Speedcore Gen4 eFPGA Datasheet (DS012)

Speedcore eFPGA IP

Achronix®

Data Acceleration

Copyrights, Trademarks and Disclaimers

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Table of Contents

Chapter - 1: Overview

Introducing Speedcore Gen4 eFPGA

Achronix's Speedcore™ Gen4 embedded FPGA (eFPGA) IP, which includes look-up-table, memory, DSP, and machine learning processor (MLP) building blocks. Each of these blocks are designed to be modular to allow customers to define any mix of resources required for their end system.

Achronix delivers the Speedcore Gen4 IP in GDSII format along with all files and documentation required for the customer to integrate their Speedcore eFPGA instance into their ASIC. Achronix also delivers the supporting ACE design tools that are used to compile designs targeting their Speedcore eFPGA.

Feature Summary

Because Speedcore Gen4 eFPGA is an embeddable IP, it is designed to be completely surrounded by the end user ASIC (see the figure below). A Speedcore eFPGA includes the following features.

- Programmable core fabric, with customer defined functionality
- Core I/O ring
- FPGA configuration unit (FCU)
- Configuration memory (CMEM)
- Interfaces for debug and programming
- Interface for test (DFT)

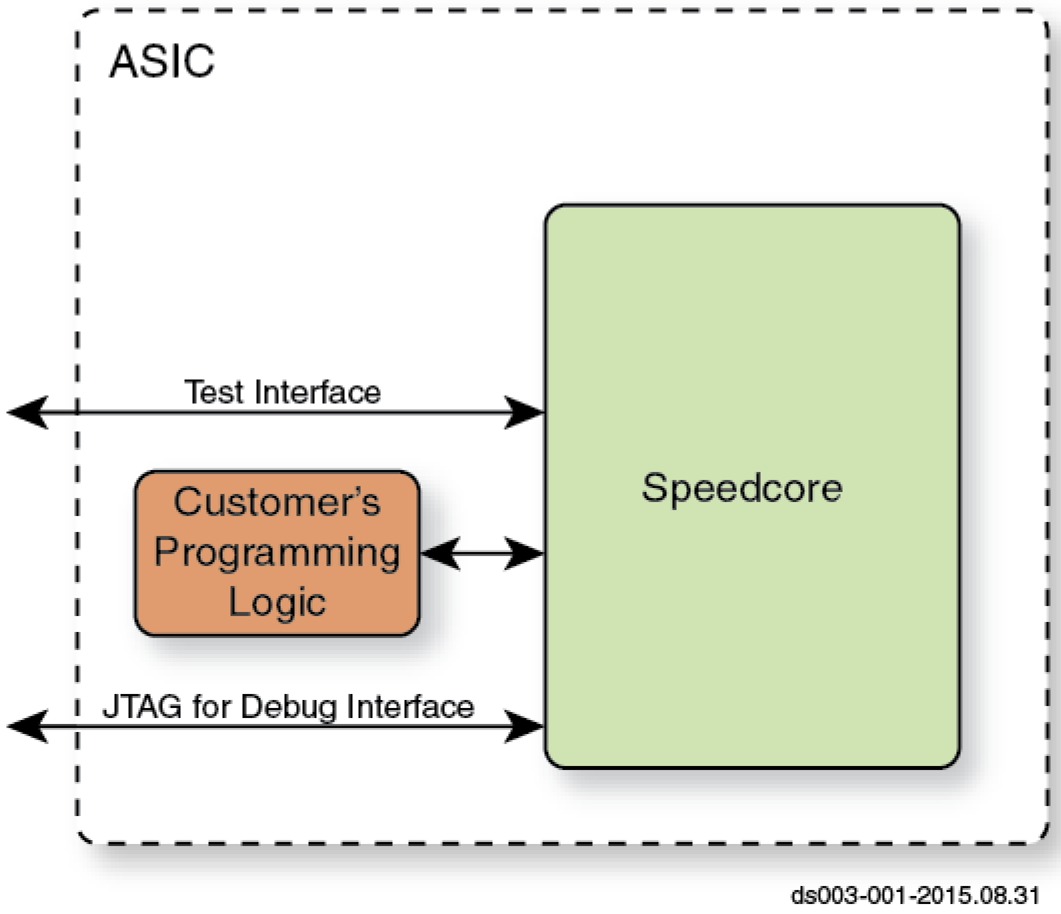


Figure 1: ***Embedded Speedcore***

Functionality

Customers define the functionality of their Speedcore Gen4 eFPGA by choosing the quantity of each of the resources listed below:

- **Logic** – 6-input look-up-tables (LUTs) plus integrated fast adders, and multiplier LUT (MLUT) mode for efficient multiplies
- **Logic RAM** – up to 2 kb per memory block for LRAM2k, and up to 4kb per memory block for LRAM4k
- **Block RAM** – up to 72 kb per memory block for BRAM72k, and up to 20kb per memory block for BRAM20k
- **DSP64** – each block has a 18 × 27 multiplier, 64-bit accumulator and 27-bit pre-adder
- **MLP** – optimized for machine learning, each block has up to 16 multipliers, deep adder trees and accumulators

Note

The number and mix of resource blocks for each Speedcore eFPGA instance is based on customer requirements.

Process Technology

Speedcore Gen4 eFPGA IP is currently available on the TSMC N7 process node and the TSMC 16FF+ GL process node.

Programming

Customers can select the programming interface to be one or a combination of the following available options:

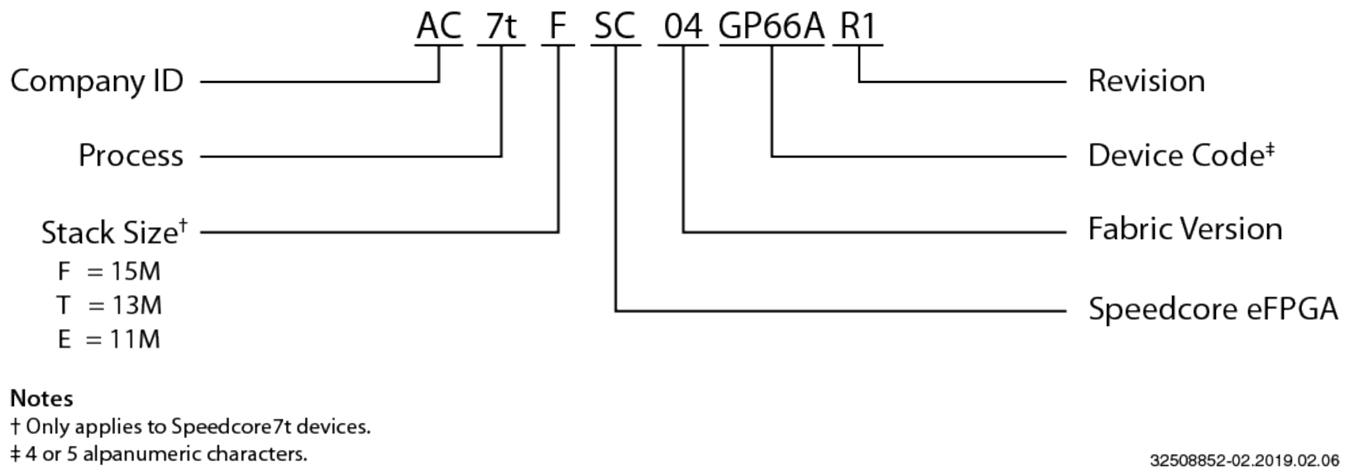
- JTAG
- Parallel CPU (×1, ×8, ×16, ×32, ×128 data width modes)
- Serial flash (1 or 4 flash devices)
- AXI 128-bit

Security

Speedcore eFPGAs support AES-encrypted bitstreams.

IP Nomenclature

For ease of identification, each Speedcore configuration carries a unique part number, based on the following nomenclature:



Chapter - 2: Speedcore Gen4 Architecture

Fabric Architecture

The Speedcore eFPGA fabric is built from a library of tiles. Each tile consists of a routing switch box plus a logic block which can consist of LUTs, memory, DSPs, etc. Each type of block is designed to snap together in a grid, where abutting routing networks connect. Various tile flavors are assembled to deliver the desired fabric size and resource mix. All connections between tiles are accomplished via abutment and guaranteed to be DRC/LVS clean for any combination of tile flavors.

Block Floorplan

The Speedcore floorplan is arranged with the various block functions arranged in columns. The block functions are connected by a uniform global interconnect, which enables the routing of signals between core elements. Switch boxes make the connection points between vertical and horizontal routing tracks. Inputs to and outputs from each of the functions connect to the global interconnect.

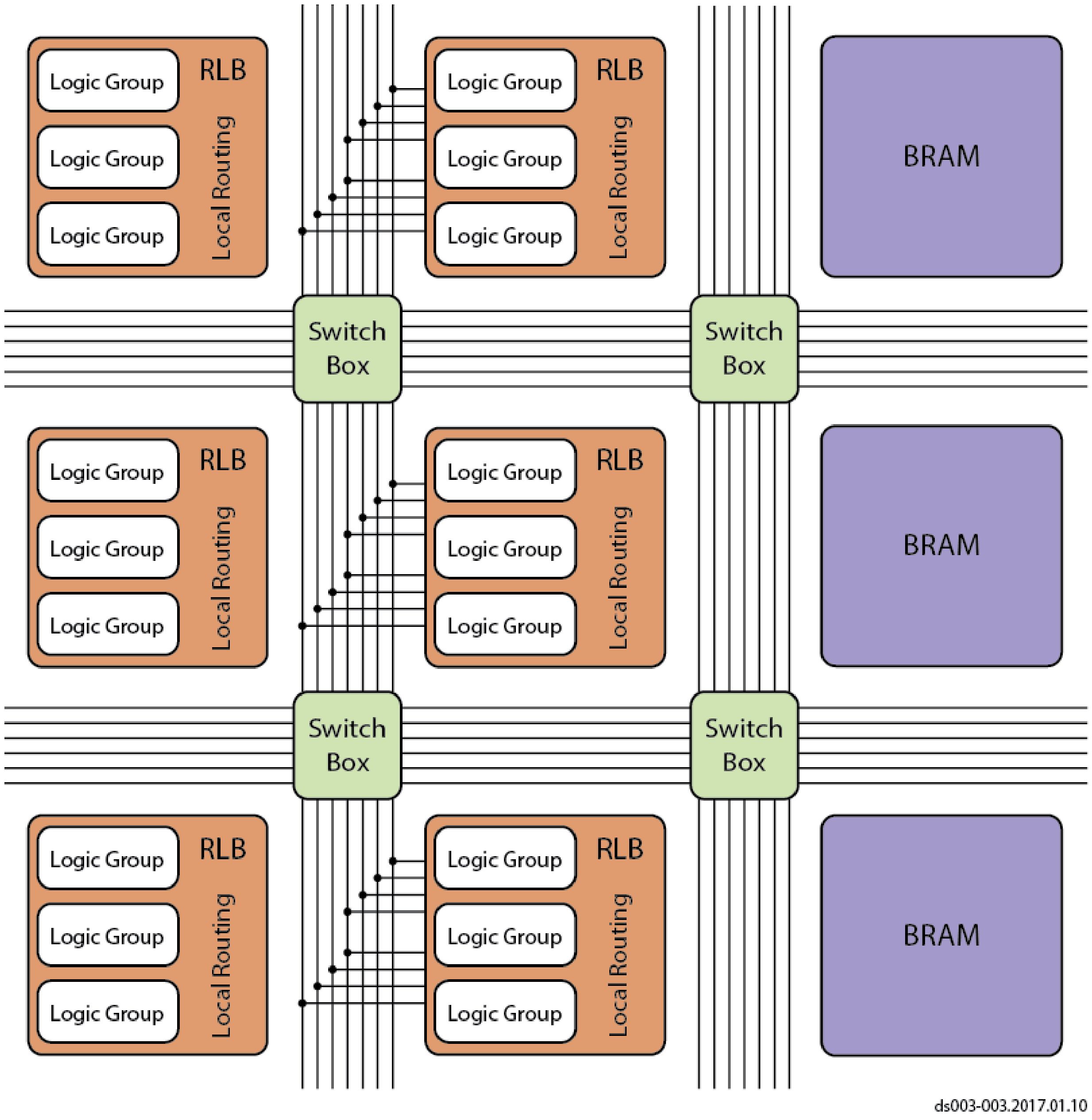


Figure 2: ***Speedcore Gen4 eFPGA Interconnect***

Speedcore Gen4 Clock Network

Speedcore eFPGAs have two types of clock networks targeted to provide both the low-skew, balanced architecture as well as addressing the source synchronous nature of data transfers with external interfaces.

The global clock network is the hierarchical network that feeds resources in the eFPGA fabric. The global clock trunk runs vertically up and down the center of the core (gray stripe in the following figure), sourced by global clock muxes at the top and bottom of the global trunk. The sources driven down the trunk are then channeled out the balanced clock mini-trunks to both the left and right halves of the core.

Within Speedcore eFPGAs, there is a second clock network available at the periphery of core, the interface clock network. As the name implies, the intent of these clocks is to facilitate the construction of interface logic within the eFPGA core operating on the same clock domain as local logic in the surround host ASIC. The clocks connect to the core through the surrounding interface clusters, allowing for clock signals to be driven both into and out of the core.

These two networks are shown in the two figures below.

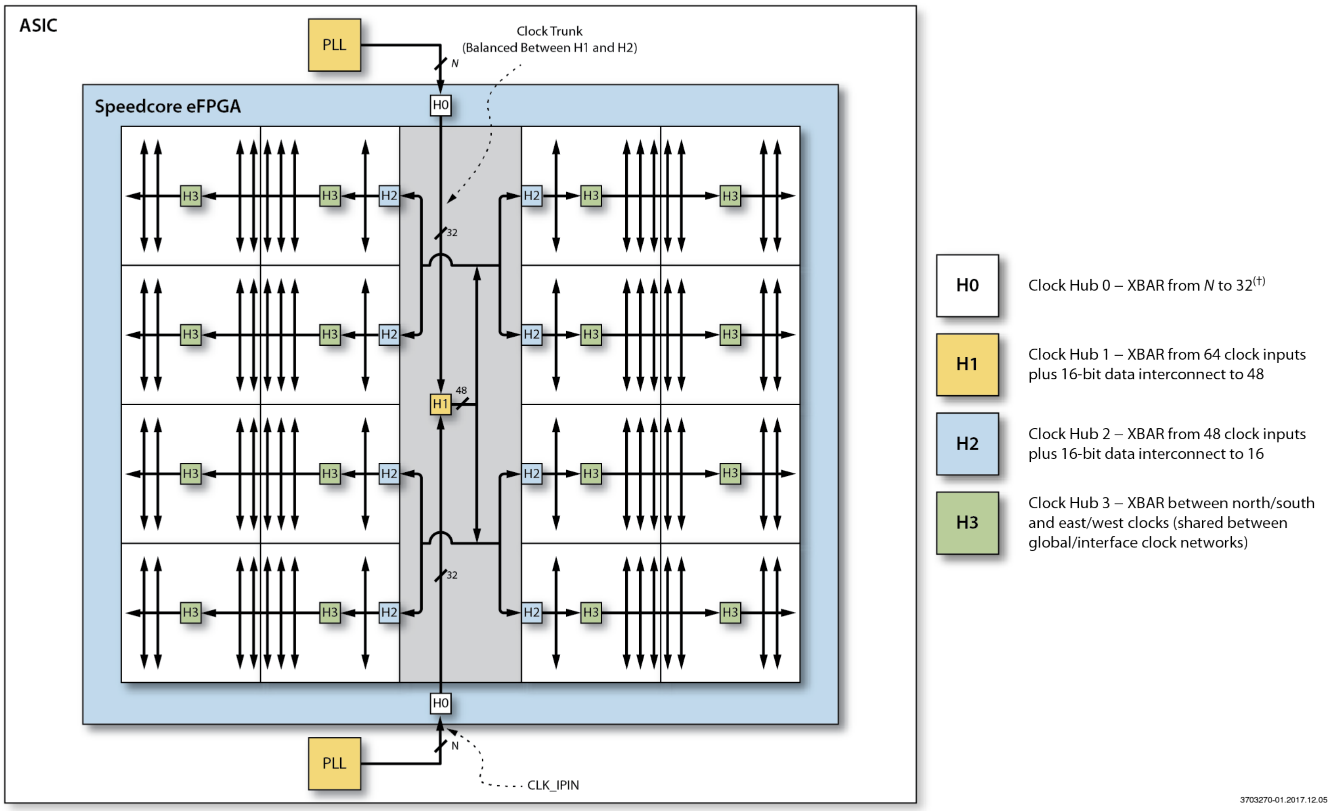

ASIC
PLL
Clock Trunk (Balanced Between H1 and H2)
N
Speedcore eFPGA
H0
H1
H2
H3
32
48
CLK_IPIN
H0 Clock Hub 0 – XBAR from N to 32(†)
H1 Clock Hub 1 – XBAR from 64 clock inputs plus 16-bit data interconnect to 48
H2 Clock Hub 2 – XBAR from 48 clock inputs plus 16-bit data interconnect to 16
H3 Clock Hub 3 – XBAR between north/south and east/west clocks (shared between global/interface clock networks)
3703270-01.2017.12.05

Figure 3: ***Global Core Clock Network***

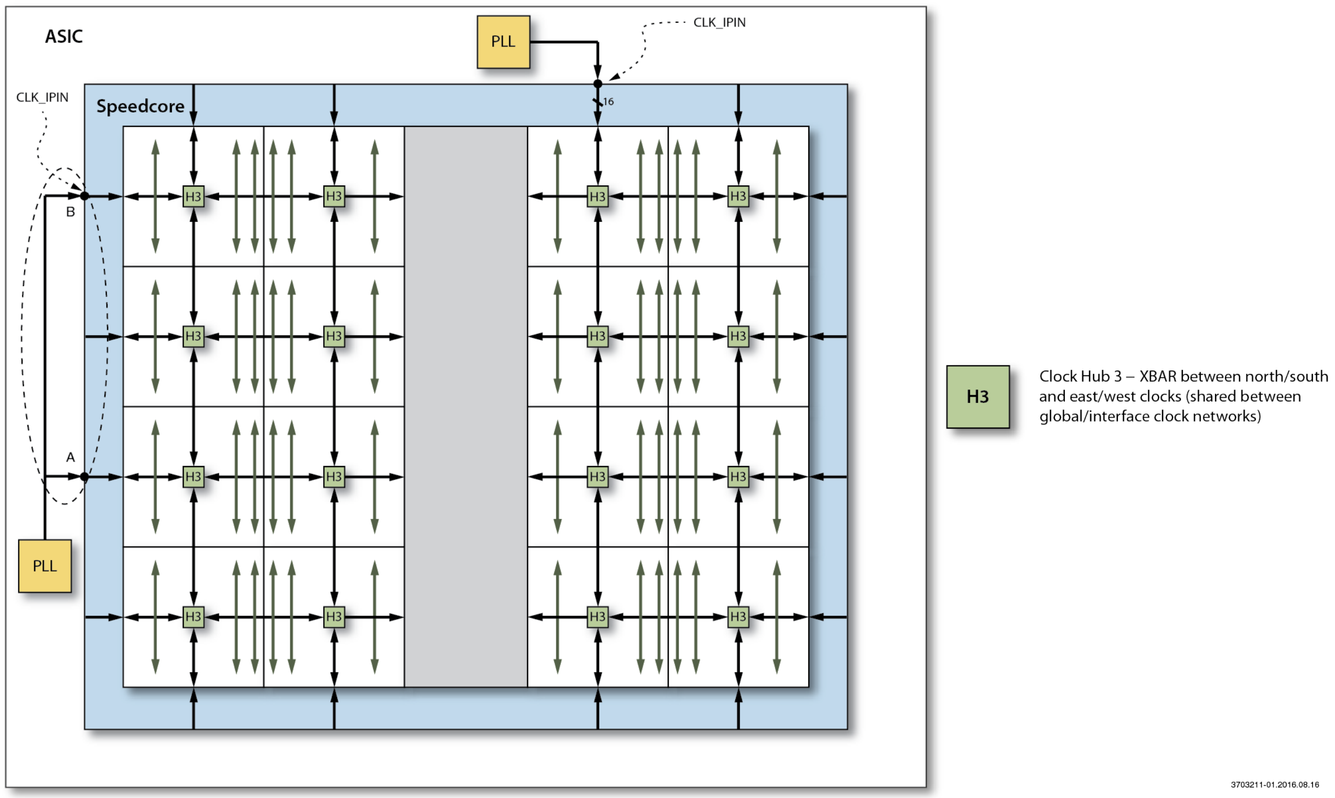

ASIC
PLL
CLK_IPIN
CLK_IPIN
Speedcore
16
B
A
PLL
H3
H3
Clock Hub 3 – XBAR between north/south and east/west clocks (shared between global/interface clock networks)
3703211-01.2016.08.16

Figure 4: ***Interface Clock Network***

Speedcore Gen4 Interface Cluster

The interface cluster is the portion of the Speedcore Gen4 architecture boundary ring that contains the registers, Achronix configuration bus (ACB) logic, and the connectivity to the Speedcore Gen4 top-level pins. The figure below shows the details of an interface cluster, illustrating the ingress and egress path for user signals to the core (both paths can be optionally registered). The second figure depicts the clock tile, showing how clocks enter and exit the fabric, including optional delays.

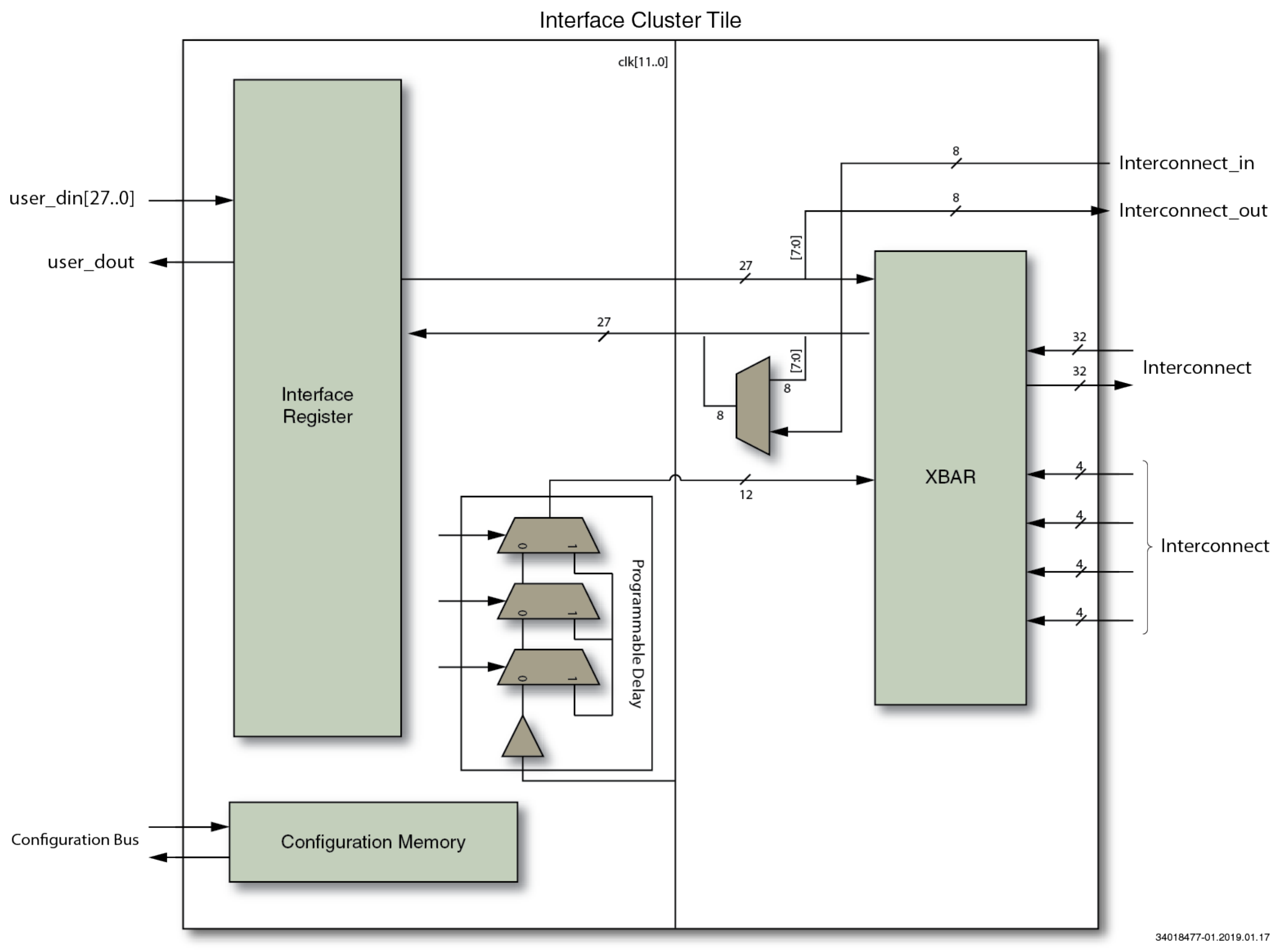


Figure 5: ***Interface Cluster Logic Tile***

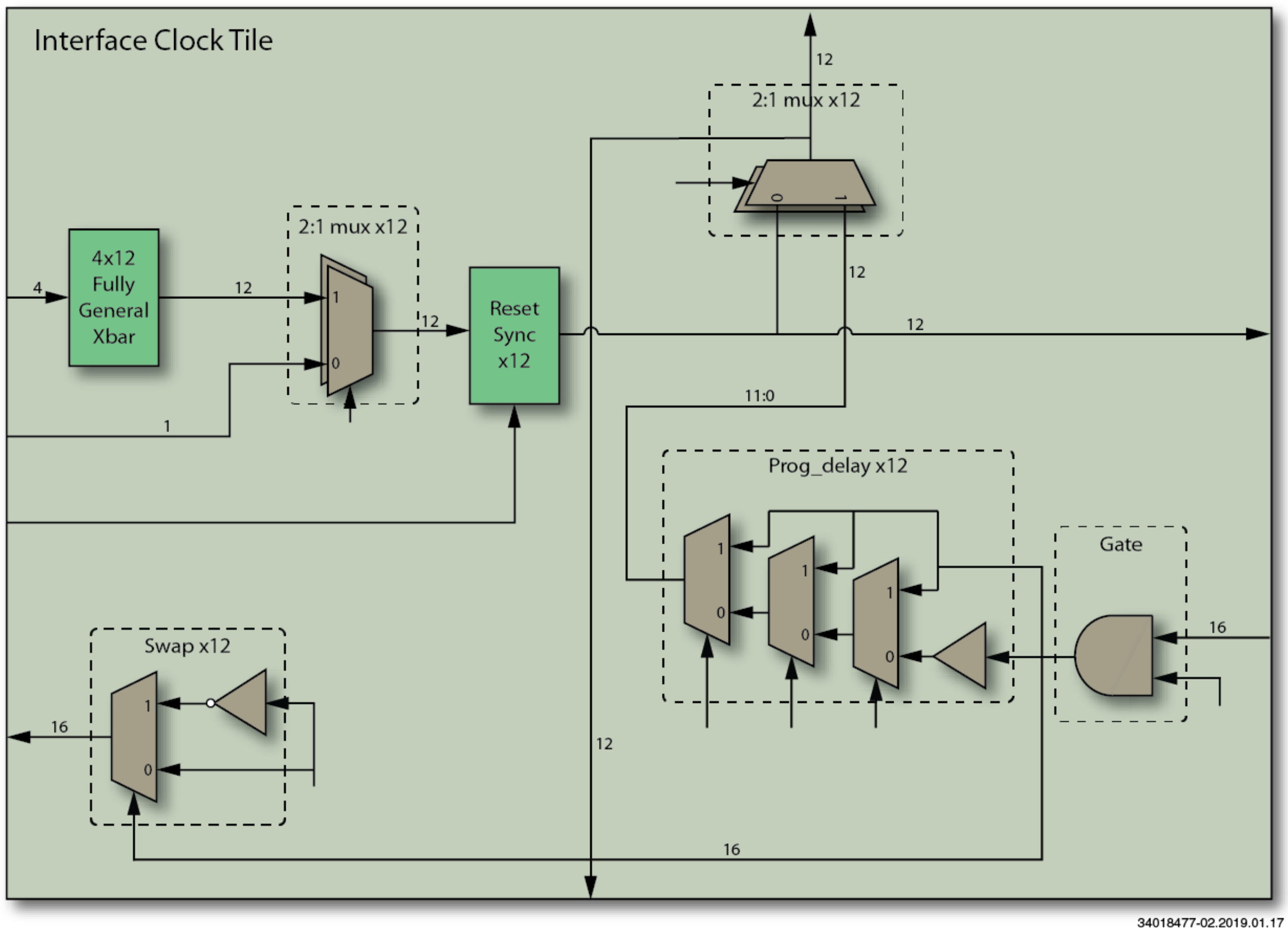


Figure 6: *Interface Cluster Clock Tile*

Interface Timing Closure

The Speedcore Gen4 interface clusters do not have programmable I/O that connect directly to the pads on the host ASIC. Instead the Speedcore Gen4 eFPGA IP supports a large number of interface clusters which connect directly to logic signals within the host ASIC. The exact number of interface I/O is dependent upon the size of the Speedcore Gen4 eFPGA instance specified.

The timing of signals from the host ASIC to any embedded eFPGA is crucial in enabling signals to close timing at the high frequencies desired for 16nm or 7nm technology. The variation in clock skew between the host ASIC and an embedded eFPGA's internal clock structures must be fully accounted for and carefully engineered. To enable this timing closure, Achronix has three different timing scenarios which enable customers to configure the optimum timing path architecture for different I/O groups.

Speedcore Gen4 Logic Fabric - Reconfigurable Logic Block

The 6-input LUT based reconfigurable logic block (RLB6) is composed of three parallel logic groups as shown in the diagram below.

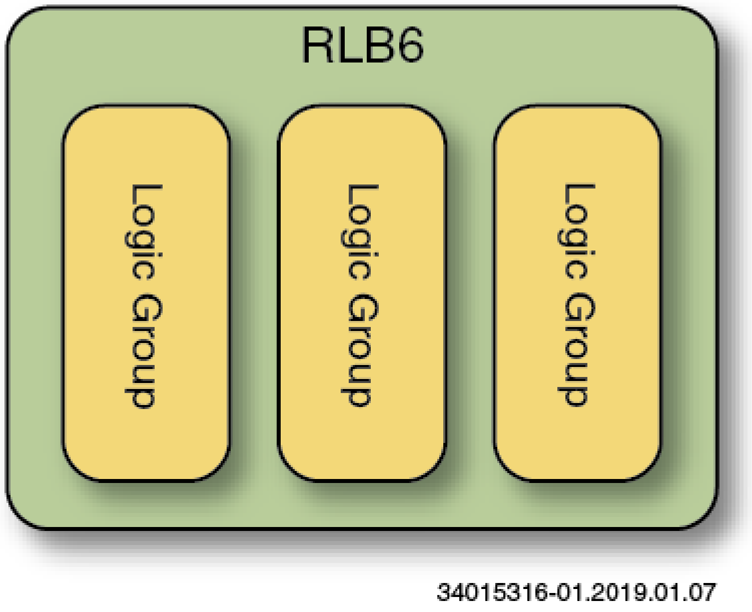


Figure 7: ***RLB6 Block Diagram***

Each logic group contains four 6-input look-up-tables (LUT), each with two optional registers and an 8-bit fast arithmetic logic unit (ALU) to implement logic functionality. Each logic group receives a carry-in input from the corresponding logic group in the RLB6 to the north and can propagate a carry-out output to the corresponding logic group in the RLB6 to the south.

The table below provides information on the resource counts inside an RLB6.

Table 1: ***RLB6 Resource Counts***

RLB6 Resource	Count
Logic Groups	3
6-LUTs	12
Registers	24
8-bit ALU	3

The following features are available using the resources in the RLB6:

- 8-to-1 MUX with single-level delay (using the 6-LUTs rather than dedicated MUX4 or MUX8 logic)
- 8-bit ALU for adders, counters, and comparators
- MAX function that efficiently compares two 8-bit numbers and chooses the max or min result
- Dedicated connections for high-efficiency shift registers
- Multiplier LUT (MLUT) mode for efficient multipliers

The figure below provides details on the circuitry inside a single logic group.

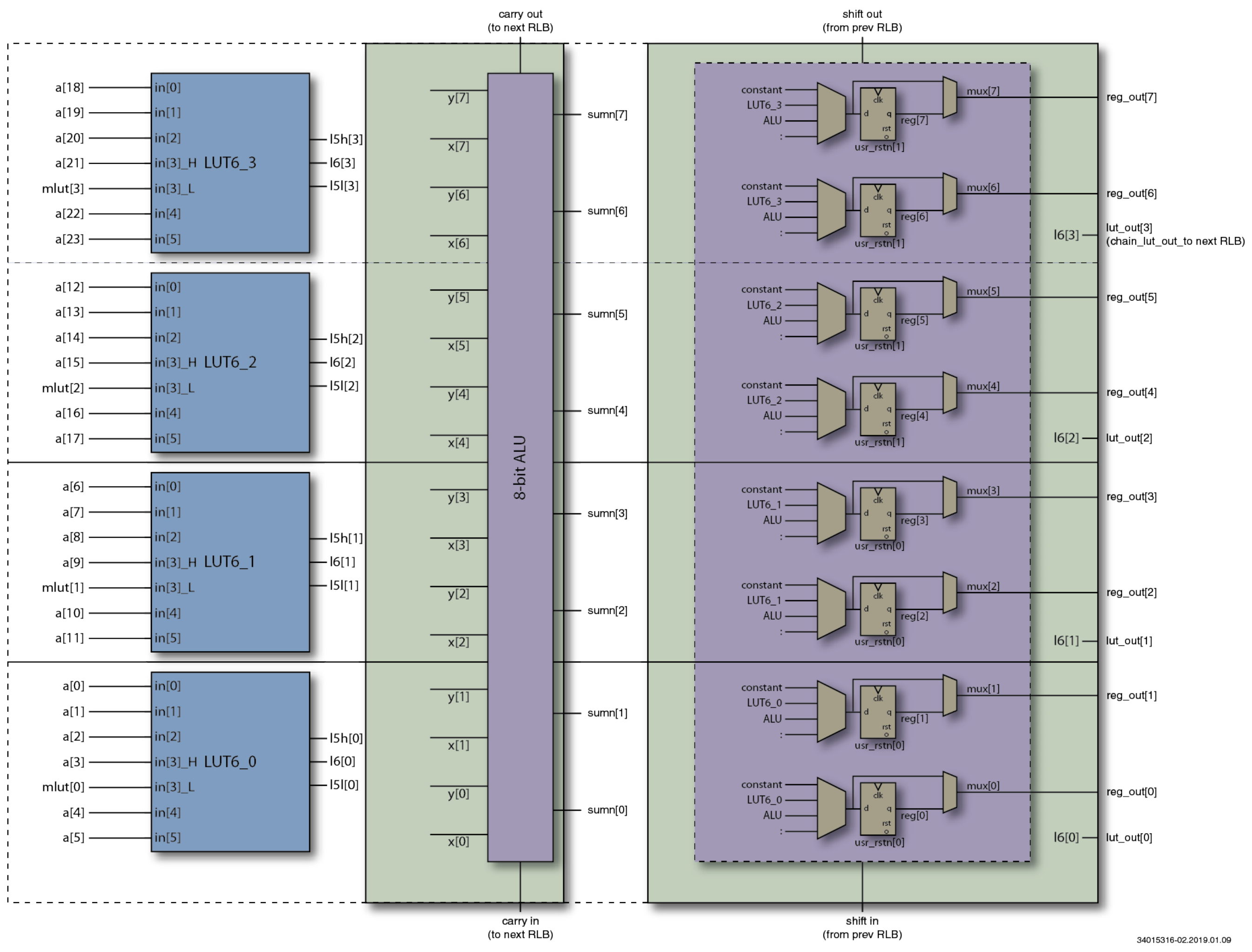


Figure 8: ***RLB6 Logic Group Details***

MLUT Mode

The RLB-6 includes an MLUT mode for an efficient LUT-based multiplication. MLUT mode results in 2 × 4 multiplier building blocks that can be stacked horizontally and vertically to generate any size signed multiplier. For example, a 2 × 8 multiplier building block can be generated with two 6-LUTs, and one RLB6 can perform a 6 × 6 multiply.

Note

MLUT mode is supported by the MLUT generator to help customers build the multiplier desired.

Routing Between RLB6s

There are special considerations when routing ALU carry chains and shift registers. The Speedcore Gen4 fabric has hard-wired connections on the signals `carry_in/carry_out` of each ALU. As mentioned above, each logic group routes to the corresponding logic group in the RLB6 above or below. In other words, the ALU `carry_in/carry_out` does not route to the next ALU within the same RLB, but rather the same logic group of the next RLB6. The figure below shows the `carry_in/carry_out` routing of an ALU.

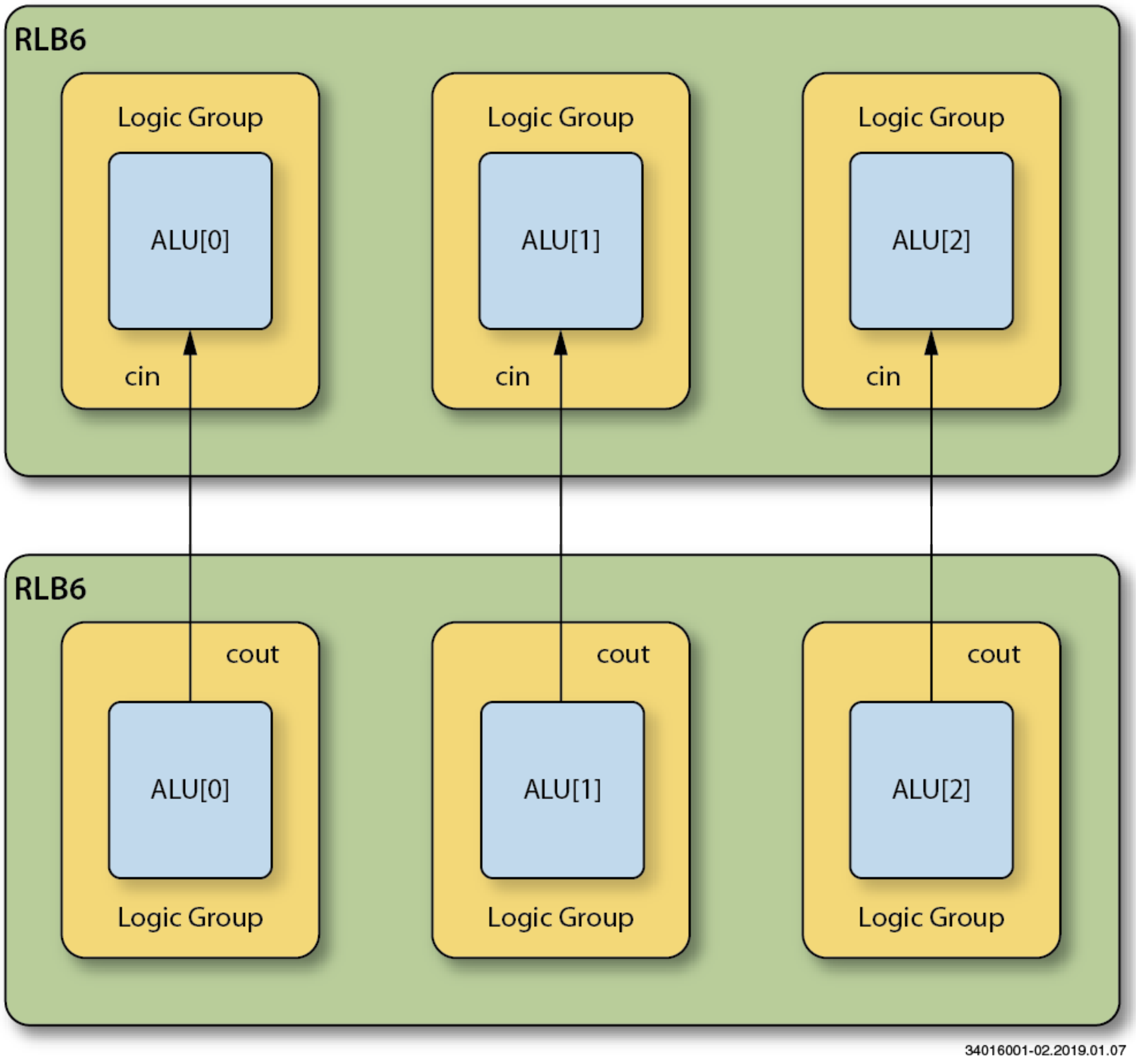


Figure 9: ***ALU Carry Chain Routing***

The same is true for the signals `shift_in/shift_out` in the registers of a logic group. When creating a shift register, the registers within a logic group route to each other, but the `shift_in/shift_out` of each logic group routes to the same logic group in the next RLB6. The figure below shows details of the routing in the Speedcore Gen4 fabric.

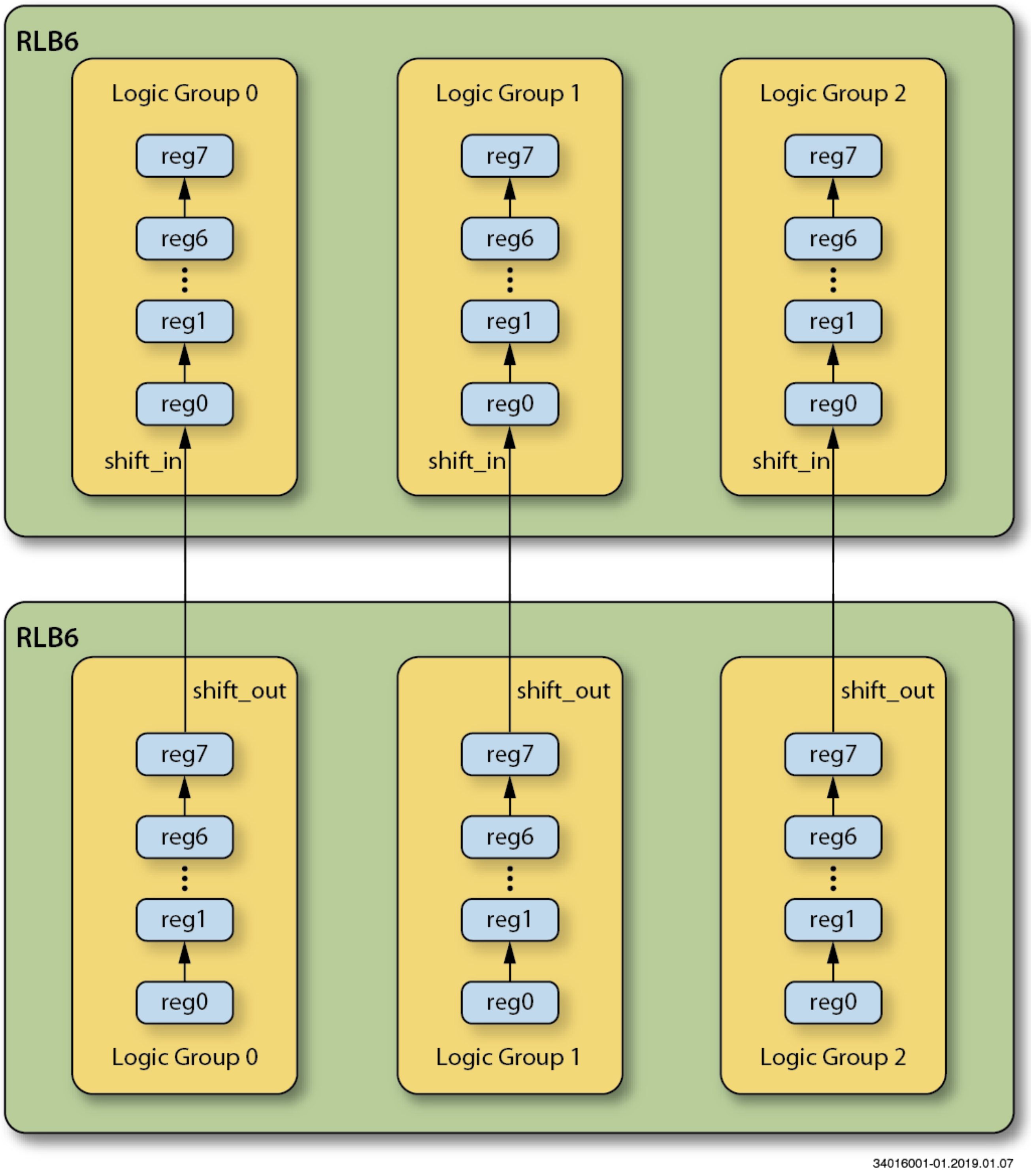


Figure 10: ***Shift Register Routing***

Speedcore Gen4 Block RAM

Block RAM 20k

The BRAM20k implements a dual-ported memory block where each port can be independently configured with respect to size and function. The BRAM20k can be configured as a single-port (one read/write port), dual-port (two read/write ports with independent clocks), or ROM memory. The key features of the BRAM20k are summarized in the table below.

Table 2: ***BRAM20k Key Features***

Feature	Value
Block RAM Size	20 kb
Organization	512 × 40(†), 1k × 20, 1k × 18, 1k × 16, 2k × 10, 2k × 9, 2k × 8, 4k × 5, 4k × 4, 8k × 2, 16k × 1
Physical Implementation	Columns throughout device
Number of Ports	Dual port (independent read and write)
Port Access	Synchronous

Note

† 512 × 40 only available as simple dual-port function.

The BRAM20k ports are illustrated in the following figure:

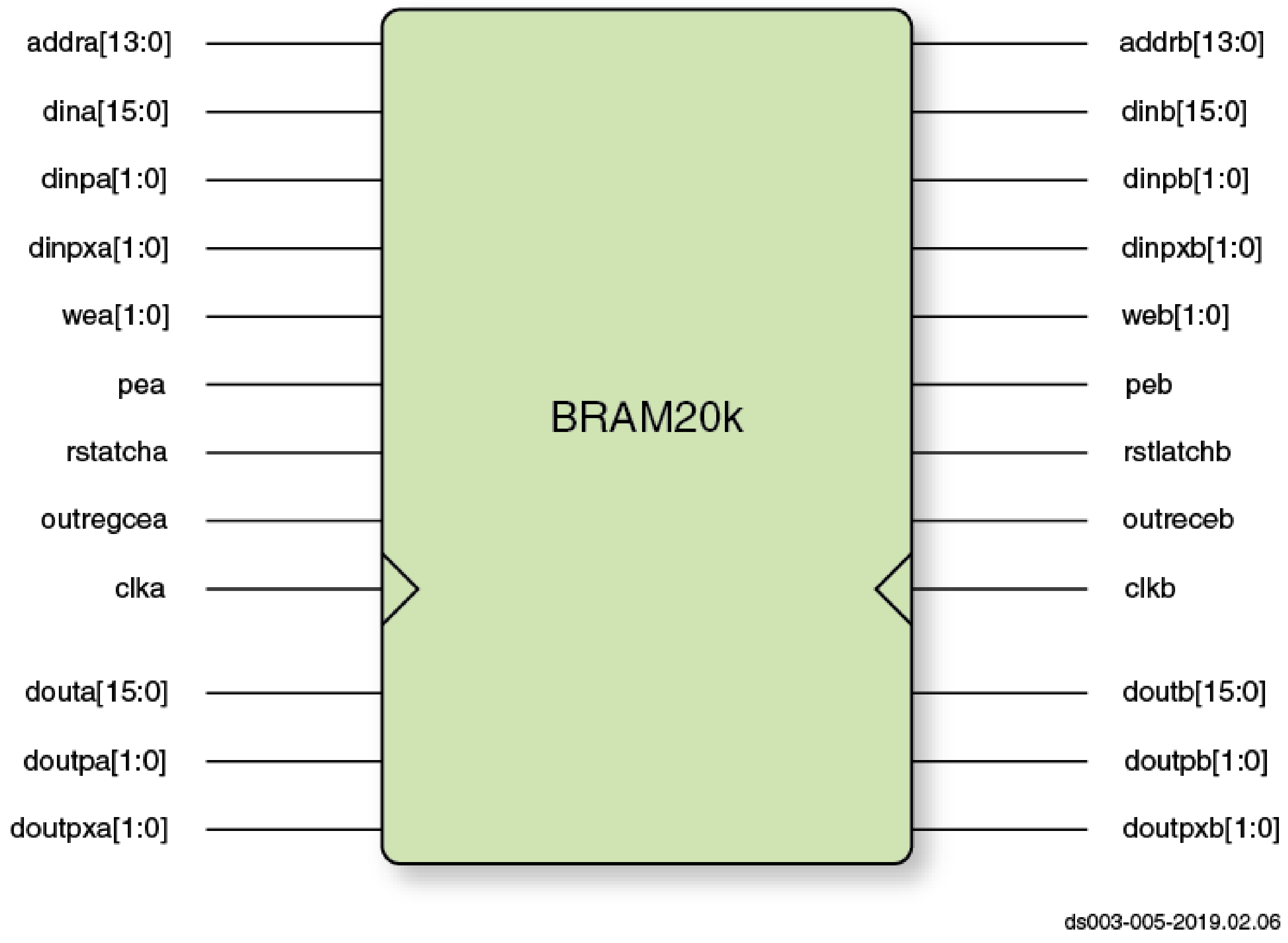


Figure 11: ***BRAM20k Ports***

Organization

The organization of each BRAM20k port can be independently configured.

Note

Access from opposite ports is not required to have the same organization; however, the number of total memory bits on each port must be the same.

Operation

The read and write operations are both synchronous. For higher performance operation, an additional output register can be enabled, which will add an additional cycle of read latency. The initial value of the memory contents may be specified by the user from either parameters or a memory initialization file. The initial/reset values of the output registers may also be specified by the user. The reset values are independent of the initial (powerup) values. The `porta_write_mode/portb_write_mode` parameters define the behavior of the output data port during a write operation. When `porta_write_mode/portb_write_mode` is set to `write_first`, the `douta/doutb` is set to the value being written on the `dina/dinb` port during a write operation. Setting `porta_write_mode/ portb_write_mode` to `no_change` keeps the `douta/ doutb` port unchanged during a write operation to `porta/portb`. Conflict arises when the same memory cell is accessed by both ports within a narrow window and one or both ports are writing to memory. If this condition occurs, the contents of the memory and the output data for the colliding address may be undefined, but no damage will occur to the core.

Built in FIFO Controller

The BRAM20kFIFO implements a 20 kb FIFO memory block utilizing the embedded BRAM20k blocks with dedicated pointer and flag circuitry. The BRAM20kFIFO can be configured to support a variety of widths and depths, ranging from 512-bit depth with 40-bit data down to 16k depth with 1-bit data. The read and write clocks may be either synchronous or asynchronous with respect to each other. If the user read and write clocks are the same clock, the user may set the sync_mode to 1'b1 to enable faster and synchronous generation of the status flags and FIFO pointer outputs.

Error Correction

Error correction is available only in 40-bit (simple dual-port) mode. The built-in error correction logic provides single-bit error correction and dual-bit error detection on a 32-bit data bus, using eight internal overhead bits. If the internal ECC logic is not used, all 40 bits can be used for other purposes such as tagging and various control functions.

Initialization and Reset

Initial content of the BRAM20k can be optionally loaded during device configuration if specified by the user. Otherwise, the BRAM20k initial content is undefined. On reset, the RAM contents are unchanged, but the output register, if used, assumes the specified reset value.

The initial state of the RAM read outputs can also be optionally loaded during device configuration.

Block RAM 72k

The BRAM72k primitive implements a 72-kb simple-dual-port (SDP) memory block with one write port and one read port. Each port can be independently configured with respect to size and function, and can use independent read and write clocks. The BRAM72k can be configured as a simple dual port or ROM memory. The key features (per block RAM) are summarized in the table below.

Table 3: ***BRAM72k Key Features***

Feature	Value
Block RAM size	72 kb
Organization	1024 × 72, 2048 × 36, 4096 × 18, 8192 × 9, 16384 × 4

Feature	Value
Physical Implementation	Columns throughout device
Number of Ports	Simple Dual Port (independent read and write)
Port Access	Synchronous writes, synchronous reads, write and read clock can be asynchronous to each other
FIFO	Built-in FIFO controller with dedicated pointer and flag circuitry

The BRAM72k ports are Illustrated in the following figure:

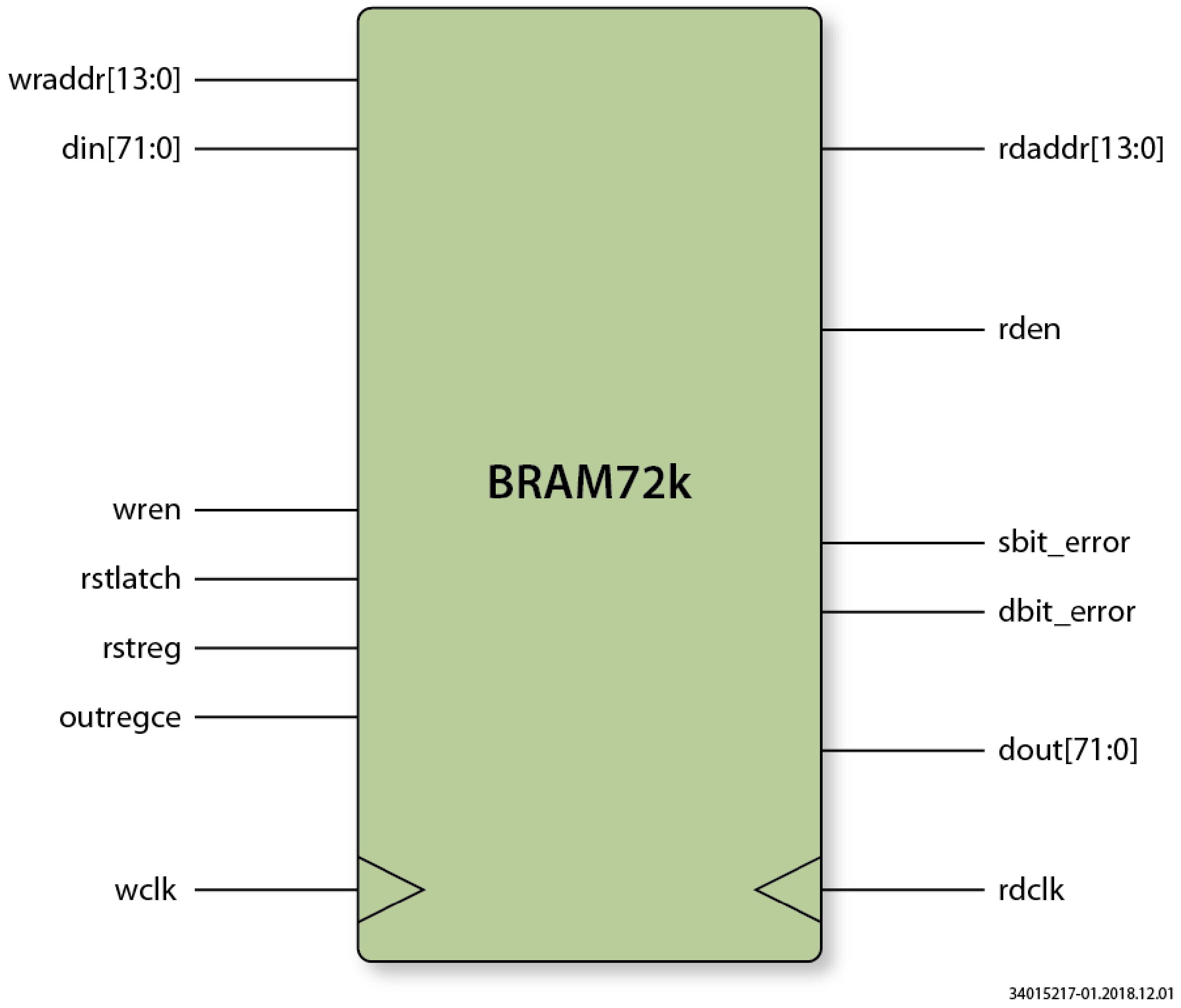


Figure 12: ***BRAM72k Block Diagram***

Organization

The organization (see table above) of each BRAM72k port can be independently configured.

Operation

The read and write operations are both synchronous. For higher performance operation, an additional output register can be enabled, which will add an additional cycle of read latency. The initial value of the memory contents may be specified by the user from either parameters or a memory initialization file. The initial/reset values of the output registers are set to zero. Byte enables are supported for the 72-bit width configuration of the BRAM72k. For other width configurations the full word is written when `wren` is asserted.

Note

Error correction is not available when using byte enables.

Memory operations may be performed simultaneously from both sides of the memory; however, there are restrictions. A memory collision is defined as the condition where both of the ports access the same memory location(s) within the same clock cycle (both ports connected to the same clock), or within a fixed time window (if each port is connected to a different clock). If one of the ports is writing an address while the other port is reading the same address, the write operation takes precedence, but the read data is invalid. The user may reliably read the data the next cycle if there is no longer a write collision.

Built in FIFO Controller

The ACE macro cell BRAM72kFIFO implements a 72 kb FIFO memory block utilizing the embedded BRAM72k blocks with dedicated pointer and flag circuitry. The BRAM72kFIFO can be configured to support a variety of widths and depths, ranging from 1024-entry depth with 72-bit data down to 16k depth with 4-bit data. The read and write clocks may be either synchronous or asynchronous with respect to each other. The BRAM72kFIFO controller supports both standard and First-Word-Fall-Through (FWFT) models. It includes `rd_err`, `wr_err`, `full` and `empty` flags, along with programmable `almost_full` and `almost_empty` flags. If the user read and write clocks are the same clock, the user may set the sync_mode to 1'b1 to enable faster and synchronous generation of the status flags and FIFO pointer outputs.

Error Correction

Error correction is available only when the BRAM72k is configured as 72-bit for both read and write. The built-in error correction logic provides single-bit error correction and dual-bit error detection on a user's 64-bit data bus. If the internal ECC logic is not used, all 72 bits can be used for other purposes such as tagging and various control functions. There are four modes of error correction:

- Normal ECC encode/decode mode – User reads/writes 64-bit data, automatic single-bit error correction and dual-bit error detection.
- ECC encoder and decoder disabled – ECC logic is disabled, no error correction/detection.
- ECC decode-only mode – User writes 72-bit data directly (64-bit data plus 8-bit error correction syndrome in bits [71:64]); user reads 64-bit data with automatic single-bit error correction and dual-bit error detection.
- ECC encode-only mode – User writes 64-bit data, automatically adds 8-bit error correction syndrome in bits [71:64]; user reads 72-bit data directly with no data correction/detection.

Initialization and Reset

Initial content of the BRAM72k can be optionally loaded during device configuration if specified by the user. Otherwise, the BRAM72k initial content is undefined. On reset, the RAM contents are unchanged, but the output register, if used, is set to zero when `rstreg` is asserted.

The initial state of the RAM read output is also set to zero on reset (asserting `rstlatch`).

Note

Access from opposite ports is not required to have the same organization; however, the number of total memory bits on each port must be the same.

Speedcore Gen4 Logic RAM

Logic RAM 2k

The LRAM2k implements a 2,304-bit memory block configured as a 32 × 72 simple dual-port (one write port, one read port) RAM. The LRAM2k has a synchronous write port. The read port is configured for asynchronous read operations with an optional output register. This memory block is distributed in the eFPGA fabric. A summary of LRAM2k features is shown in the table below.

Table 4: ***LRAM2k Key Features***

Feature	Value
Logic RAM size	2,304 bits
Organization	32 × 72, 64 × 36, 16 × 144(†) (depth × width)
Physical Implementation	Columns throughout device
Number of Ports	Simple dual port (one read, one write)
Port access	Synchronous writes, combinatorial reads
FIFO	Built-in FIFO controller with dedicated pointer and flag circuitry

Note

† 16 × 144 only available as internal path when tightly coupled with MLP.

The LRAM2k ports are shown in the following figure:

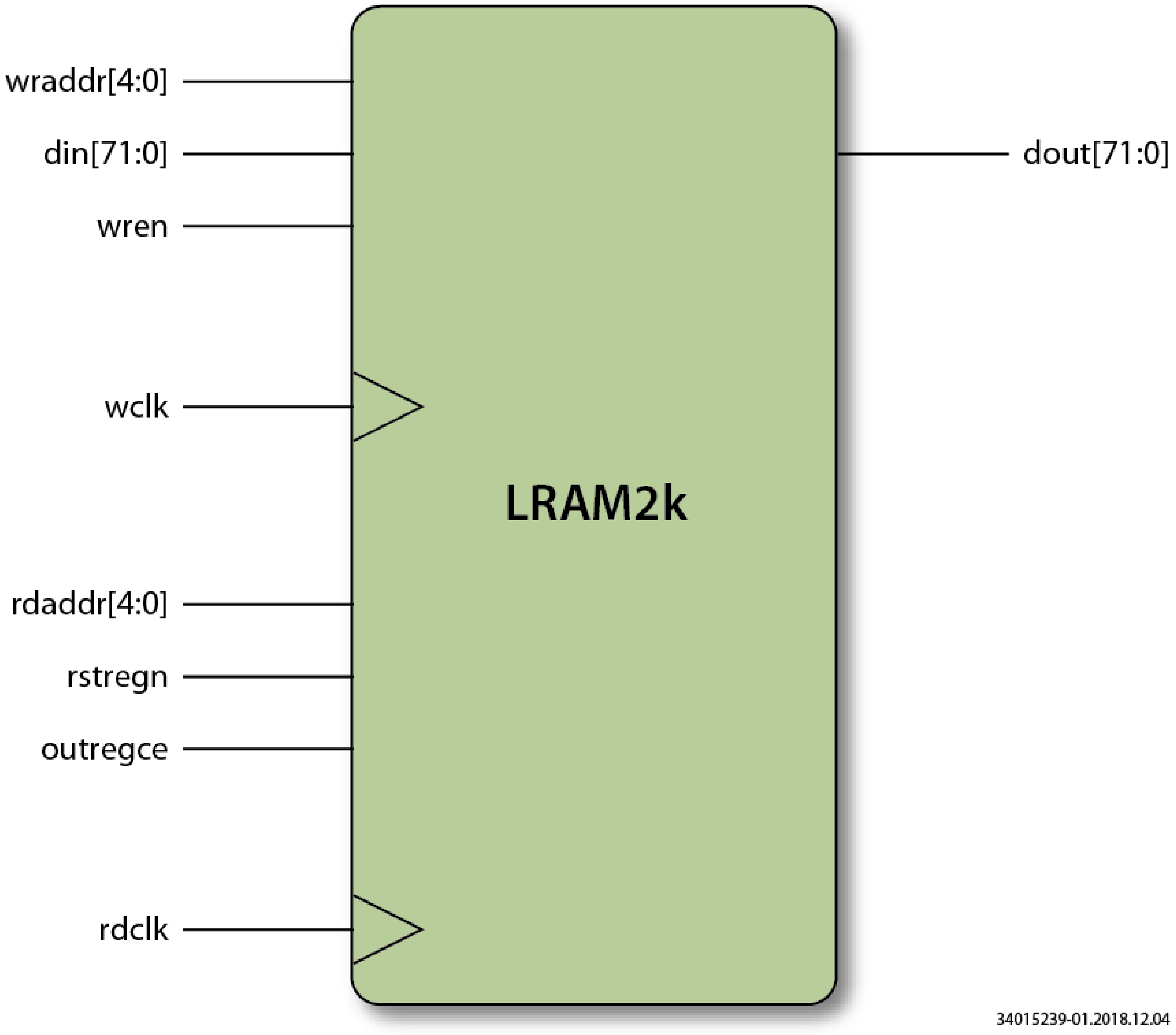


Figure 13: ***LRAM2k Block Diagram***

Organization

The LRAM2k is configured as a 32 × 72 simple dual-port (one write port, one read port) RAM.

Operation

The LRAM2k has a synchronous write port. The read port is configured for asynchronous read operations, making read data available the same cycle `rden` is asserted. There is an optional output register, that if enabled, adds a cycle of latency to the read. There are no per-byte write enables, the entire word is written on each cycle that `wren` is asserted.

Memory operations may be performed simultaneously from both sides of the memory; however, there are restrictions. A memory collision is defined as the condition where both of the ports access the same memory location(s) within the same clock cycle (both ports connected to the same clock), or within a fixed time window (if each port is connected to a different clock). If one of the ports is writing an address while the other port is reading the same address, the write operation will take effect, and the read data is invalid. The user may reliably read the data the next cycle if there is no longer a write collision.

Built in FIFO Controller

Implements a 2 kb FIFO memory block utilizing the embedded LRAM2k blocks with dedicated pointer and flag circuitry. The read and write clocks may be either synchronous or asynchronous with respect to each other. A single clock for read and write provides for lower latency. The LRAM2k block also supports shift register mode where the FIFO implements a fixed delay of 72-bit words from 1 to the depth of the memory block.

Initialization

By default, the contents of the LRAM2k memory are undefined. Optionally, the user may initialize memory contents from either parameters or a memory initialization file. If the output register is enabled, it will be set to zero on reset.

Logic RAM 4k

The LRAM4k implements a 4,096-bit memory block configured as a 128 × 32 simple dual-port (one write port, one read port) RAM. The LRAM4k has a synchronous write port. The read port is configured for asynchronous read operations with an optional output register. This memory block is distributed in the eFPGA fabric. A summary of LRAM4k features is shown in the table below.

Table 5: ***LRAM4k Key Features***

Feature	Value
Logic RAM size	4,096 bits
Organization	128 × 32
Physical implementation	Dedicated columns
Number of ports	Simple dual port (one read, one write)
Port access	Synchronous writes, asynchronous reads

The LRAM4k ports are shown in the following figure:

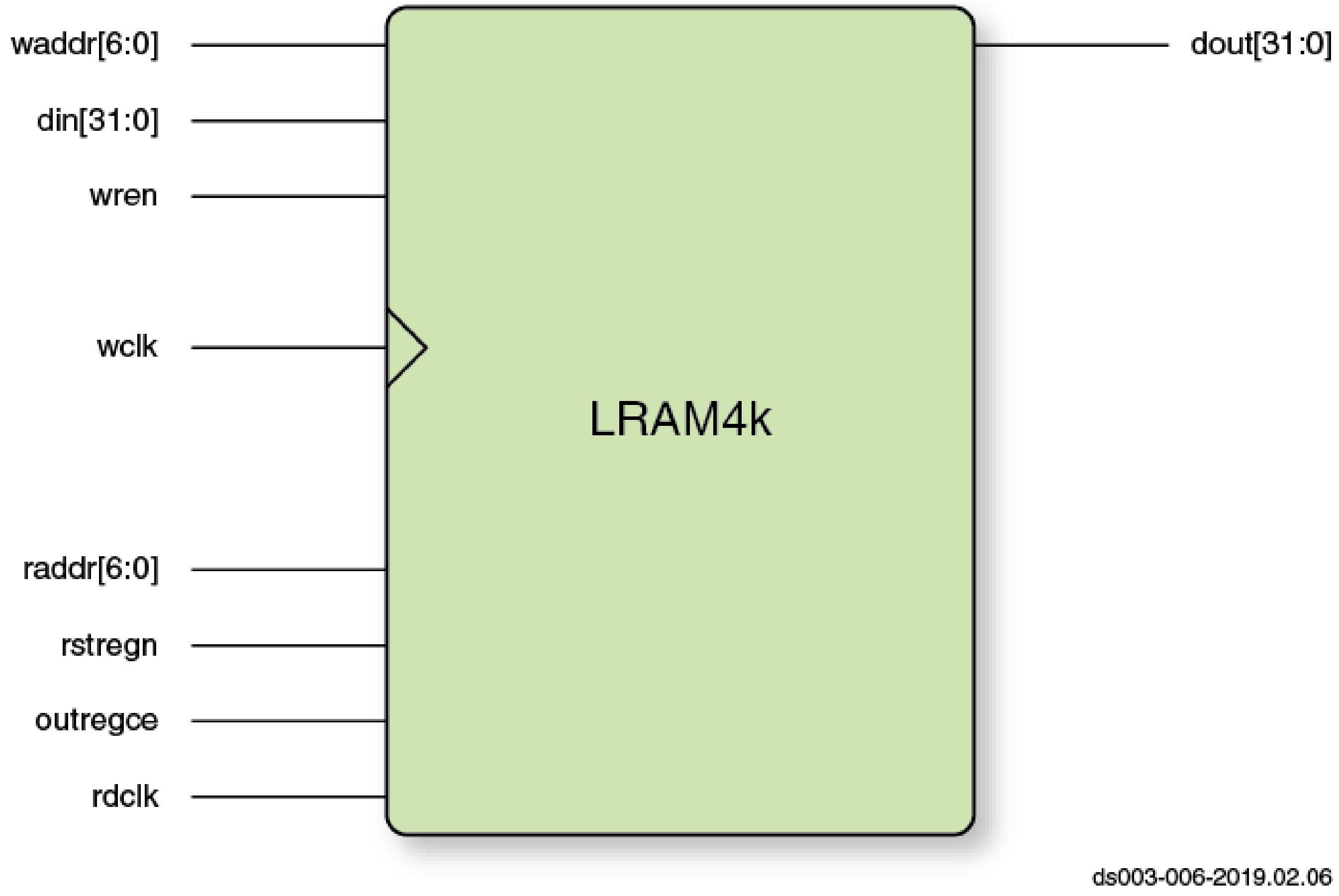


Figure 14: ***LRAM4k Ports***

Organization

The LRAM4k is configured as a 128 × 32 simple dual-port (one write port, one read port) RAM.

Operation

The LRAM4k has a synchronous write port. The read port is configured for asynchronous read operations with an optional output register. The memory is organized as little-endian order with bit 0 mapped to bit 0 of parameter `mem_init_00` and bit 4095 mapped to bit 255 of parameter `mem_init_15`.

Initialization and Reset

By default, the contents of the LRAM4k memory are undefined. If the user wants the initial contents to be defined, he may assign them from either a file pointed to by the `mem_init_file` parameter or assign them from the value of the `mem_init` parameter.

Speedcore Gen4 DSP64 Block

The DSP64 blocks include multiple/accumulate and associated logic to efficiently implement math functions such as finite impulse response (FIR) filters, fast Fourier transforms (FFT), and infinite impulse response (IIR) filters. The DSP64 blocks are optimized to operate with the logic fabric and LRAM blocks to implement math functions. Refer to the *Speedcore IP Component Library User Guide* for more details.

The DSP64 blocks have the following functions:

- 27-bit preadder
- 18 × 27 multiplication/accumulation with programmable load value
- Add/subtract
- Saturating add/subtract support
- $(A \pm B)^2$ and $(A \pm B)^2$ + constant
- Output rounding

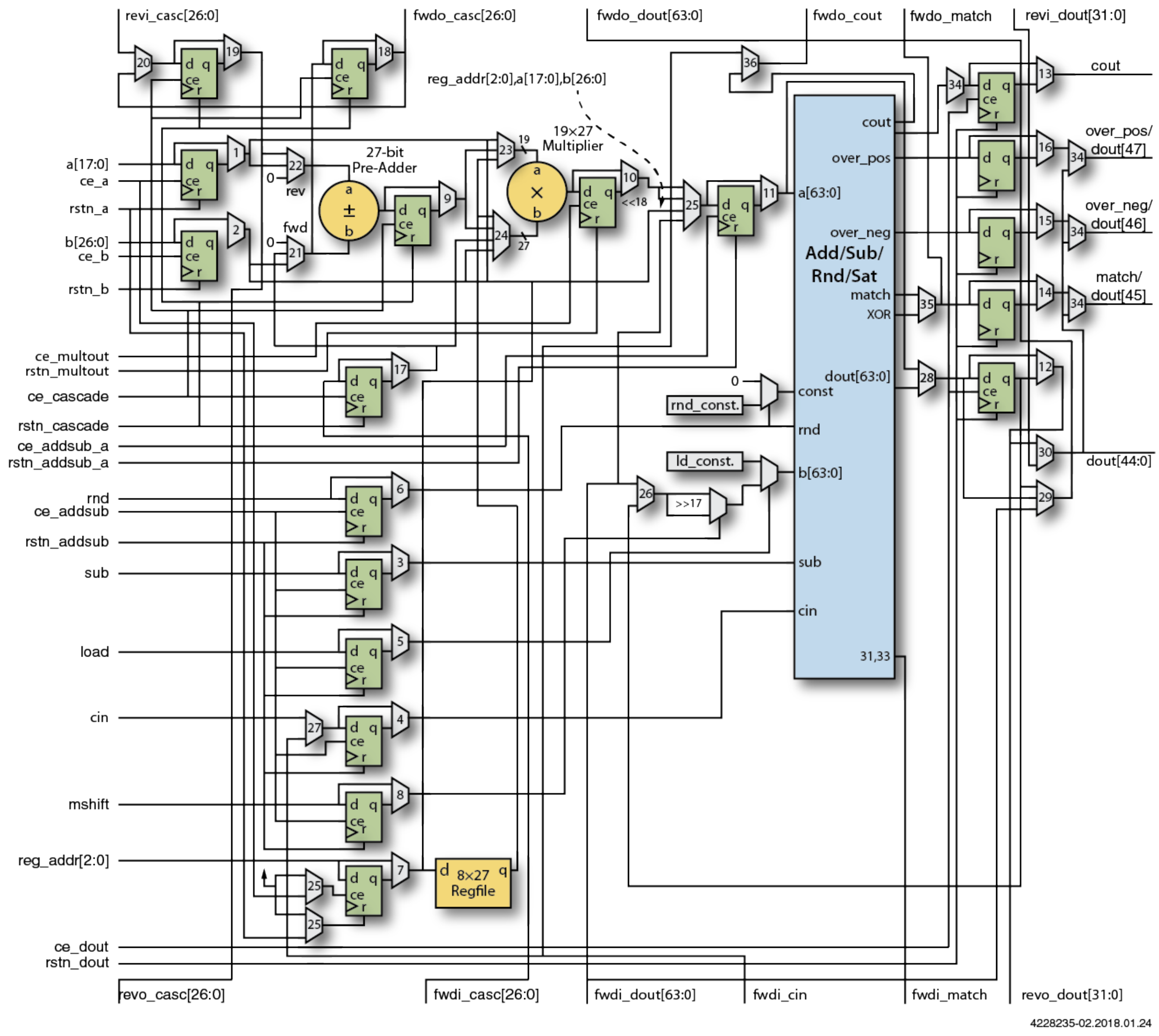


Figure 15: *DSP64 Block*

Speedcore Gen4 MLP Block

The machine learning processing block (MLP) consists of an array of up to 12 multipliers, followed by an adder tree, an accumulator, and a rounding/saturation/normalize block. The number of multipliers available varies with the bit width of the operands. The MLP offers a range of features including integer multiply with optional accumulate, bfloat16 operations, floating point 16, block floating point, and floating point 24. Additionally, when the MLP is placed next to a BRAM or LRAM tile, the number of data inputs to the MLP block doubles. This configuration allows twice the number of multipliers to be used within the MLP block. Below is a list of features available with the MLP block.

- Configurable multiply precision and multiplier count
- Multiple number formats (fixed and floating point)
- Multiple rounding and saturation features

Below is a table detailing the number of multiplies based on data type.

Table 6: *MLP Multiply By Data Type*

Data Type	No. of Multiplies with Data Inputs from Fabric Only	No. of Multiplies with Data Inputs from Fabric and BRAM/LRAM
3-bit integer	12	24
4-bit integer	8	16
6-bit integer	6	12
8-bit integer	4	8
16-bit integer	1	–
Bfloat16	1	1
Floating point 16	1	1
3-bit Block floating point (mantissa)	12	24
4-bit Block floating point (mantissa)	8	16
6-bit Block floating point (mantissa)	6	12
8-bit Block floating point (mantissa)	4	8
16-bit Block floating point (mantissa)	1	–
Floating point 24 (8-bit exponent) Rounding on mantissa LSB only		–

Along with the above multiply configurations, the MLP block includes optional input registers and optional pipelining registers at various locations to improve high frequency designs. There is a deep adder tree after the multipliers, with the option to bypass the adders and output the multiplier products directly. There is a normalization block for floating point and block floating point operations. In addition, a feedback path allows for accumulation within the MLP block. A cascade path allows for the adder tree to extend across multiple MLP blocks in a column without using extra fabric resources.

Below is a figure showing an example of eight inputs from the fabric of eight bits each, creating four multipliers in the MLP.

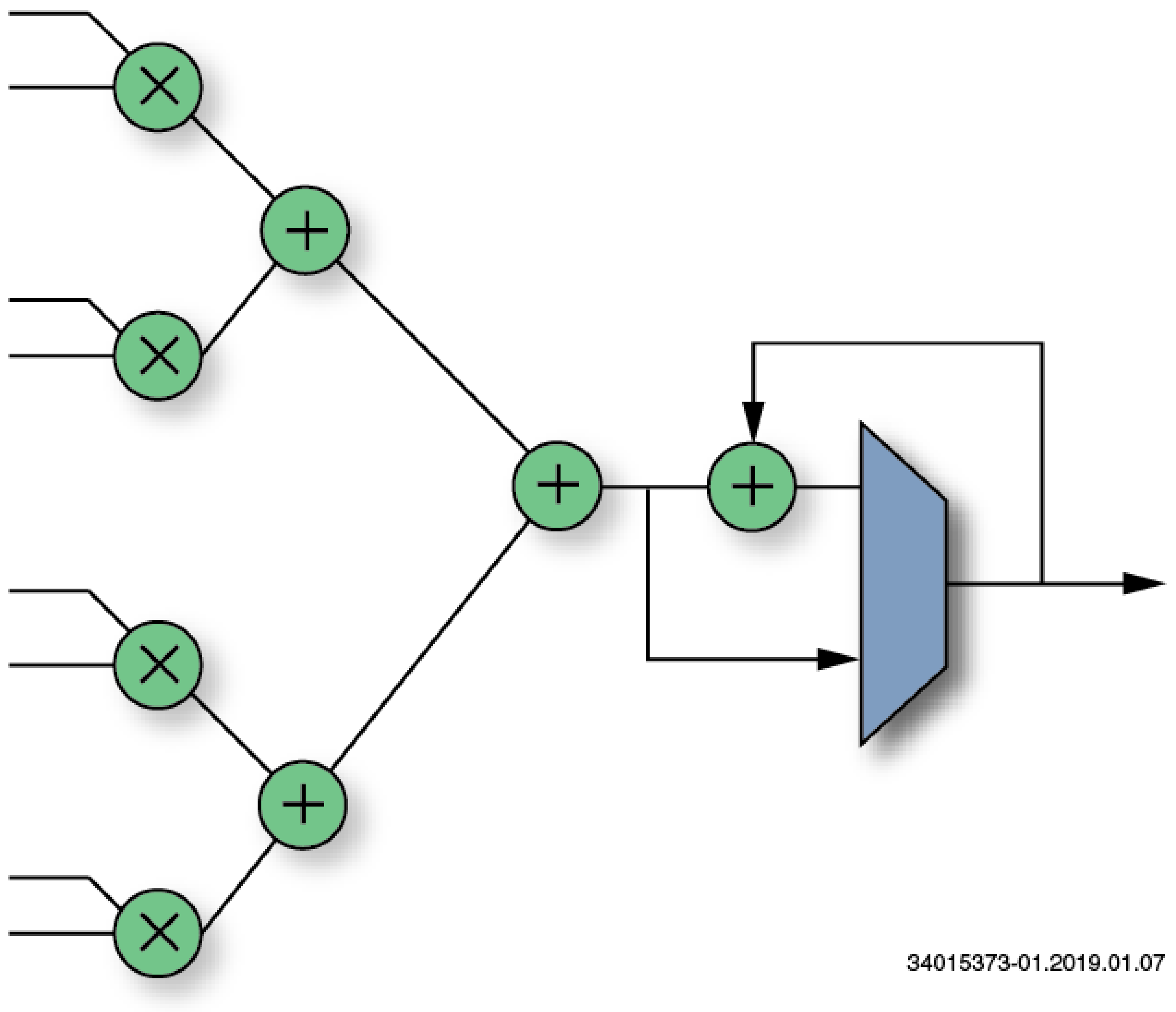


Figure 16: ***MLP Multipliers with Fabric-only Inputs***

MLP Combined with Memory Blocks

In order to maximize the connectivity to the MLP block, BRAM or LRAM columns can be placed adjacent to the MLP column. This placement doubles the number of allowed multipliers within an MLP block and provides the following features with the MLP block:

- Cyclical register file, similar to a cache, that can double the compute performance (data is saved for efficient reuse)
- Column bonding and MLP cascade paths creates hardened paths between memory blocks and MLP blocks which enables high-performance functionality while freeing up general-purpose routing

Below is a figure illustrating the number of multipliers with eight inputs from the fabric and eight inputs from a BRAM, each with 8-bit inputs, delivering eight multiplies.

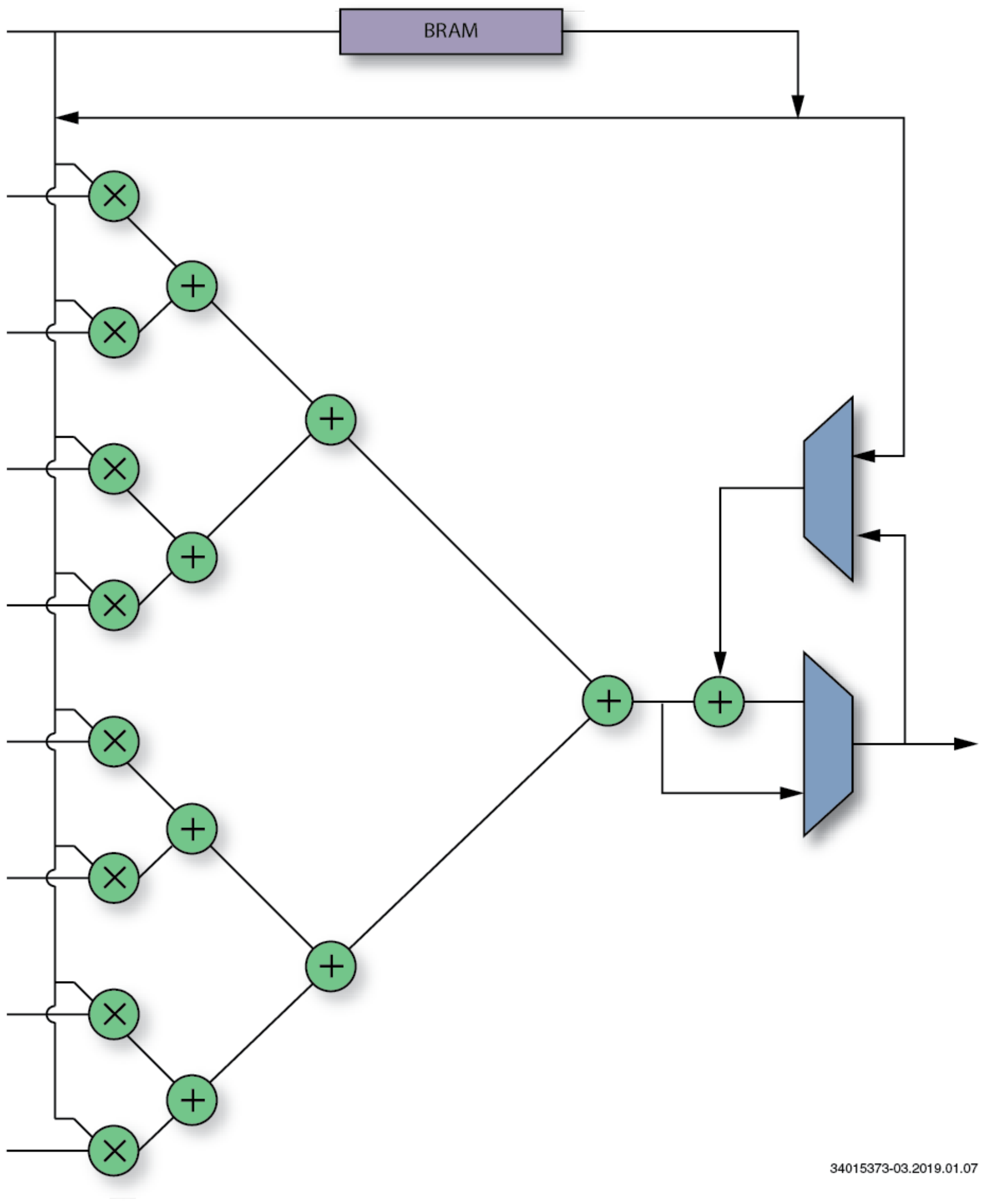


Figure 17: ***MLP Multipliers with Fabric and BRAM Inputs***

Additionally, an LRAM adjacent to the MLP block can be used as a cache, with the results of the multiplies written back to the LRAM and the results reused as inputs to the multipliers. The figure below illustrates the use of the LRAM with the MLP block, showing eight inputs from the fabric, eight inputs from the LRAM (each with 8-bit data inputs), and the result of the MLP being written back to the LRAM.

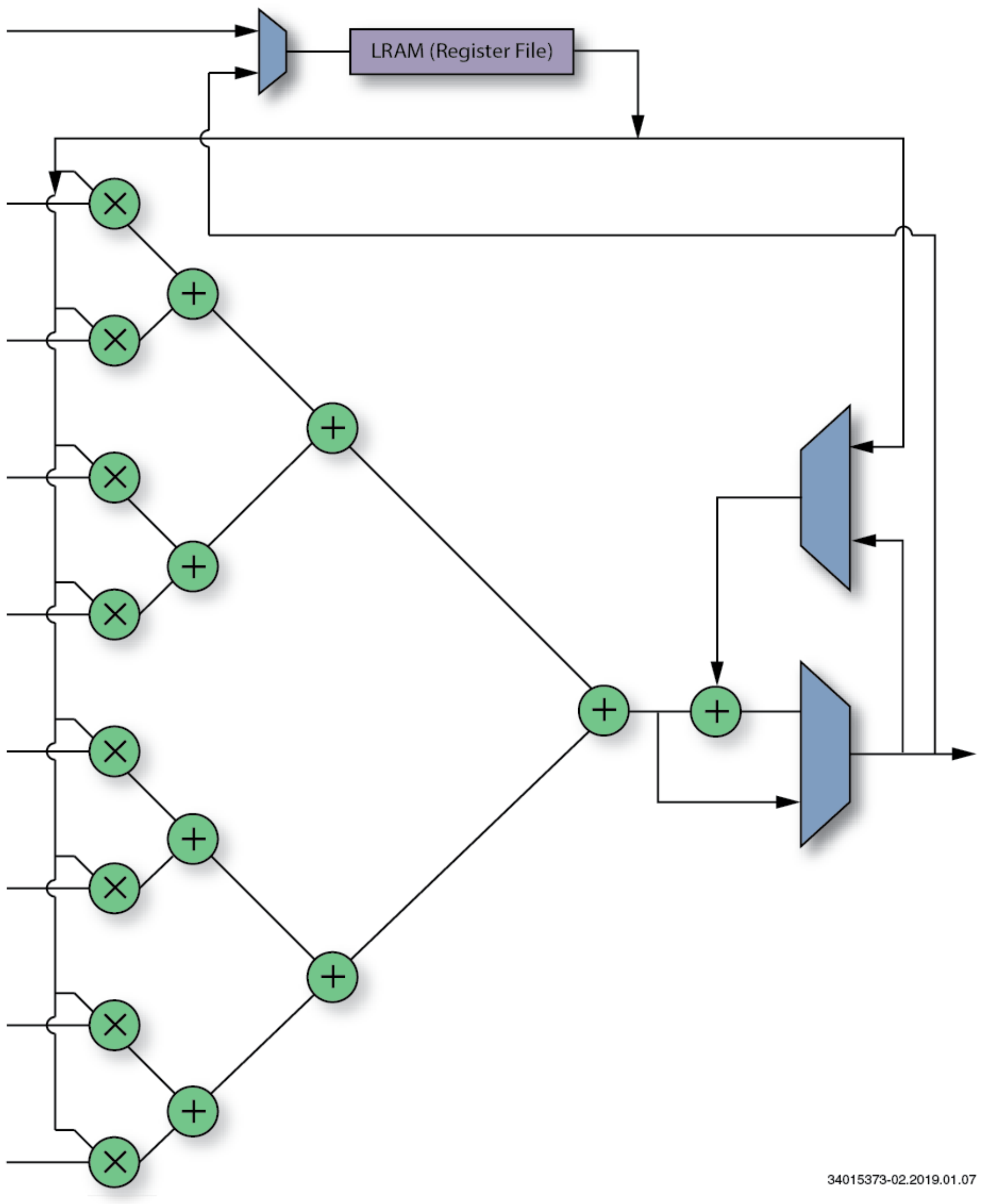


Figure 18: ***MLP Multipliers with LRAM Inputs and Write-Back***

Bus Routing

New in Speedcore Gen4 architecture is dedicated bus routing. In addition to traditional per-bit FPGA routing, the Speedcore Gen4 architecture includes separate dedicated bus-based routing for high performance datapaths. These buses are placed into groups of up to 8 bits wide and are routed independently from standard routing in order to significantly reduce congestion.

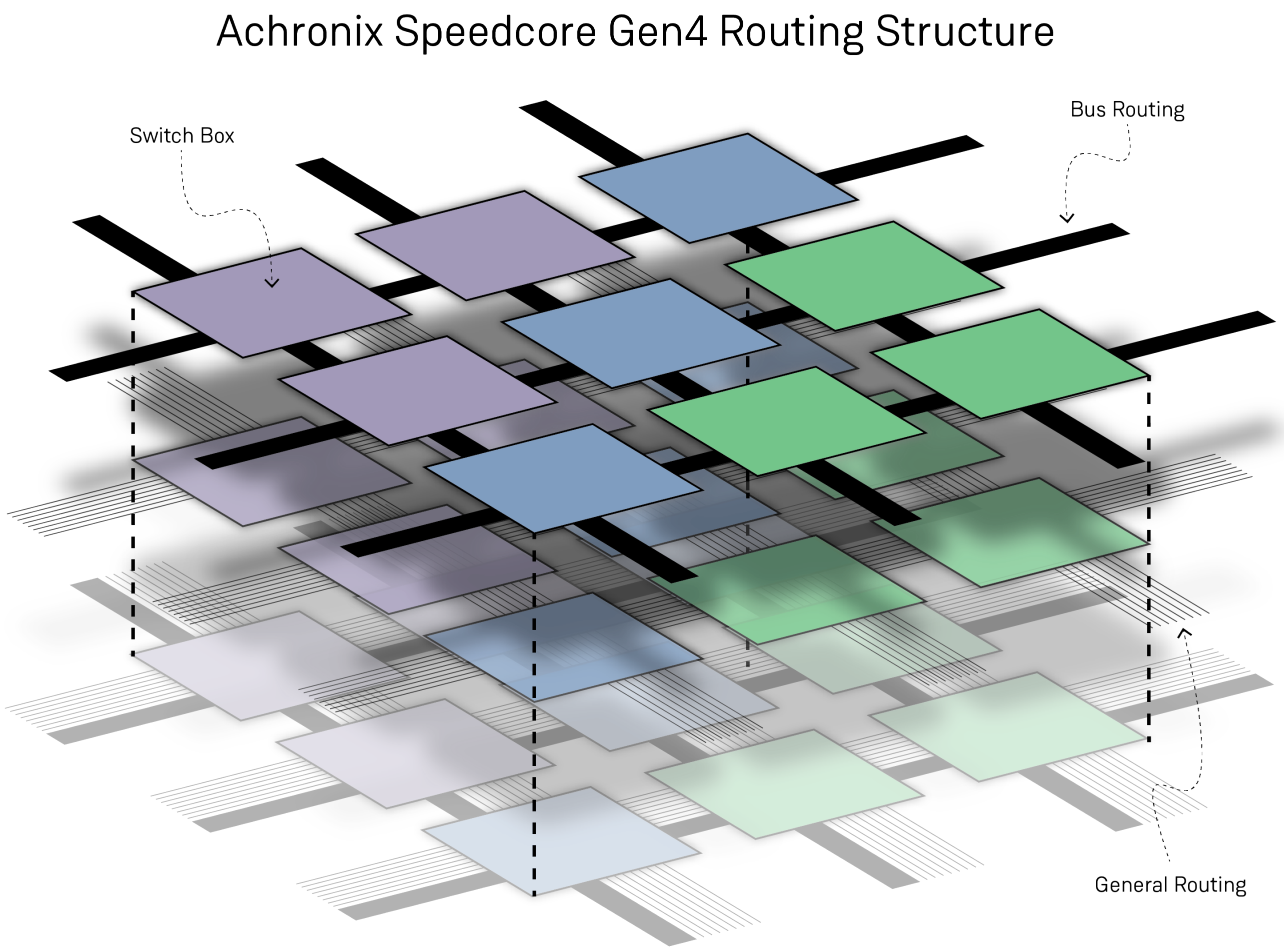


Figure 19: *Speedcore Gen4 Bus Routing*

Additionally, the Speedcore Gen4 architecture introduces a programmable switch network for bus routing. There is a 4 × 1 bus MUX for each of the 8-bit buses inside each Speedcore switchbox. These bus MUXes are cascadable for wider MUX requirements. This added MUXing reduces overall logic and routing resources for a design, leading to improved performance and smaller area. The figure below shows an example of how to route logic that selects between two 8-bit buses. This logic might look something like:

end_FF[7:0] = select ? FF0[7:0] : FF1[7:0]

The top portion shows the 8-bit buses routing on the programmable switch network. The bottom portion shows how the same logic routes using standard fabric routing. The red sections show the active run-time logic, and the blue sections are fixed after configuration. With the MUXing occurring inside the programmable switch network, and the full 8-bit bus routing, the number of routes is drastically reduced. The programmable switch network includes MUX logic inside the switchbox, whereas the standard fabric routing requires routing to extra logic for the MUXing function.

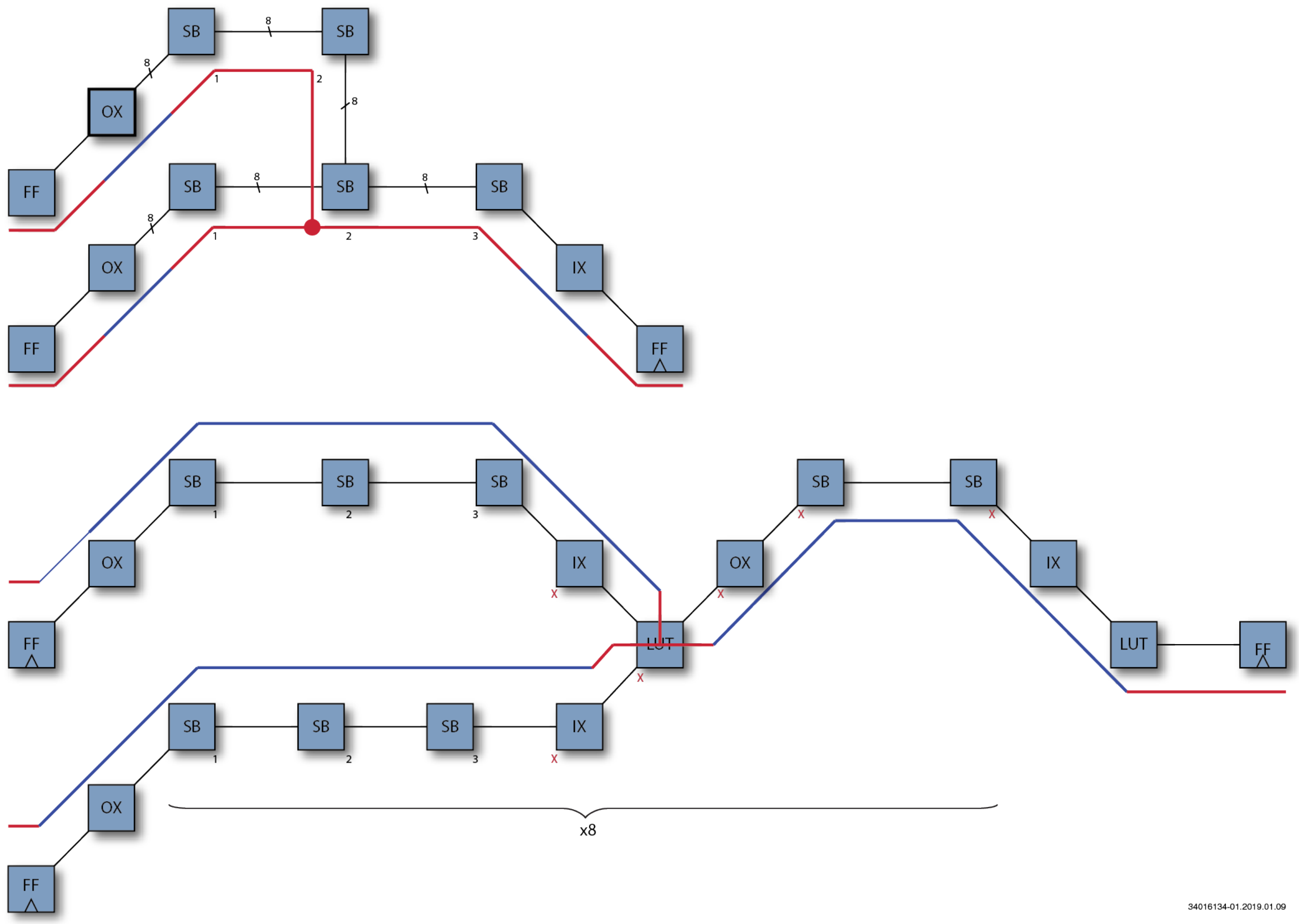


Figure 20: ***Bus Routing with Programmable Switch Network vs. Standard Routing***

Chapter - 3: Speedcore Gen4 IP Interface

Interfaces

There are three sets of interfaces to the Speedcore™ block (see the figure below).

Data Signals

Data signals (inputs and outputs) can be on all four sides or only on two opposite sides. At the boundary, there is an option to either register the signals or send the signals directly to the programmable logic core.

Clock Inputs

Clock inputs follow the same pattern as data. These can be on all four sides or on two opposite sides. There are 16 interface clocks per cluster, per side.

Programming Interface

There is a dedicated set of signals for programming of eFPGA block. The number of these signals depends on the programming options selected.

The table following the figure lists the interface signals of the eFPGA block.

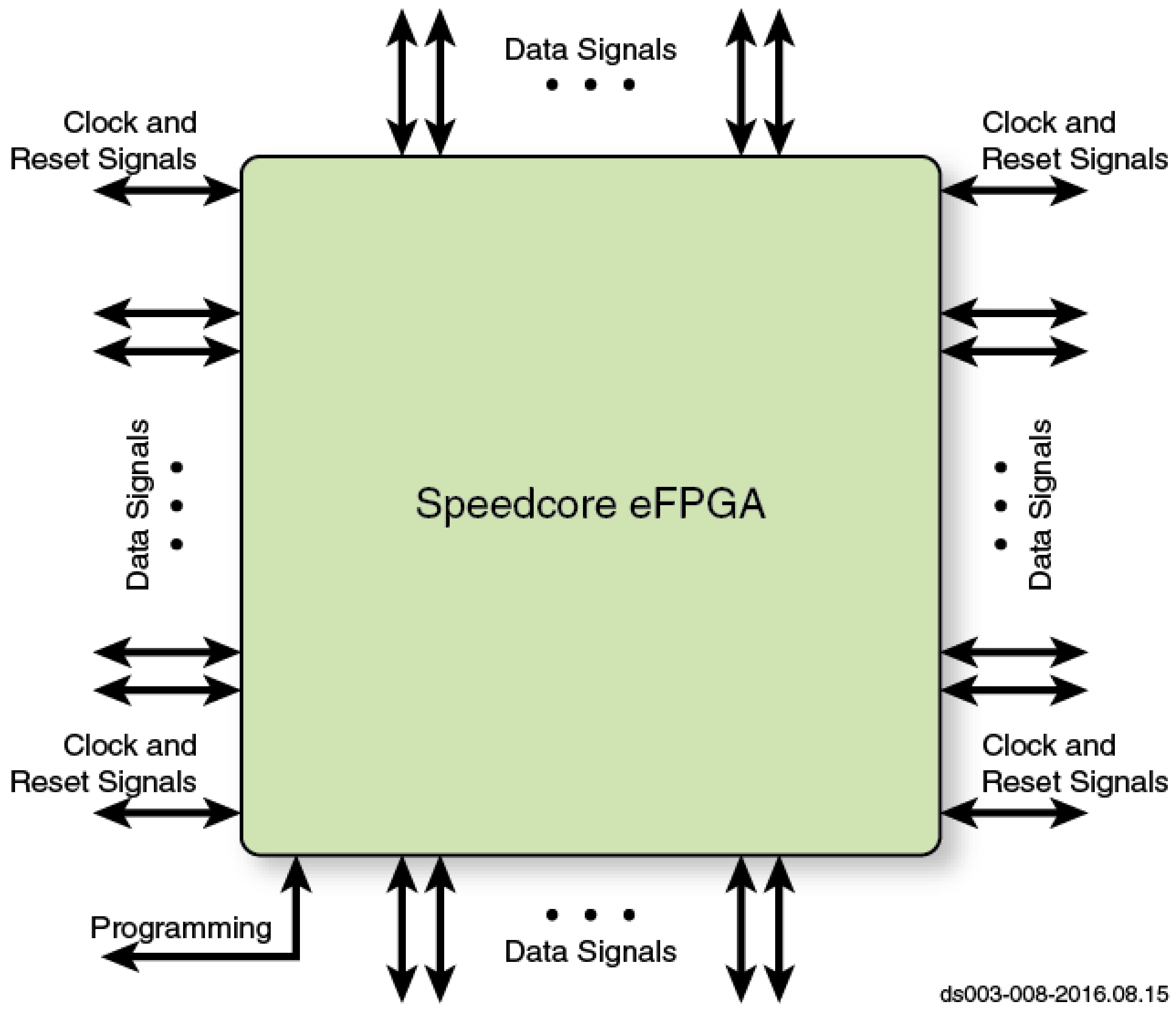


Figure 21: ***Speedcore eFPGA Interfaces***

Pins

The following table describes the input/output pins of the Speedcore eFPGA core:

Table 7: ***Speedcore eFPGA Pins***

Pin Name	Direction	Description
i_data_w/e/n /s[n:0]	Input	Data inputs to the programmable core. The bit width depends on size and customer requirements.
o_data_w/e /n/s[m:0]	Output	Data outputs from the programmable core. The bit width depends on size and customer requirements.
i_clock[c:0]	Input	Clock inputs to the programmable core.

Pin Name	Direction	Description
o_clock[d:0]	Output	Clock outputs from the programmable core.
i_config[x:0]	Input	Bitstream data, control and configuration setting selection pins for Speedcore. The width of these signals depends on the selected programming option.
o_config[y:0]	Output	Status output and signaling pins for Speedcore.

High-Speed AXI

Although a Speedcore Gen4 eFPGA instance can support a large number of individual I/O, many applications require standard interfaces in order to help construct an ASIC using existing IP, buses and host protocols. To support these requirements, Achronix offers a high-speed, advanced extensible interface (AXI) which can connect to a customer's internal AXI structure, running at the frequencies of the host ASIC system. This AXI then de-multiplexes to multiple AXIs within the Speedcore eFPGA instance, each AXI running at the frequencies of the application or accelerator which is created within the Speedcore eFPGA instance.

Note

In addition to AXI, other industry-standard bus protocols can also be supported, please contact the factory for details.

Chapter - 4: Speedcore Gen4 In-System Debug

Snapshot is the real-time design debugging tool for Achronix FPGAs and cores. The Snapshot debugger, which is embedded in the ACE software, delivers a practical platform to observe the signals of a user's design in real-time. To use the Snapshot debugger, the Snapshot macro needs to be instantiated inside the user's RTL. After instantiating the macro and programming the device, the user will be able to debug the design through the Snapshot Debugger GUI within ACE, or via the `run_snapshot` TCL command API.

The Snapshot macro can be connected to any logic signal mapped to the Achronix core, to monitor and potentially trigger on that signal. Monitored signal data is collected in real time in regular BRAMs, prior to being transferred to the ACE Snapshot GUI. The Snapshot macro has configurable monitor width and depth, as well as other configuration parameters, to allow user control over resource usage. The ACE Snapshot GUI interacts with the hardware via the JTAG interface: interactively specified trigger conditions are transferred to the design, and collected monitor data is transferred back to the GUI, which displays the data using a builtin waveform viewer.

The figure below shows the components involved in a Snapshot debug session.

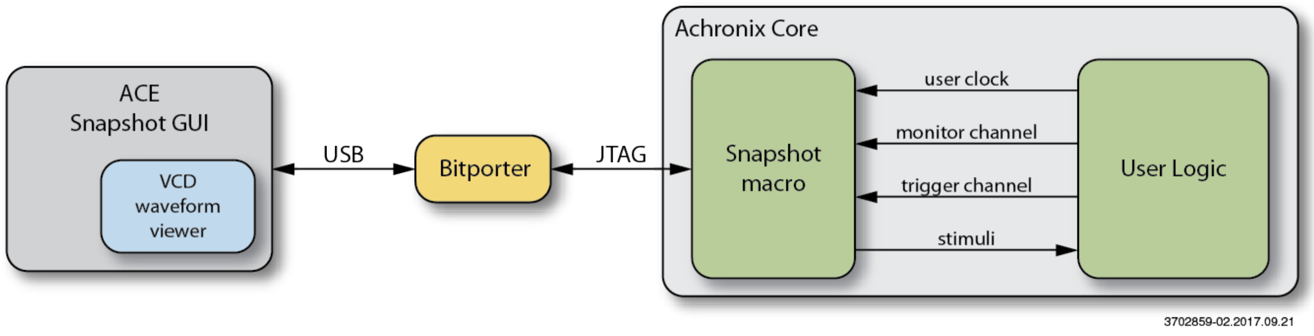


Figure 22: *Snapshot Overview*

Features

The Snapshot macro samples user signals in real time, storing the captured data in one or more BRAMs. The captured data is then communicated through the JTAG interface to the ACE Snapshot GUI.

The implementation supports the following features:

- Monitor channel capture width of 1 to 4064 bits of data.
- Monitor channel capture depth of 512 to 16384 samples of data at the user clock frequency.
- Trigger channel width of 1 to 40 bits.
- Supports up to three separate sequential trigger conditions. Each trigger condition allows for the selection of a subset of the trigger channel, with AND or OR functionality.
- Bit-wise support for edge- (rise/fall) or level-sensitive triggers.
- The ACE Snapshot GUI allows specification of trigger conditions and circuit stimuli at runtime.
- An optional initial trigger condition, specified in RTL parameters, to allow capture of data immediately after startup, before interaction with the ACE Snapshot GUI.

- A stimuli interface, 0 to 512 bits wide, that allows the user to drive values into the Achronix core logic from Snapshot. Stimuli values are specified with the ACE Snapshot GUI and made available before data capture.
- Optionally, the data capture can include values before the trigger occurred. This "pre-store" amount can be specified in increments of 25% of the depth.
- Captured data is saved in a standard VCD waveform file. The ACE Snapshot GUI includes a waveform viewer for immediate feedback.
- The VCD waveform file includes a timestamp for when the Snapshot was taken.
- ACE automatically extracts the names of the monitored signals from the netlist, for easy interpretation of the waveform.
- A repetitive trigger mode, in which repeated Snapshots are taken and collected in the same VCD file.
- The JTAG interface can be shared with the user design.
- A TCL batch/script mode interface is provided via the `run_snapshot` TCL command

Chapter - 5: Speedcore Gen4 Integration Flow

Physical Integration with Customer ASICs

The Speedcore™ eFPGA is provided as a fixed-transistor-layout building block that integrates with industry-standard ASIC flows such as Synopsys Design Compiler and IC Compiler. The following collateral will be provided:

- Verilog definition of logical connectivity at boundary
- Liberty timing library for timing closure at the boundary
- LEF defining the physical floorplan, pins, and metal blockages
- GDS/Oasis physical database

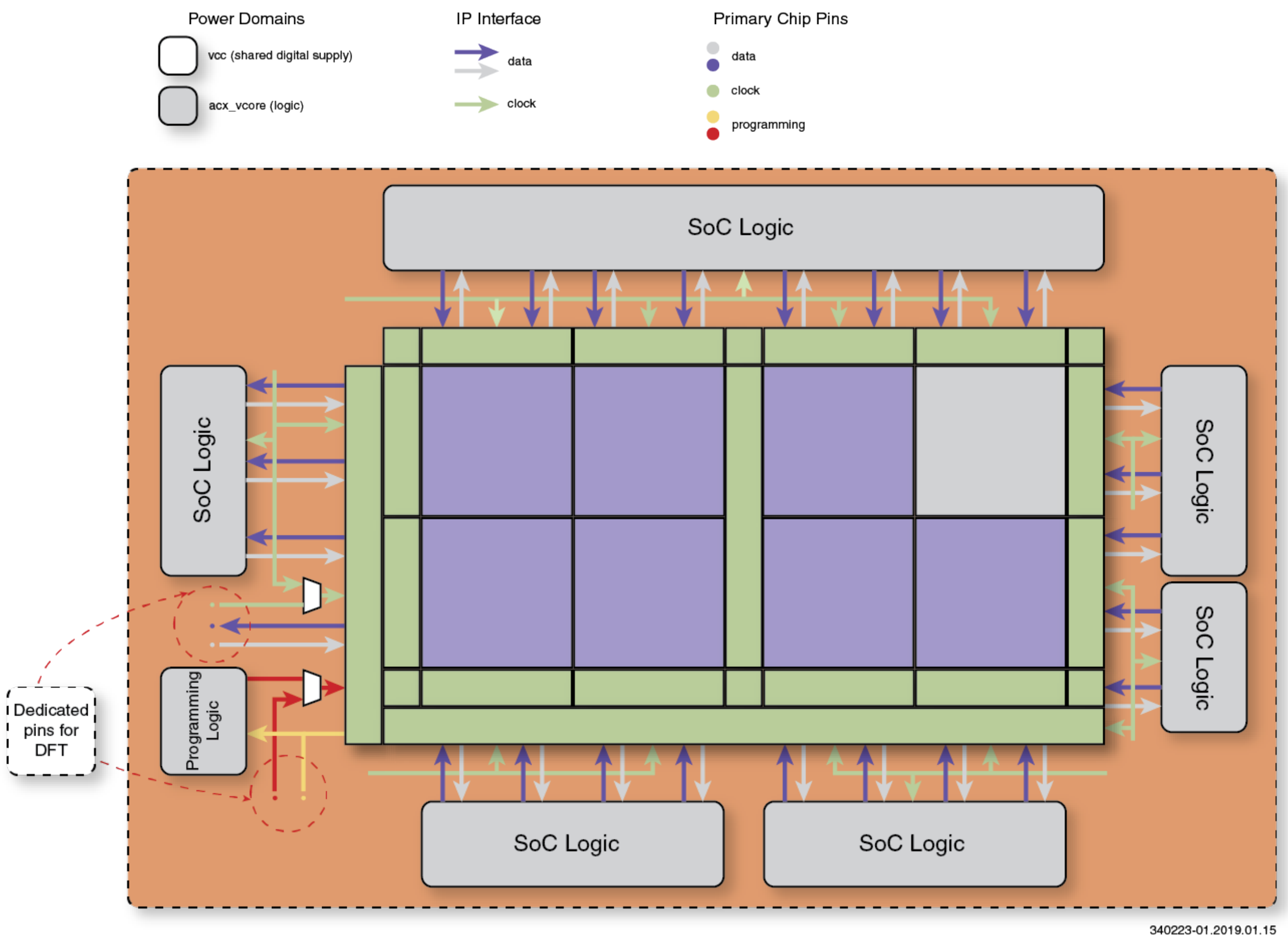


Figure 23: ***Sample eFGPA Instantiation***

The data inputs/outputs and clock inputs can come from the ASIC logic or can come directly from the package pins (balls) of the ASIC. The programming interface must have access to the package pins of the ASIC to enable Speedcore programming. In addition, a certain number of Simulation and Validation data inputs/outputs must be accessible through the package pins for eFPGA IP standalone testing. Details on the number of pins and connectivity will be provided in the *Design and Integration Manual*.

Simulation and Validation

The Speedcore eFPGA will be supplied with ACE (Achronix CAD Environment) software that provides a complete solution for simulating, synthesizing, mapping, and timing any user logic in the eFPGA fabric. The behavioral models or gate-level netlists representing the logic mapped inside the FPGA can then be directly integrated into the user's simulation/verification flow. In addition, SDF-annotated simulation models and standard Liberty timing models of the user logic can be emitted and integrated into the user's system-level timing validation flow.

Chapter - 6: Speedcore eFPGA Device Specifications

Device Resource Counts

The table below lists the resource counts for Speedcore AC16tSC04HI03A.

Table 8: ***Resource Counts for AC16tSC04HI03A***

Resource	Details	Count	Amount
Port_connections		9,000	–
LUTs	6LUT	198,720	–
BRAMs	20k TDP	1,080	21.09 Mb
LRAMs	4k SDP	0	0.00 Mb
Math	DSP64	180	–

Key Metrics and Supported Operation

The table below provides information on the performance, power supply requirements and supported operating temperatures for Speedcore AC16tSC04HI03A.

Table 9: ***Key Metrics and Supported Operation for AC16tSC04HI03A***

Metric	Description
Operating Frequency	Up to 500MHz.
Power Supplies	This device has two power supplies, $\mathbf{V_{DDL}}$ and $\mathbf{V_{DD}}$, which are both set to either 0.7V or 0.8V. These supplies are shared on-die and need to be powered from a single regulator.
Temperature	Industrial temperature range of -40˚C to +125˚C.

Revision History

The following table lists the revision history of this document.

Version	Date	Description
1.0	07 Feb 2019	• Initial release.
1.1	29 Jul 2019	• First public release: removed confidential markings. • Speedcore Gen4 Architecture (see page 9): updated the key feature table for the LRAM2k.