB.

The Imux registers support a subset of CLB flip-flop features. Each Imux register has a clock enable and synchronous or asynchronous reset capabilities. They have no readback/writeback, and no sync /async set capability. Initialization is programmable but the options are either init=0 or init=data input (no init=1). The init=data input is necessary when using Imux registers as hold-time fixing elements.

It is important to note that the Imux registers used for CE, SR, and WE CLB inputs have no clock enable or reset capabilities. From a control set standpoint, Imux registers are clumped with the CLB flip-flops that their associated LUTs drive. In other words, if an Imux drives an input on LUTA, then the Imux register is on the same control set as CLB flip-flop A. Similarly, if a bypass pin drives an Imux register that drives a downstream CLB flip-flop, the Imux register is on the same control set as the downstream flip-flop. This creates control set clusters of 16 Imux registers. The Imux control sets are shown in the following figure.

Figure 3: **Imux Control Sets**

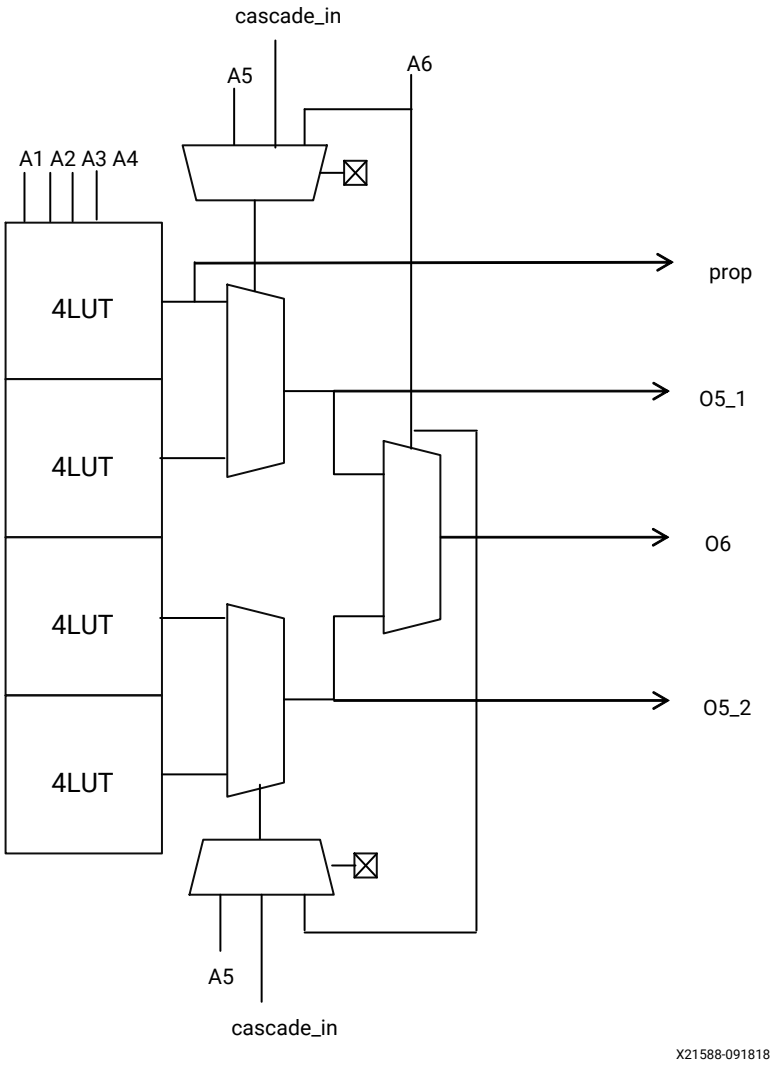


The clock enable can independently be ignored for each cluster of 16 IMUX registers shown in the red boxes in the above figure. The reset can independently be ignored for each slice which is a cluster of 64 IMUX registers. Clock gating exists at the same cluster of 16 granularity, as well as for each latch and slice.

Look-Up Table

All look-up tables (LUTs) in the configurable logic block (CLB) are 6LUTs. The 6LUT is enhanced with additional multiplexing to enable even more functionality. All features of a LUT are shown in the following figure.

Figure 4: **LUT Features**



The two multiplexers near the top and bottom of the diagram are new to Versal™ architecture. They are static memory cell controlled muxes. These multiplexers are used for the following purposes:

1. Cascadable LUT -> LUT connections (O6 -> A5)
2. Enables dual LUT functions of up to six inputs (five in prior architectures)

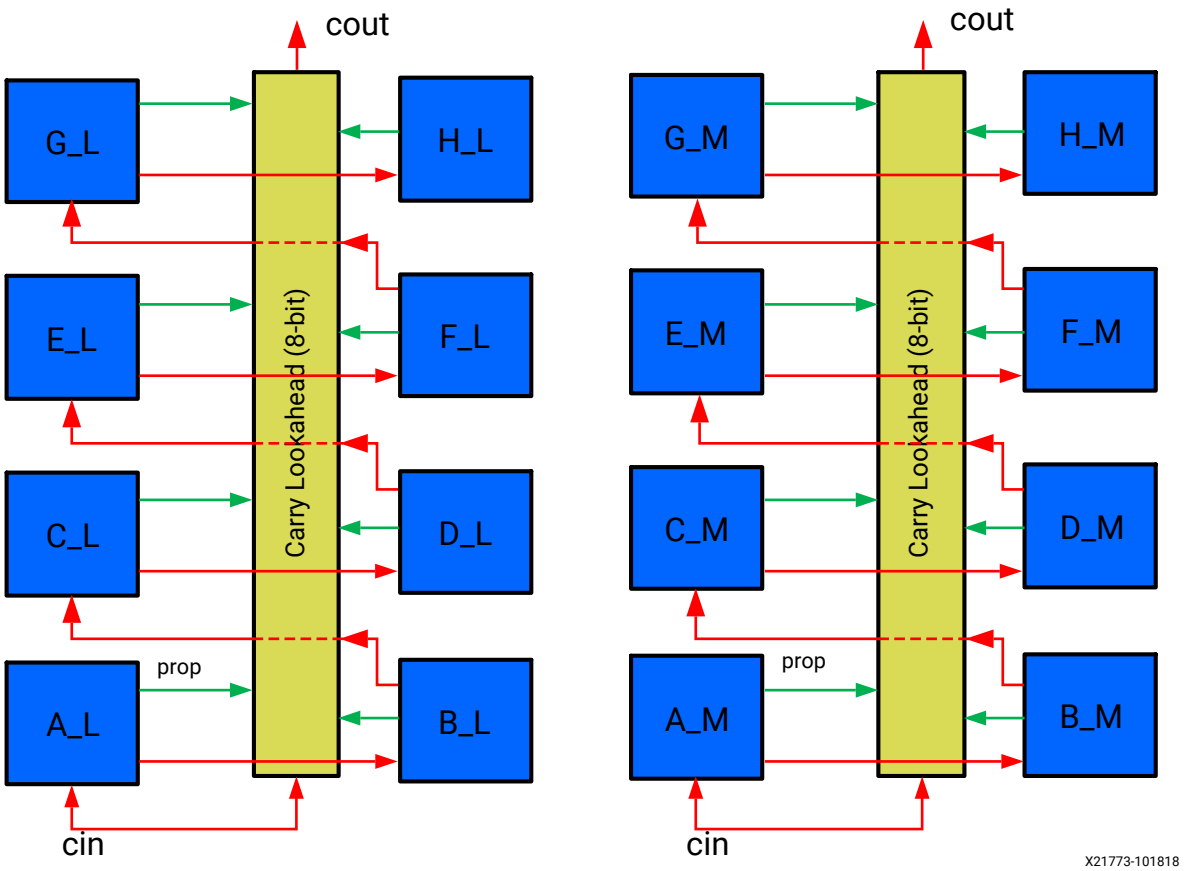
3. Used for carry logic paths

These multiplexers select the A5 input by default. They can also be programmed to select no input and in that case their outputs are driven High.

There are four LUT outputs. The prop output is only used for carry logic and is not visible outside the CLB. For standard 6LUT mode, O6 is used as the output. For dual 5LUT mode, O5_1 and O5_2 are both used to bring out the two function outputs to the logic.

Cascade multiplexer paths exist between logically adjacent LUTs in one direction (A->B->C->D->E->F->G->H but not the reverse). The O6 output of one LUT connects through the cascade multiplexer to the A5 pin of the following LUT. The intention is to create a short, fast path between two LUT stages in order to a) reduce delay on critical paths and b) reduce external routing consumption. Optimal use of cascades might result in swapping around LUTs inside a CLB to place critical LUTs next to each other when the penalty to surrounding logic is low. Note that cascade connections are self-contained within a CLB slice (8 LUTs), and do not cross slice boundaries. Also note that the odd connections travel through the carry chain to get to the subsequent LUT. When used as a cascade (and not for carry logic), the carry block is configured such that the prop output is forced to make the carry logic act as a route through for those cascade paths. The following figure is a picture of the cascade connections for one half of the CLB (red arrows). The solid red lines represent actual wires. The dotted red lines represent logical connections that only exist in cascade mode.

Figure 5: **Cascade Pattern**



LUTRAM

Versal™ architecture LUTRAMs are simplified compared to previous architectures. Notable changes are:

- LUTRAM cells are available for depths of 64 and 32. Anything deeper will be decomposed to a set of those cells and a multiplexer tree.
  - No high-order address decode circuitry, and there are six write address inputs to each LUT.
  - No MUXF7, MUXF8, MUXF9 bels. Deeper LUTRAMs can be implemented using additional logic.
  - Because hardware support for deeper LUTRAM configurations is reduced placement restrictions exist for clusters of LUTs/SRLs/LUTRAMs to achieve optimal performance.
- Support for dual-edge clocking.
- Dedicated LRAM_WE site input for LUTRAM write-enable.
- LUTRAMs or SRLs can be combined with logic LUTs in a SLICEM.

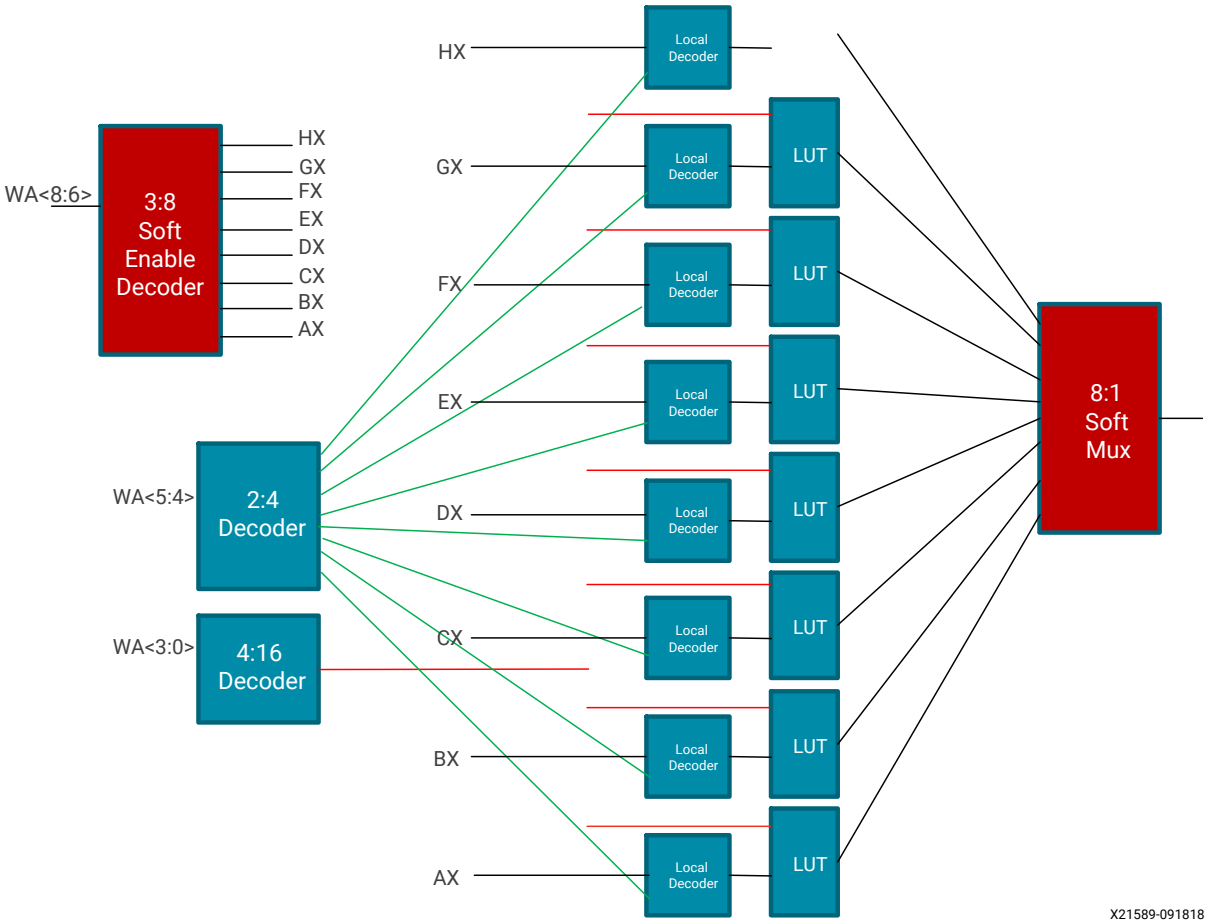
As previously discussed, one half of the LUTs in every CLB are LUTRAM and SRL-capable (see CLB Architecture, Figure 1). In LUTRAM mode, a LUT can be configured as a 32-bit or 64-bit RAM. In SRL mode, a LUT can be configured as a 32-bit or two 16-bit shift register. Distributed RAM can be combined to create LUTRAM or SRL blocks with various size and features.

Address line masking circuitry that exists on a per column of eight LUTs, not on a per LUT basis. This is new to Versal architecture. As a result software packs either SRLs, LUTRAMs, or LUTs in a 8-LUT cluster but not a combination of these. The 4-LUT clusters in a SLICEM are based on columns of LUTs. {A_M,C_M,E_M,G_M} is one cluster, and {B_M,D_M,F_M,H_M} is another. The clusters can contain LUTRAMs, SRLs, or regular logic LUTs. Combinations of more than one LUT type in a 4-LUT cluster are not allowed. Each half of the CLB can have LUTRAM or SRL but not both. If LUTRAM is used in one 8-LUT cluster, the other 8-LUT cluster on the same side can only be regular a LUT or LUTRAM. Similarly, if SRL is used in one 8-LUT cluster the other 8-LUT cluster on the same side can only be a regular LUT or SRL. LUTRAM and SRL can be in the same CLB but they have to be in different sides. Each 8-LUT cluster can only have regular LUTs, LUTs configured as LUTRAM, or LUTs configured as SRL. Combinations of other LUT types in an 8-LUT cluster are not allowed.

Another difference new to Versal architecture is the removal of wide function multiplexers (F7,F8,F9). These were previously used to enable hardened support for deeper LUTRAM and SRL modes. Soft logic (such as LUTs implemented as dynamic multiplexers) must be used for LUTRAMs greater than 64 bits deep as well as SRLs greater than 32 bits that use dynamic tap selection.

External LUTs are also necessary to create the write enable logic. Prior architectures used a hardened write decoder, which enables faster best case timing but no placement flexibility. By requiring the write enable logic to be soft, individual LUTs of a deeper LUTRAM configuration can be shuffled around within a slice or even placed in different slices (although each separate slice would require using the H-LUTs input pins for write address). The following figure illustrates these concepts for a 512x1 LUTRAM. Blue blocks are hardened logic in the CLE, while the red blocks are created from external LUTs. Enable signals are routed to bypass pins associated with each LUTRAM LUT.

Figure 6: **Write Decoder**



Versal architecture supports all LUTRAM modes supported in previous architectures. Distributed RAM can be combined in various ways to store larger amount of data. RAM elements are configurable to implement the following configurations:

- Single-Port 32 x (1 to 16)-bit RAM
- Dual-Port 32 x (1 to 4)-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 1-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

  There is dedicated hardware support for LUTRAM 32x1, 32x2, and 64x1. Anything larger requires soft logic created with LUTs to support read and write address decoding.

Shift Registers

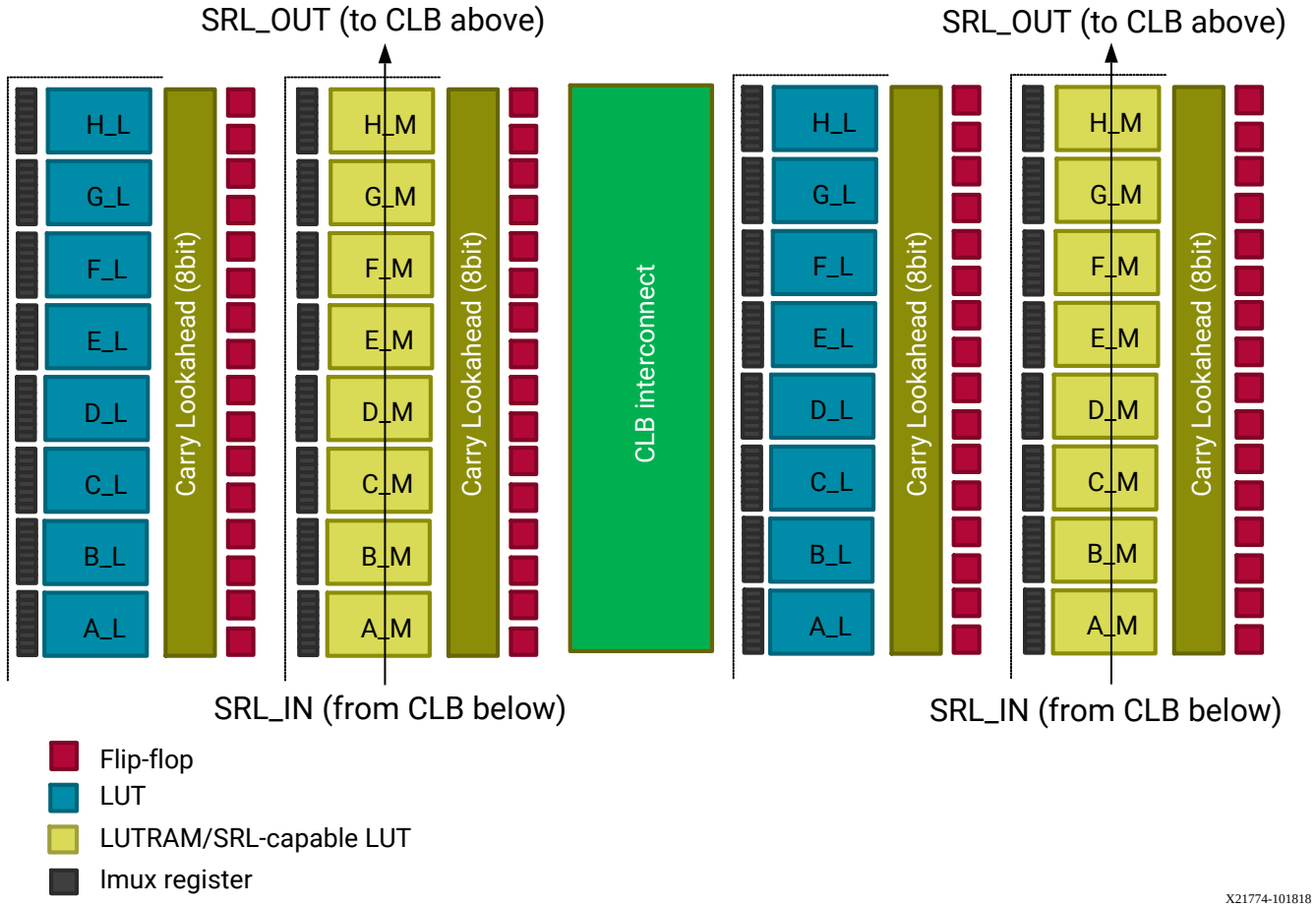
Versal™ architecture CLB shift registers are very similar to those in used previous architecture CLBs, with the 6-LUT supporting a 32-bit SRL and the 5-LUT/6-LUT pair supporting a pair of 16-bit SRLs. The 32-SRLs can be chained together with a dedicated shift chain using the 6-LUT's SIN and SOUT bel pins. Each slice has an initial SIN shift input and SOUT shift output, and which are connected to the slices in the CLB below and above respectively.

Notable SRL-related changes are in how the SRLs are connected:

- The SIN/SOUT shift chain between slices in a column are broken at the center of the clock regions and the clock region boundaries.
- The LUT6 bel output SOUT connects only to to the following LUT6's SIN input, unlike in previous architecturres where that bel pin also could connected through a site output to interconnect.
- SRL cannot be combined with LUTRAM or with LUT in a slice.
- Inter-LUT shift direction is from A->H, the opposite of previous architectures. The intra-LUT shift direction is still LSB->HSB.
- Support for dual-edge clocking.

SRLs are internally cascadeable in a manner similar to the carry chain. They cascade between logically adjacent LUTs (A_M->B_M->C_M->D_M->E_M->F_M->G_M->H_M) and also between CLBs (H_M->A_M). This enables creation of SRLs of arbitrary lengths. There is no memory cell output to logic like in prior architectures, the SRL output can still reach the interconnect if the read decoder of the last LUT in the chain is used to bring out the SRL output through the LUT output pin. The following figure shows how the LUTs can be cascaded to form long SRL chains.

Figure 7: **SRL Cascade**



The shift order is from LUT A_M to LUT H_M which is opposite to what was in previous architectures. However, within a LUT the shift order is still the same as in previous architectures.

Storage Elements

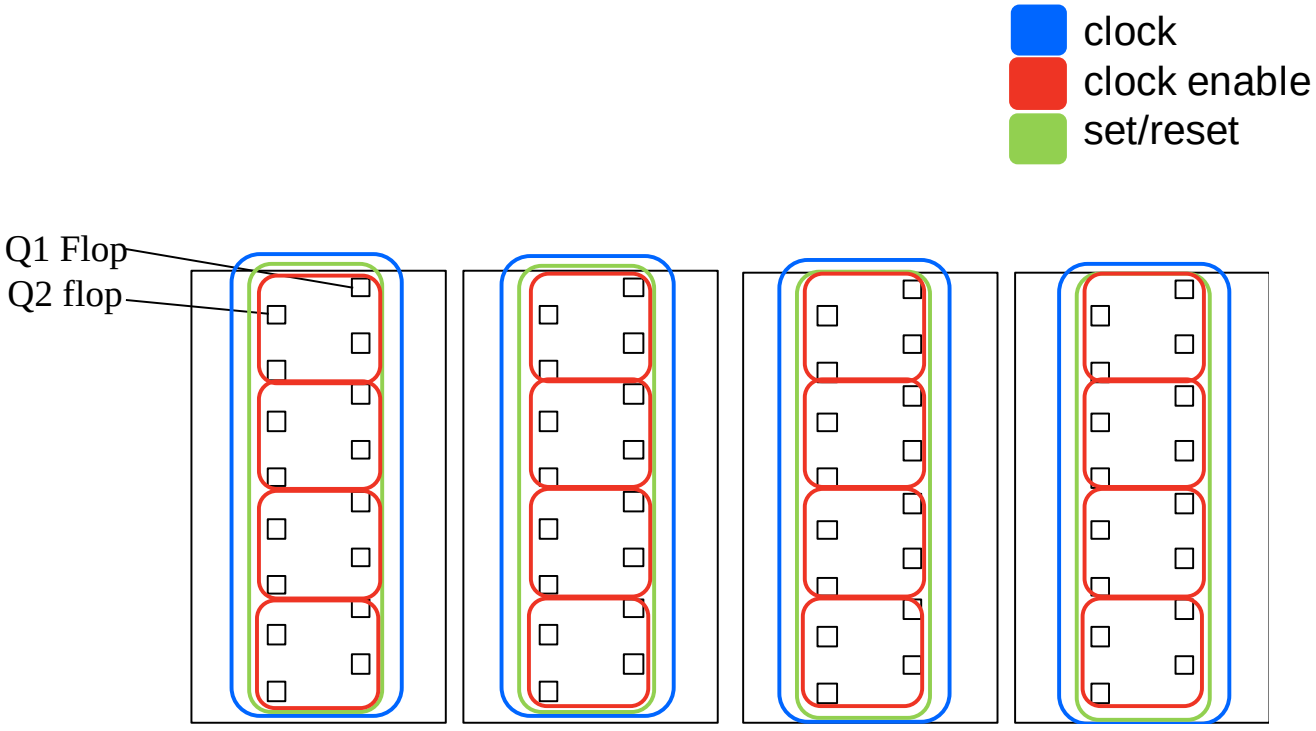
There are 64 slice flip-flops in the CLB. Each can be driven by one of multiple sources:

- O5/sum: LUT output in dual LUT mode and SUM output when using carry logic
- O6/cout: LUT output in 6LUT mode and Carry Out output when using carry logic (only on even bits)
- Bypass: each flop has an independent bypass signal
- Miscellaneous: inputs from super long line SLL connections or carry-out of odd bits

Control Signals

There are 4 clocks, 4 SRs, and 16 CEs for each CLB as shown in the following figure.

Figure 8: **Control Signals**



Q1 and Q2 flip-flops use the same clock enables. This is different than with previous architectures because clock enables were interleaved between Q1 and Q2 flip-flops. Clock and SR are on an 8-bit LUT granularity. In other words, 8 LUTs associated with the same carry chain share a common clock and SR. SRLs use the same clock as the flip-flops associated with that LUT.

SRLs use the same write enable signal (WE) as is used for LUTRAM, which is a slice-level granularity enable signal. SR signals are optionally ignorable on a per slice level.

CE does not have any gating features, as the granularity for CE is already on a per 4 flop basis and a constant 1 can be provided for free in the interconnect control multiplexers feeding the flip-flop CE pins.

For LUTRAM mode, a separate per slice (8 LUTs) write enable is provided; one for the left half of the CLB and one for the other half. The write enable is a separate signal than the clock enable for the flops. The LUTRAM write clock is the same as the clock that drives flops in the same slice. Asynchronous FIFOs can still be constructed using LUTRAM but outputs would have to be registered in a different slice in order for read and write clocks to be different.

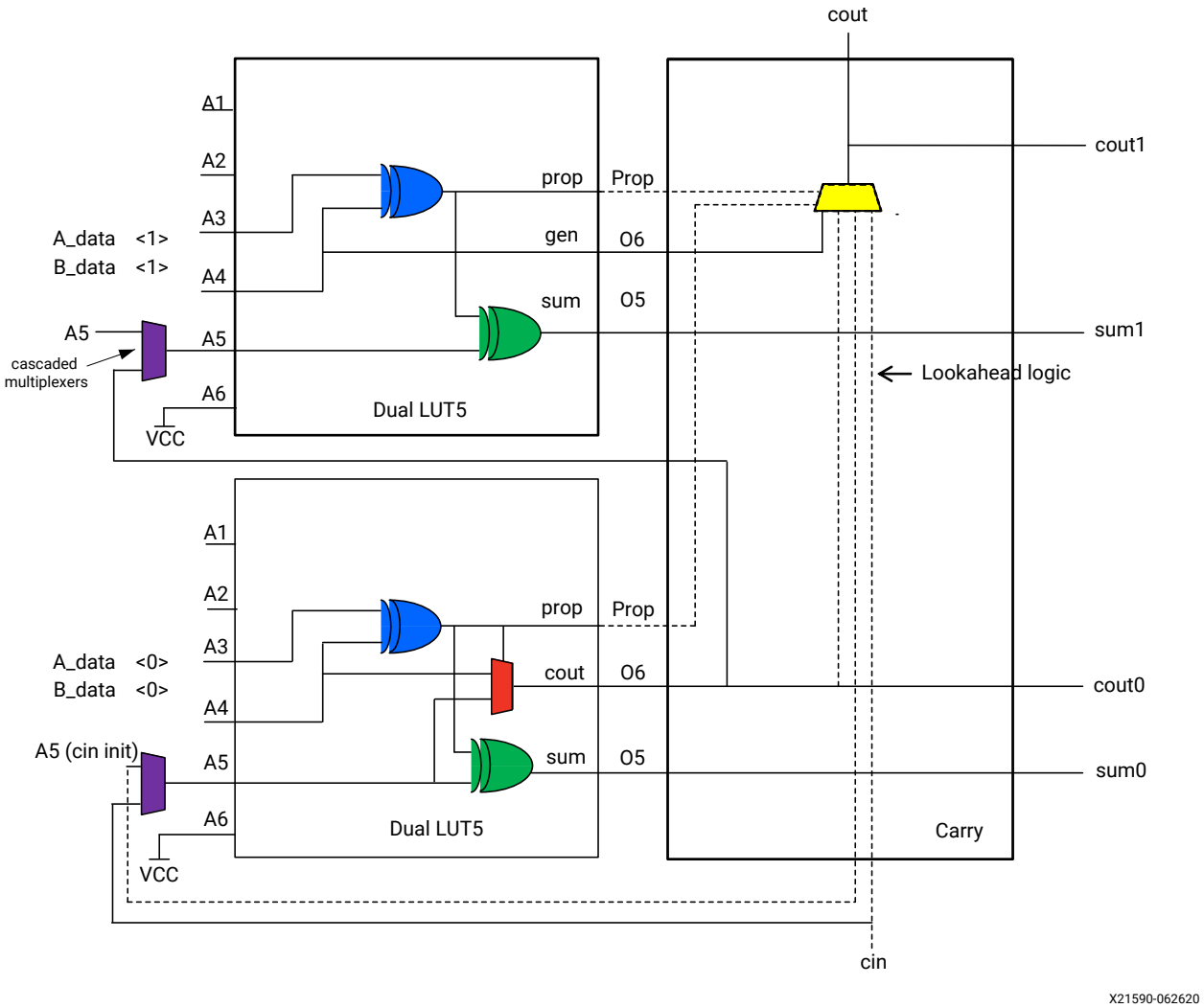
LUTRAM and SRL always use the same clock as CLB flip-flops, as a result the CLB flip-flops cannot be in latch mode in the same slice that uses LUTRAM or SRL. The CLE Imux registers and flip-flops also share the same clock in a slice. Therefore, the latch mode for CLE flip-flop is not compatible with any Imux register usage (hold-fix and pipeline mode).

Carry Logic

Dual 5LUT mode is supported in Versal™ architecture, meaning the carry select and the generate inputs to the carry chain come from the dual 5LUT output pins. Bypass pins no longer feed the carry block. The propagate, sum, and generate functions all are programmed into the LUT. Even LUTs also include the carry select multiplexer. The external carry block only includes carry lookahead multiplexer to accelerate carry propagation delays. Propagate is a sub-function of sum, so an intermediate output (prop) brings the propagate signal out to the carry lookahead logic. There is no longer any dedicated XOR (sum) logic or carry select multiplexer.

The new carry structure is very flexible in terms of carry chain lengths and starting/end points. Because every cascade multiplexer functions as a carry chain initialization multiplexer, any LUT can start or end a carry chain.

Figure 9: **Carry Chain**

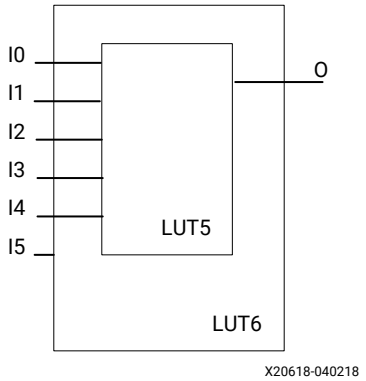


Primitives

LUT Primitives

The LUT primitives allow direct specifications of the look-up table configurations. The example shown is LUT6.

Figure 10: **LUT6 Primitive**



Attributes

Table 1: **Attribute Name, Description, and Possible Values**

Attribute	Description	Values
INIT	Specifies logic expression	64-bit value

Port Descriptions

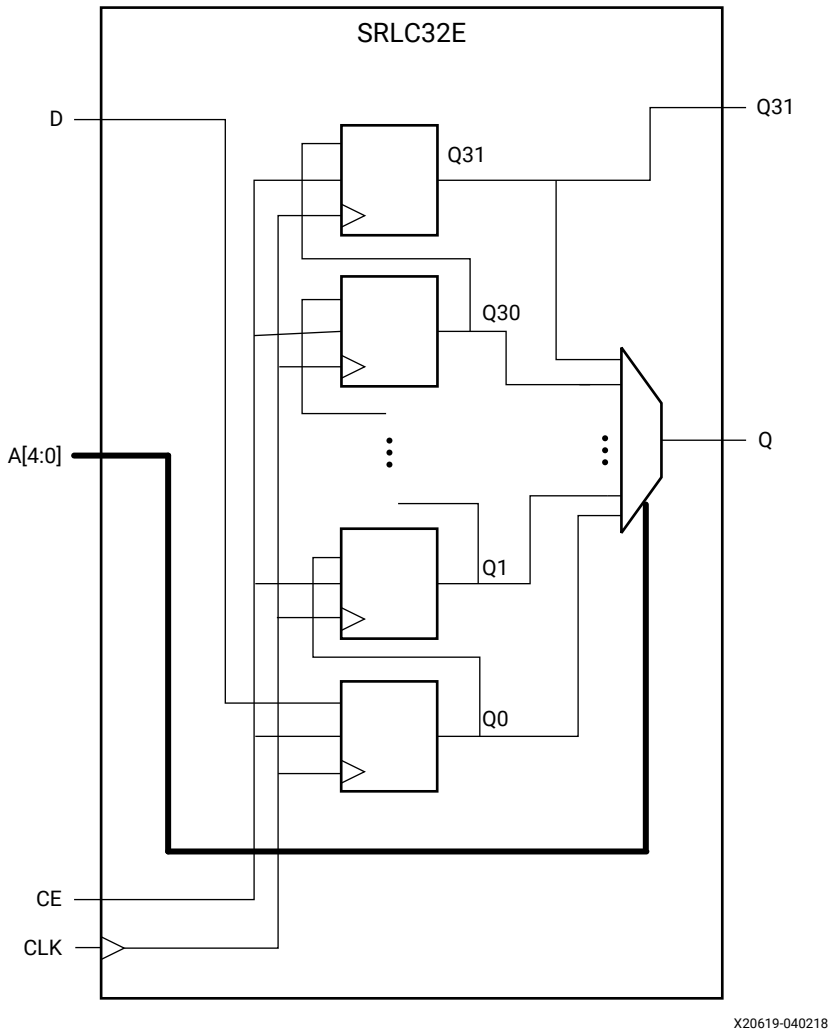
Table 2: **Port Name, Type, and Description**

Port	Type	Description
I0	Input	LUT input
I1	Input	LUT input
I2	Input	LUT input
I3	Input	LUT input
I4	Input	LUT input
I5	Input	LUT input
O	Output	LUT output

SRL Shift Register Primitives

This example of an SRL shift register primitive is the 32-bit shift register (SRLC32E).

Figure 11: **SRL Shift Register Primitive**



Attributes

Table 3: **Attribute Name, Description, and Possible Values**

Attribute	Description	Values
INIT	Specifies logic expression.	64-bit value (hex)
IS_CLK_INVERTED	Specifies whether or not to use the optional inversion on the clock pin (CLK).	1'b0 or 1'b1

Port Descriptions

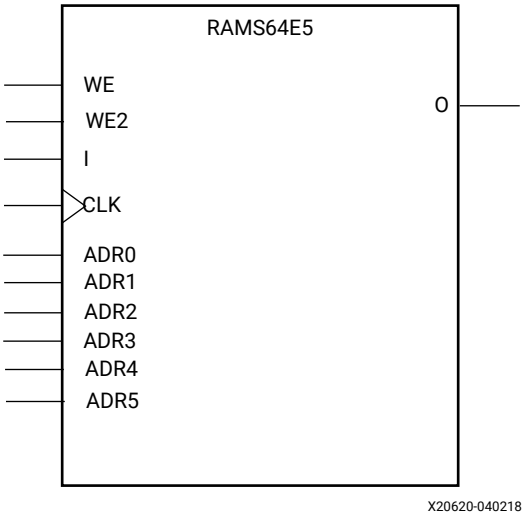
Table 4: **Port Name, Type, and Description**

Port	Type	Description
A<4:0>	Input	The address input selects the bit to be read
CE	Input	Active-High clock enable
CLK	Input	Clock
D	Input	SRL data input
Q	Output	SRL data output selected by the address inputs
Q31	Output	SRL data output, provides the last bit value of the 32-bit shift register

Distributed RAM Primitives

Examples are shown for 64-bit single-port and 64-bit dual-port distributed RAM primitives.

Figure 12: **RAMS64E5 Distributed RAM Primitive (Single-Port)**



Attributes

Table 5: **Attribute Name, Description, and Possible Values**

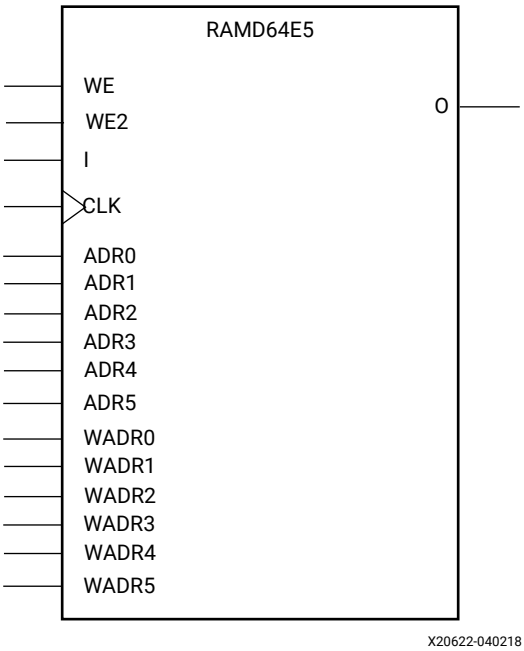
Attribute	Description	Values
INIT	Specifies logic expression.	64-bit value (hex)
IS_CLK_INVERTED	Specifies whether or not to use the optional inversion on the clock pin (CLK).	1'b0 or 1'b1

Port Descriptions

Table 6: **Port Name, Type, and Description**

Port	Type	Description
O	Output	Data output
I	Input	Data input
ADDR0	Input	Address input
ADDR1	Input	Address input
ADDR2	Input	Address input
ADDR3	Input	Address input
ADDR4	Input	Address input
ADDR5	Input	Address input
WE	Input	Write enable
WE2	Input	Additional WE for modes deeper than 64 bits
CLK	Input	Write clock (synchronous)

Figure 13: **RAMD64E5 Distributed RAM Primitive (Dual-Port)**



Attributes

Table 7: **Attribute Name, Description, and Possible Values**

Attribute	Description	Values
INIT	Specifies logic expression.	64-bit value (hex)

Table 7: **Attribute Name, Description, and Possible Values** *(cont'd)*

Attribute	Description	Values
IS_CLK_INVERTED	Specifies whether or not to use the optional inversion on the clock pin (CLK).	1'b0 or 1'b1

Port Descriptions

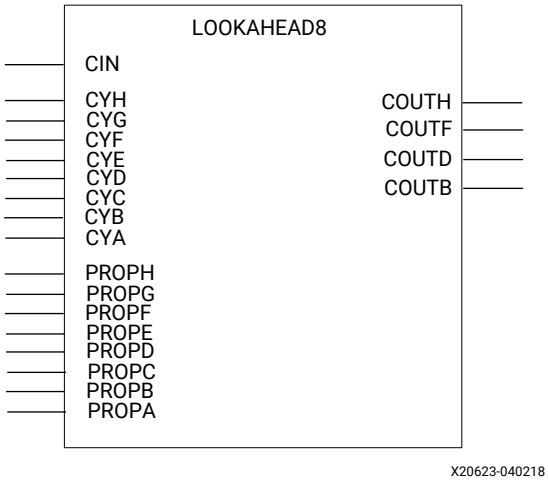
Table 8: **Port Name, Type, and Description**

Port	Type	Description
O	Output	Data output reading from read address
I	Input	Data input
ADDR0	Input	Read address input
ADDR1	Input	Read address input
ADDR2	Input	Read address input
ADDR3	Input	Read address input
ADDR4	Input	Read address input
ADDR5	Input	Write address input
WADR0	Input	Write address input
WADR1	Input	Write address input
WADR2	Input	Write address input
WADR3	Input	Write address input
WADR4	Input	Write address input
WADR5	Input	Write address input
WE	Input	Write address input
WE2	Input	Additional WE for modes deeper than 64 bits
CLK	Input	Write clock (synchronous)

Carry Look-Ahead Logic Primitives

One primitive is available for the carrylookahead8 which containsonly carry look-ahead multiplexers. The following functions are pushed into the LUTS: SUM XOR, CIN multiplexers, and Carry propagation multiplexers.

Figure 14: **Carry Look-Ahead Logic Primitive**



Attributes

Table 9: **Attribute Name, Description, and Possible Values**

Attribute	Description	Values
LOOKB	If TRUE, enable LOOKAHEAD to bring carry results from last stage.	TRUE or FALSE
LOOKD	If TRUE, enable LOOKAHEAD to bring carry results from last stage.	TRUE or FALSE
LOOKF	If TRUE, enable LOOKAHEAD to bring carry results from last stage.	TRUE or FALSE
LOOKH	If TRUE, enable LOOKAHEAD to bring carry results from last stage.	TRUE or FALSE

Port Descriptions

Table 10: **Port Name, Type, and Description**

Port	Type	Description
COUTB	Output	Output of CLA multiplexer.
COUTD	Output	Output of CLA multiplexer.
COUTF	Output	Output of CLA multiplexer.
COUTH	Output	Output of CLA multiplexer.
CYA	Input	Input of CLA multiplexer.
CYB	Input	Input of CLA multiplexer.
CYC	Input	Input of CLA multiplexer.
CYD	Input	Input of CLA multiplexer.
CYE	Input	Input of CLA multiplexer.
CYF	Input	Input of CLA multiplexer.
CYG	Input	Input of CLA multiplexer.

Table 10: **Port Name, Type, and Description** *(cont'd)*

Port	Type	Description
CYH	Input	Input of CLA multiplexer.
PROPA	Input	Input of CLA multiplexer.
PROPB	Input	Input of CLA multiplexer.
PROPC	Input	Input of CLA multiplexer.
PROPD	Input	Input of CLA multiplexer.
PROPE	Input	Input of CLA multiplexer.
PROPF	Input	Input of CLA multiplexer.
PROPG	Input	Input of CLA multiplexer.
PROPH	Input	Input of CLA multiplexer.
CIN	Input	Input of CLA multiplexer.

XILINX

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open 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 DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

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

Please Read: Important Legal Notices

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

This document contains preliminary information and is subject to change without notice. Information provided herein relates to products and/or services not yet available for sale, and provided solely for information purposes and are not intended, or to be construed, as an offer for sale or an attempted commercialization of the products and/or services referred to herein.

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