
Speedcore Gen3 eFPGA Datasheet (DS003)

Speedcore eFPGA IP



Copyrights, Trademarks and Disclaimers

Copyright © 2019 Achronix Semiconductor Corporation. All rights reserved. Achronix, Speedcore, Speedster, and ACE are trademarks of Achronix Semiconductor Corporation in the U.S. and/or other countries. All other trademarks are the property of their respective owners. All specifications subject to change without notice.

NOTICE of DISCLAIMER: The information given in this document is believed to be accurate and reliable. However, Achronix Semiconductor Corporation does not give any representations or warranties as to the completeness or accuracy of such information and shall have no liability for the use of the information contained herein. Achronix Semiconductor Corporation reserves the right to make changes to this document and the information contained herein at any time and without notice. All Achronix trademarks, registered trademarks, disclaimers and patents are listed at <http://www.achronix.com/legal>.

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 4

 Feature Summary 4

 Functionality 4

 Process Technology 5

 Programming 5

Chapter - 2: Speedcore Architecture 6

 Block Floorplan 6

 Speedcore Clock Network 8

 Speedcore Interface Cluster 9

 Speedcore Logic Fabric – Reconfigurable Logic Block 11

 Routing Between RLBs 13

 Speedcore Memory Resources 15

 Block RAM 20k 15

 Logic RAM 4k 17

 Speedcore DSP64 Blocks 19

Chapter - 3: Speedcore IP Interface 21

 Interfaces 21

 Data Signals 21

 Clock Inputs 21

 Programming Interface 21

 Pins 22

Chapter - 4: Speedcore In-System Debug 24

 Features 24

Chapter - 5: Speedcore Integration Flow 26

 Physical Integration with Customer ASICs 26

 Simulation and Validation 27

Revision History 28

Chapter - 1: Overview

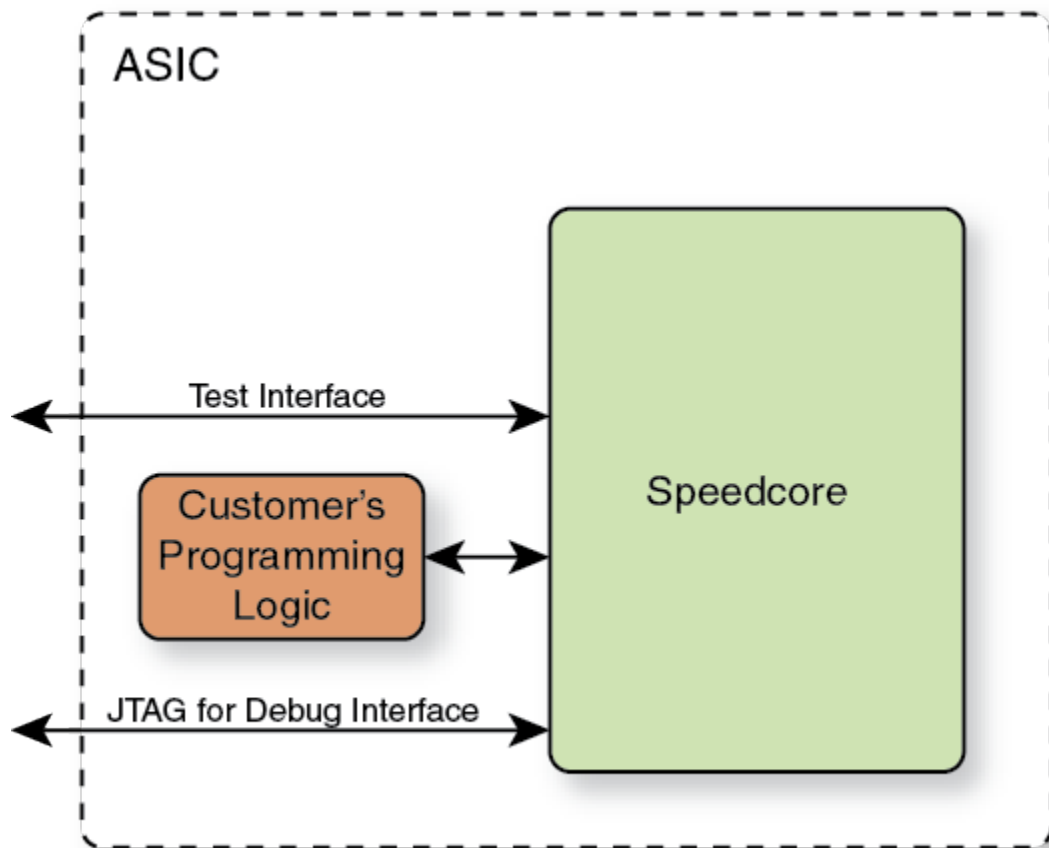
Introducing Speedcore eFPGA

Achronix's Speedcore™ embedded FPGA (eFPGA) architecture includes look-up-table, memory, and DSP building blocks that are designed in a modular structure which allows customers to define any quantity of resources required for their end system.

Achronix delivers the Speedcore IP in GDSII format for the customer to integrate into their ASIC and delivers the Achronix ACE design tools that are used to compile designs into the Speedcore eFPGA.

Feature Summary

Speedcore eFPGA is an embeddable IP with a pin-based interface to connect to. This interface provides a very large number and high density of input/output data and clock pins for high bandwidth and throughput. Speedcore IP does not include programmable I/O — it is designed to be completely surrounded by the end user ASIC (see the figure below).



ds003-001-2015.08.31

Figure 1: Embedded Speedcore

Functionality

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

- **Logic** – 4-input look-up-tables (LUTs) plus integrated wide mux functions and fast adders
- **Logic RAM** – up to 4 kb per memory block
- **Block RAM** – up to 20 kb per memory block
- **DSP64** – each block has a 18 × 27 multiplier, 64-bit accumulator and 27-bit pre-adder

There are design rules that dictate minimum and maximum relative quantities for each of the available resources. Customers submit their requirements to Achronix and receive back a Speedcore specification document that defines the exact resource count for their Speedcore eFPGA.

Process Technology

Speedcore eFPGAs can be built on any digital process technology. Achronix charges a process technology port fee if the customer needs the Speedcore IP on a process technology and metal stack not currently supported by Achronix. Please contact Achronix directly to obtain a list of process technologies and metal stacks that are currently supported.

Programming

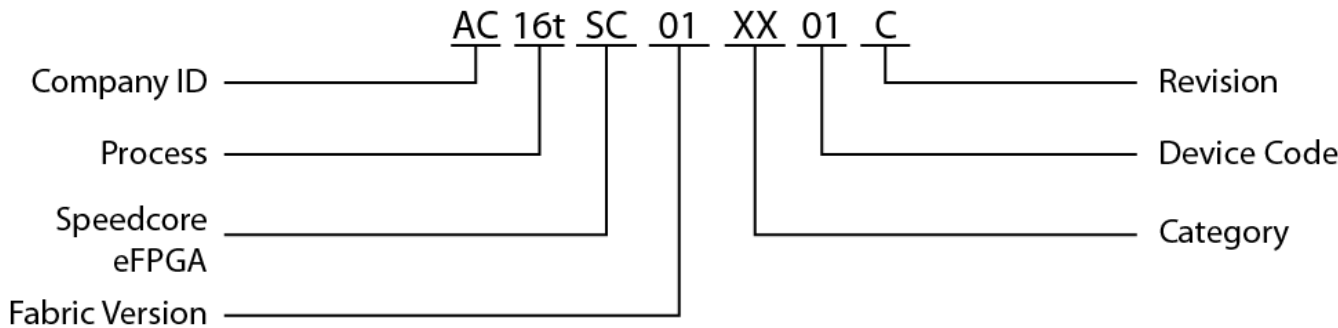
Customers can select the programming interface to be one or a combination of the following available options:

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

Please refer to the *Speedcore Power User Guide* (UG066) for details on connectivity and power rail sharing.

IP Nomenclature

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



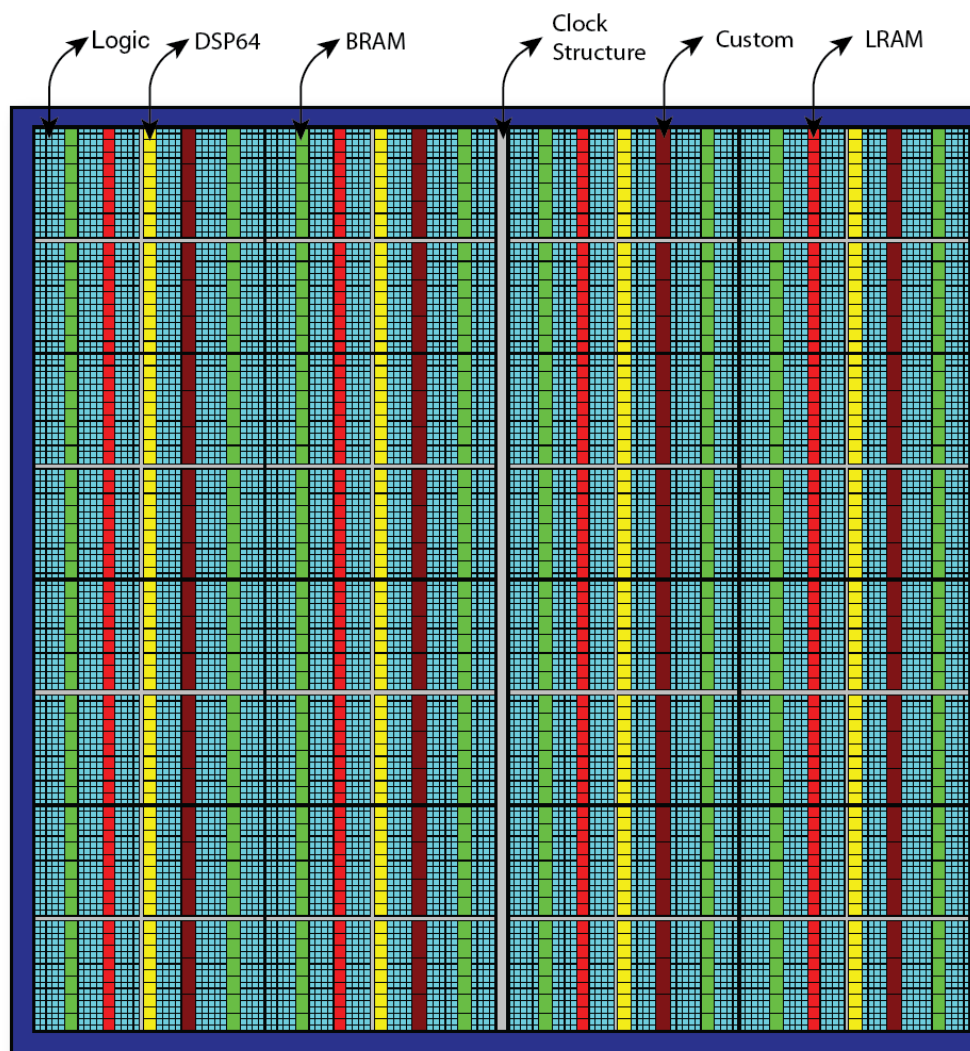
3703209-02.2018.10.25

Chapter - 2: Speedcore Architecture

Block Floorplan

The resource types are arranged in homogeneous columns. The height of the columns, number of columns and mix of column contents are customizable as defined by the customer. Implementing the exact mix of resources is performed by Achronix using automation, based on the specification set by the customer.

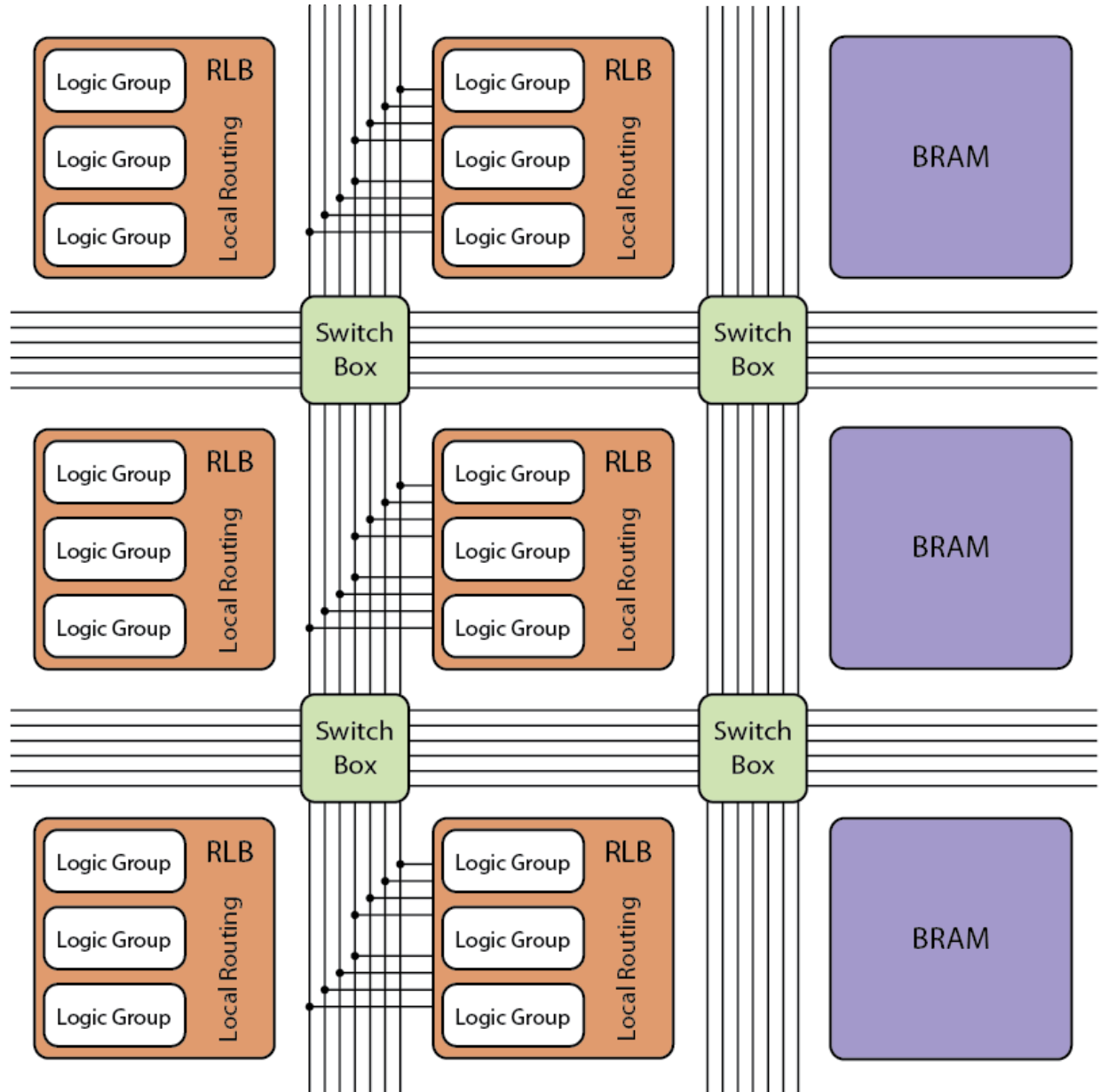
The Speedcore™ fabric performance is design dependent and is typically between 300 MHz and 500 MHz (F_{MAX}).



ds003-002-2019.01.14

Figure 2: Sample Speedcore eFPGA Floorplan

The reconfigurable logic blocks (RLBs), block RAMs (BRAMs), logic RAMs (LRAMs) and digital signal processing blocks (DSP64s) are connected by a uniform global interconnect. This 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.



ds003-003.2017.01.10

Figure 3: Speedcore eFPGA Interconnect

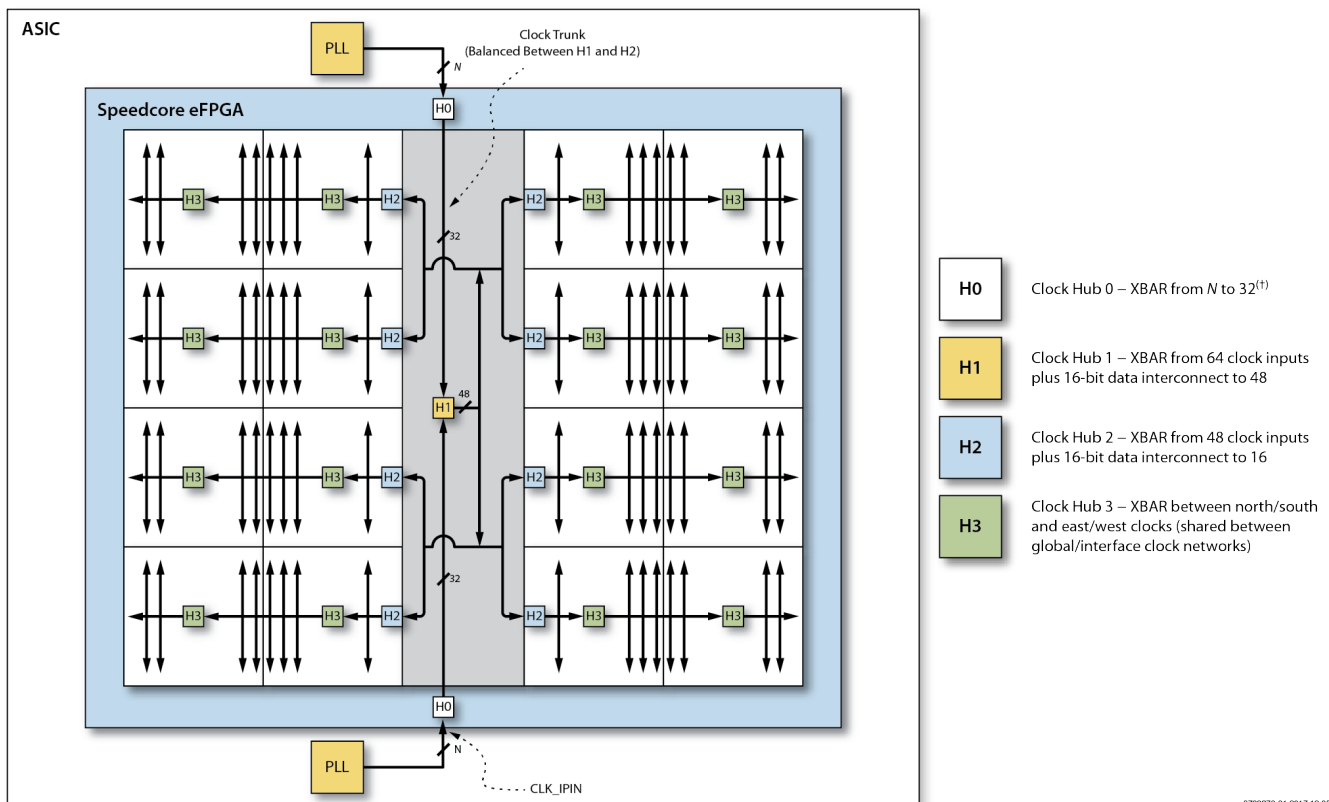
Speedcore Clock Network

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.



3703270-01.2017.12.05

Figure 4: Global Core Clock Network

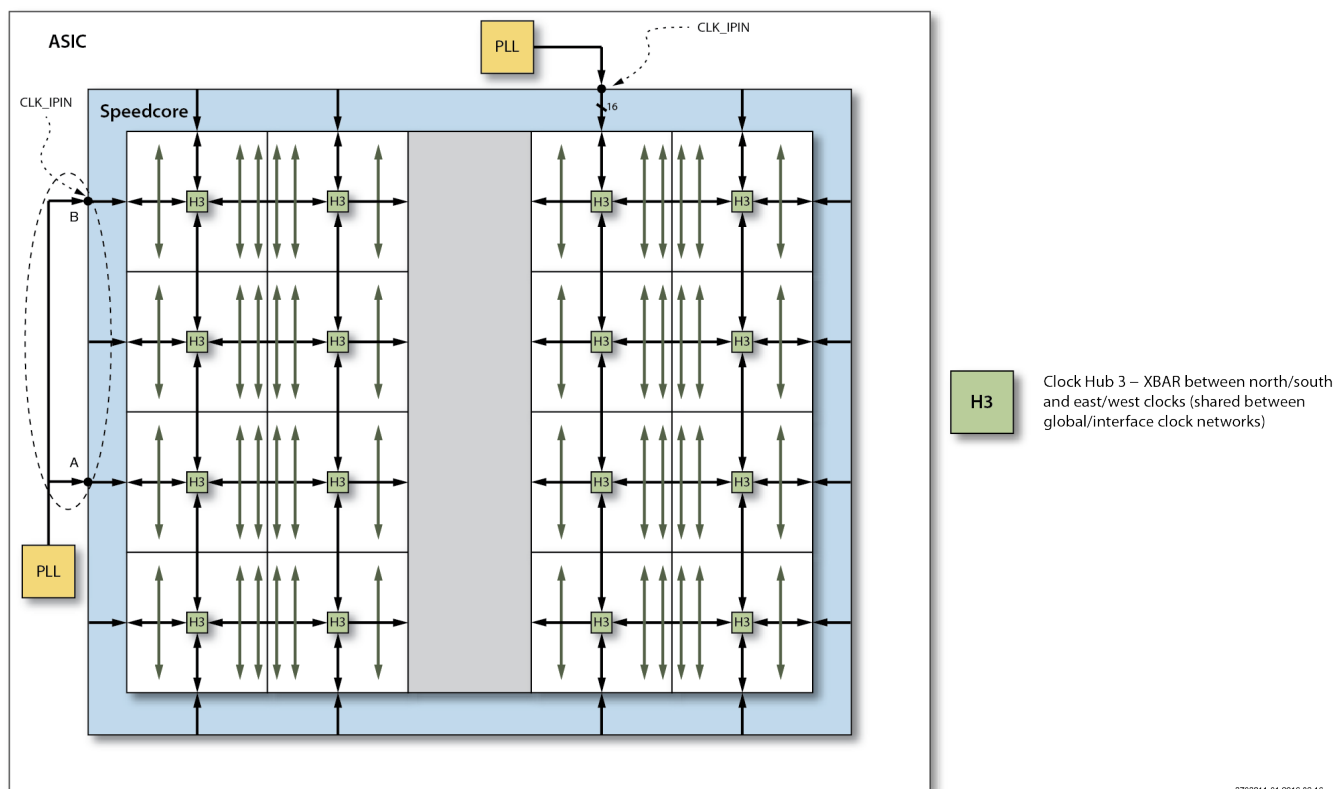


Figure 5: Interface Clock Network

Speedcore Interface Cluster

The interface cluster is the portion of the Speedcore boundary ring that contains the registers, ACB logic and the connectivity to the Speedcore top-level pins. The figure below shows the details of an interface cluster. The upper half shows the ingress and egress path for user signals to the core. Both paths can be optionally registered.

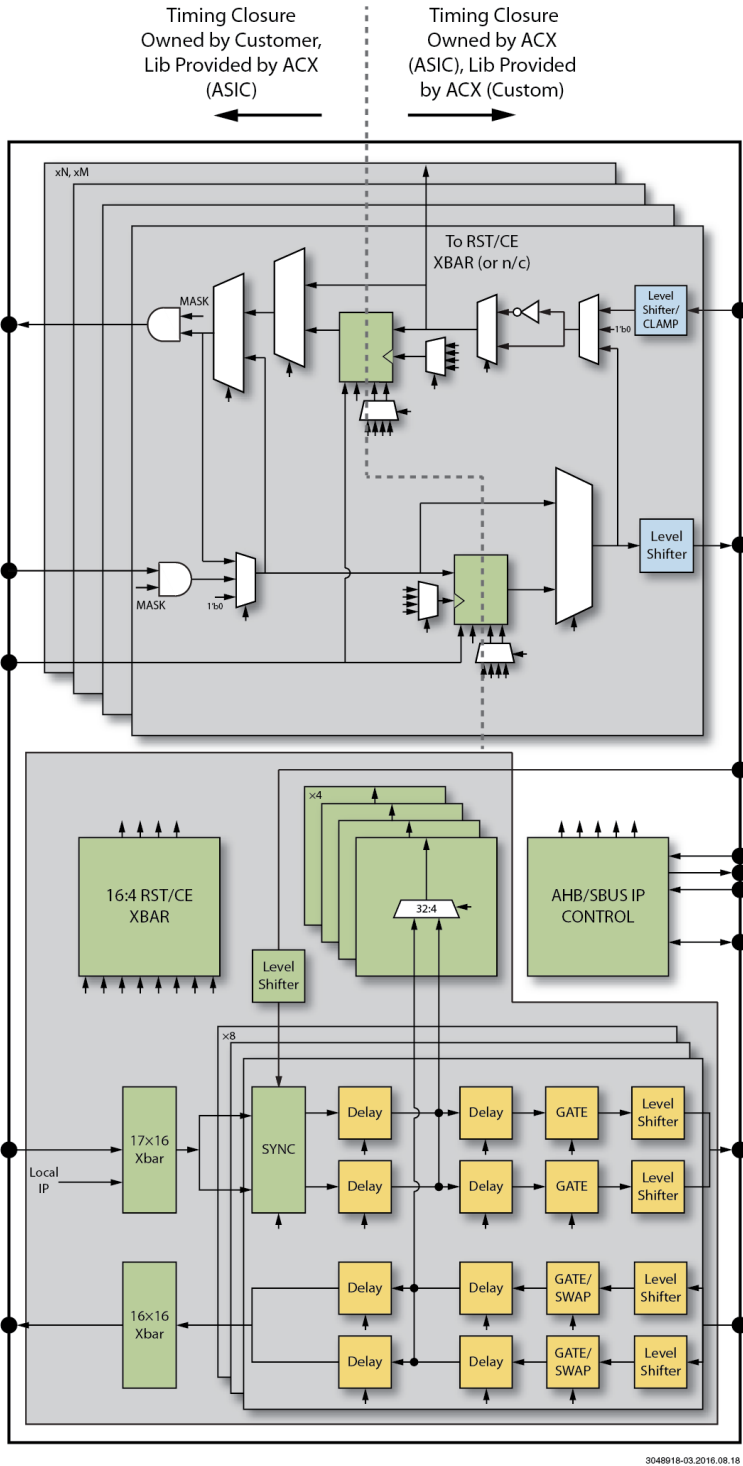
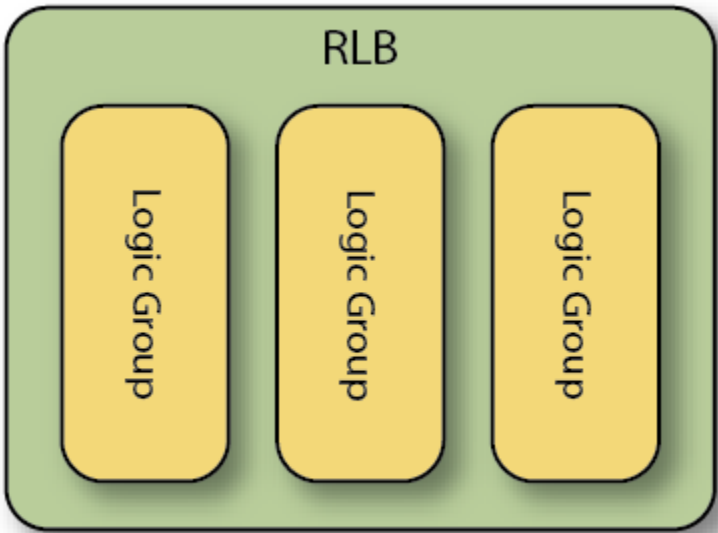


Figure 6: Interface Cluster Detail

Speedcore Logic Fabric – Reconfigurable Logic Block

The reconfigurable logic block (RLB) is composed of three parallel logic groups as shown in the RLB Block Diagram below.



3703211-08.2016.11.05

Figure 7: RLB Block Diagram

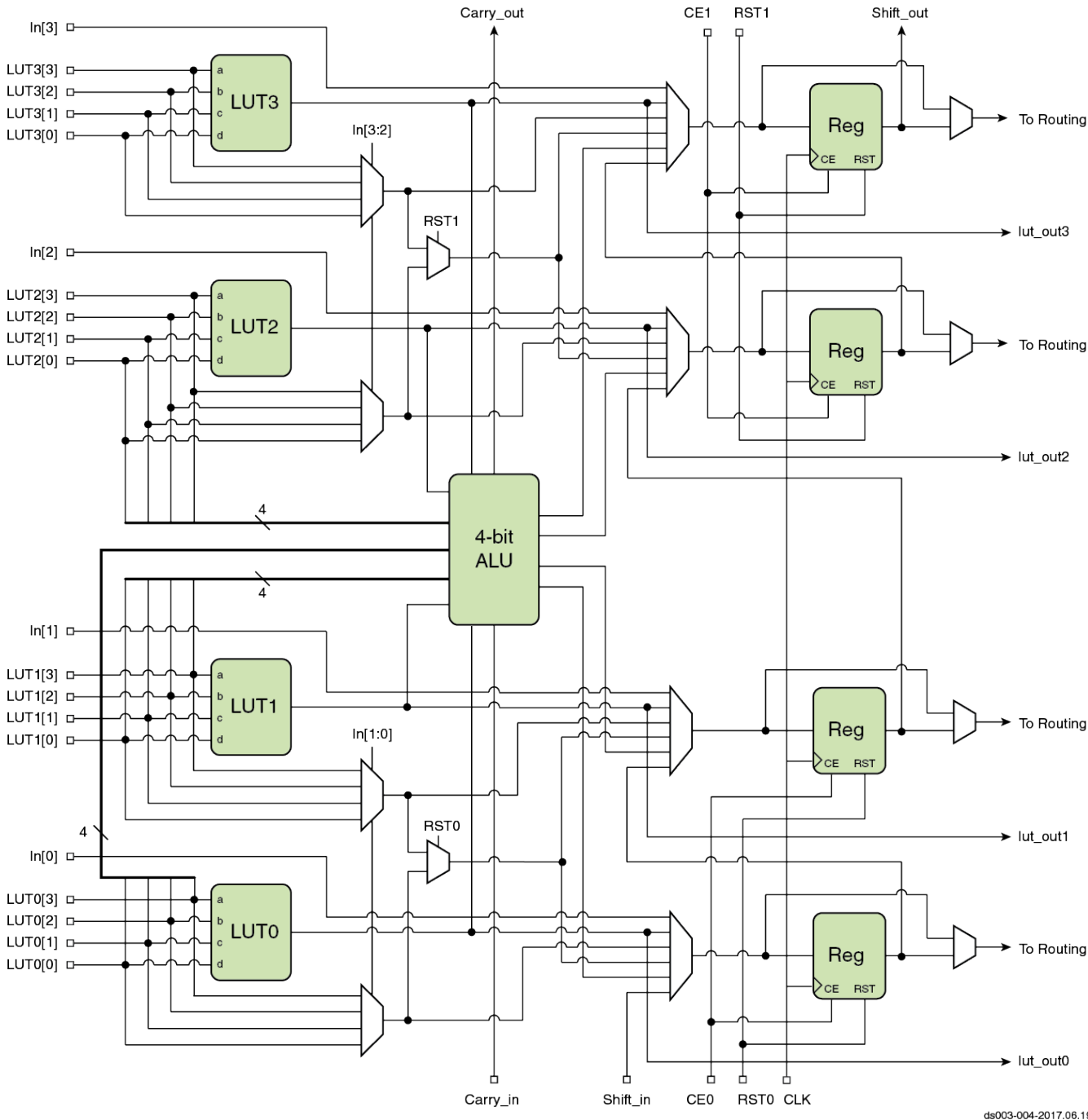
Each logic group contains four 4-input look-up-tables (LUT), each with an optional register and a 4-bit fast arithmetic logic unit (ALU) to implement logic functionality. A logic group also contains four dedicated 4-to-1 MUXes and two 8-to-1 MUXes. Each logic group receives a carry-in input from the corresponding logic group in the RLB to the north and can propagate a carry-out output to the corresponding logic group in the RLB to the south.

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

Table 1: RLB Resource Counts

| RLB Resources | Count |
|---------------|-------|
| Logic groups | 3 |
| 4-LUTs | 12 |
| Registers | 12 |
| ALU | 3 |
| MUX4 | 12 |
| MUX8 | 6 |

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

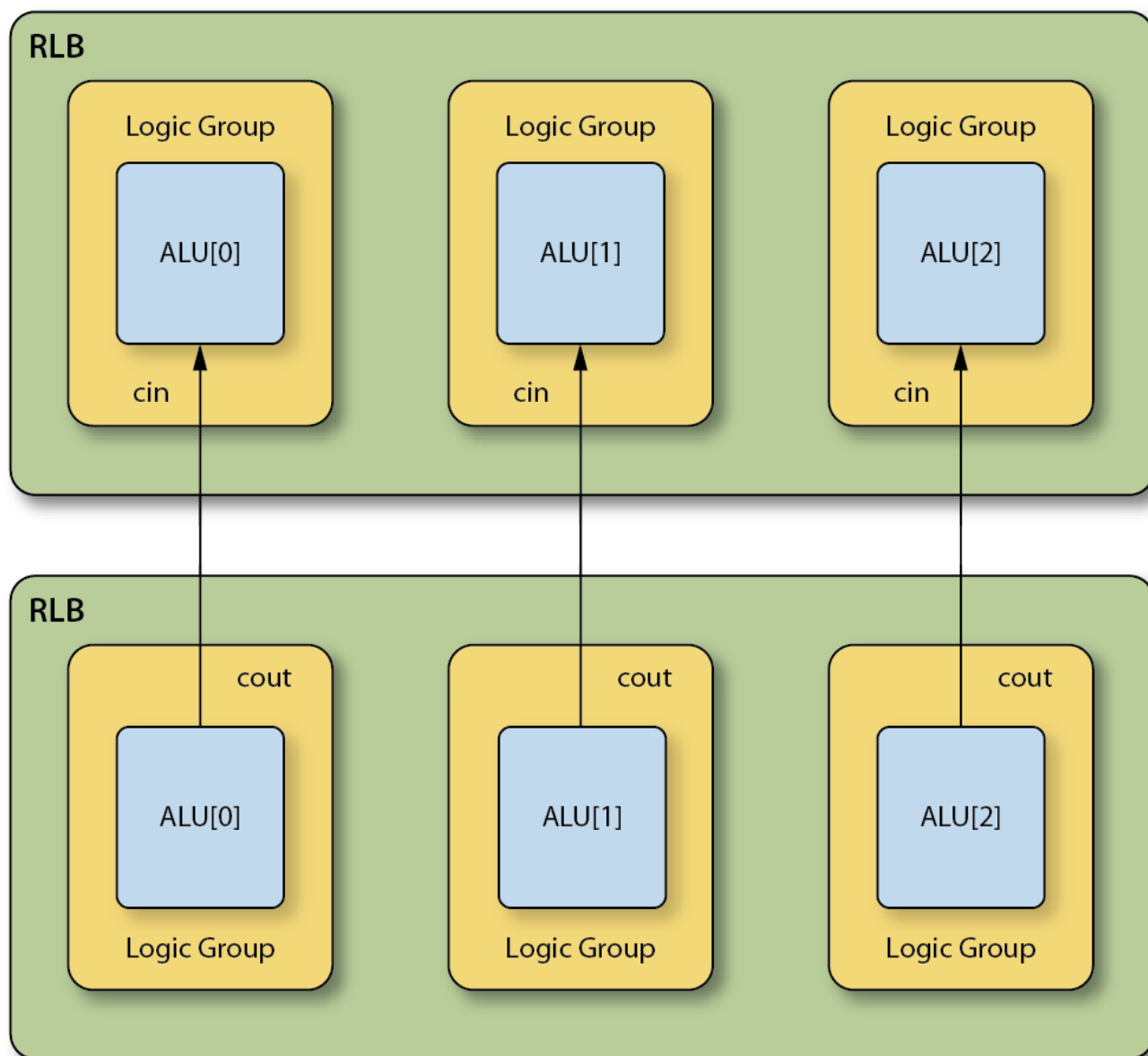


ds003-004-2017.06.19

Figure 8: Logic Group Details

Routing Between RLBs

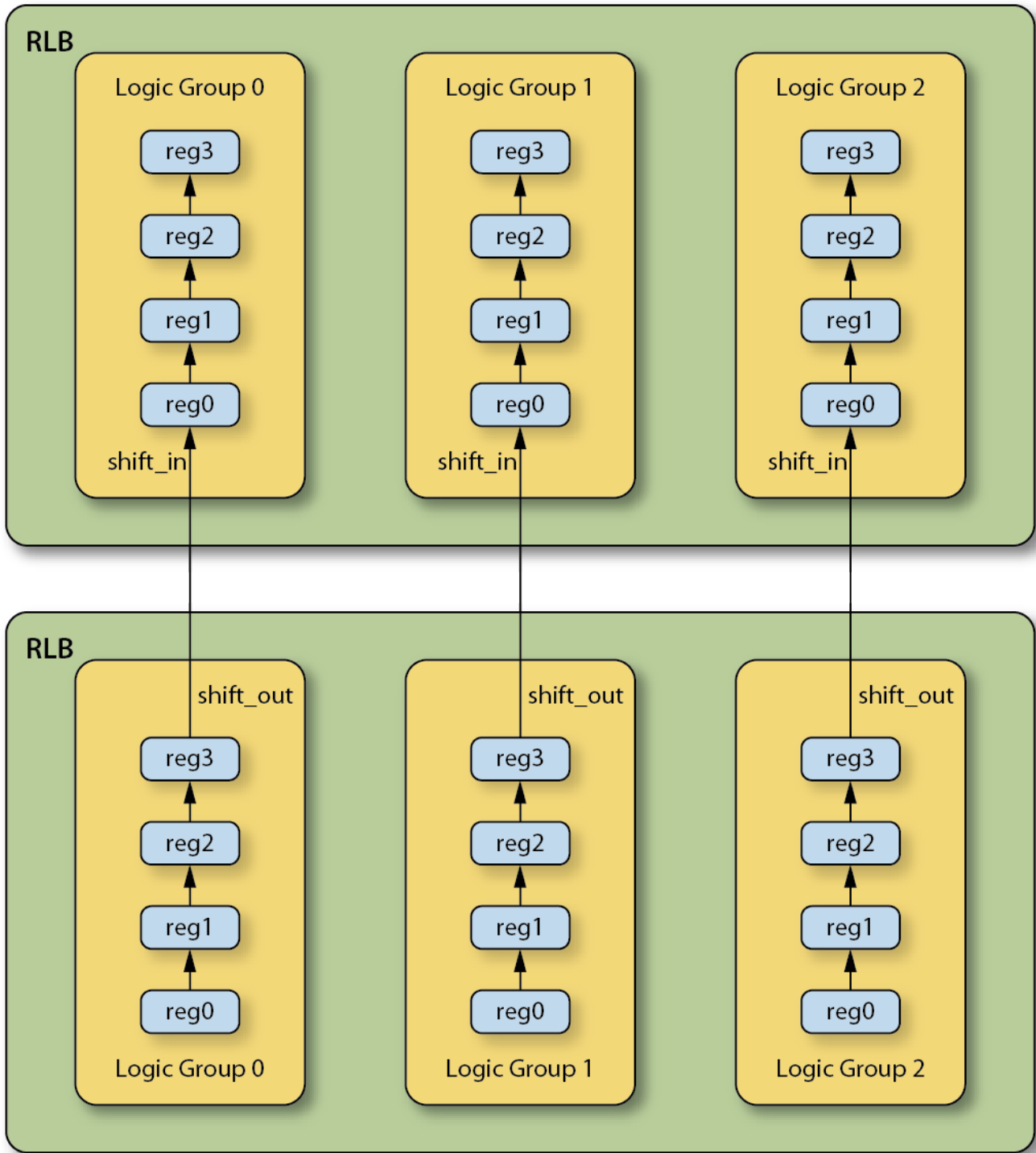
There are special considerations when routing ALU carry chains and shift registers. The Speedcore fabric has hard-wired connections on the signals `carry_in/carry_out` of each ALU. As mentioned above, each logic group routes to the corresponding logic group in the RLB above or below. In other words, 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 RLB. The figure below shows the `carry_in/carry_out` routing of an ALU.



3703211-10.2018.03.30

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 RLB. The figure below shows details of the routing in the Speedcore fabric.



3703211-11.2018.03.30


Figure 10: Shift Register Routing

Speedcore Memory Resources

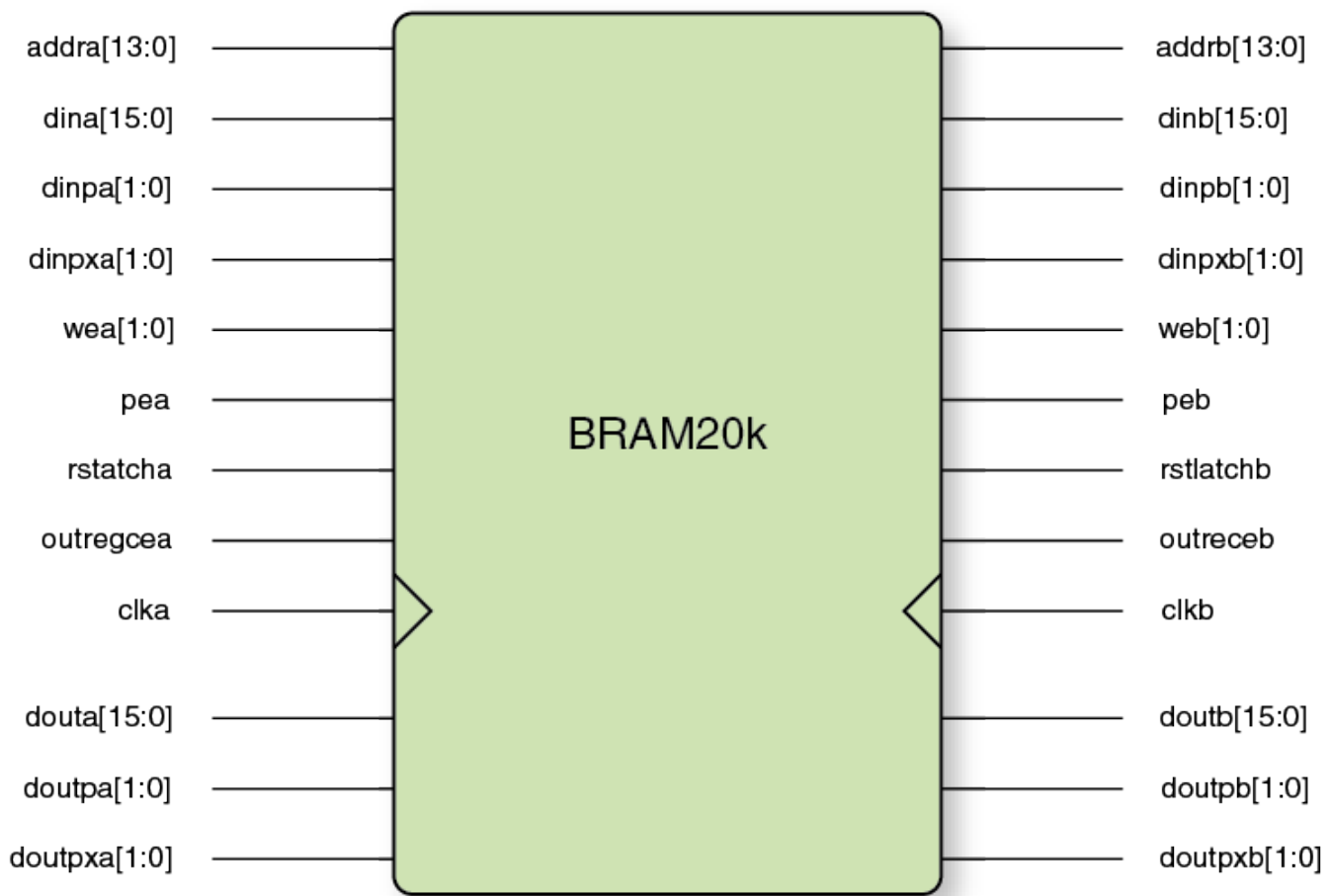
Block RAM 20k

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 |
| <div><div></div><div>Note † 512 × 40 only available as simple dual-port function.</div></div> | |

The BRAM20k ports are illustrated in the following figure:



ds003-005-2019.02.06

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.

Logic RAM 4k

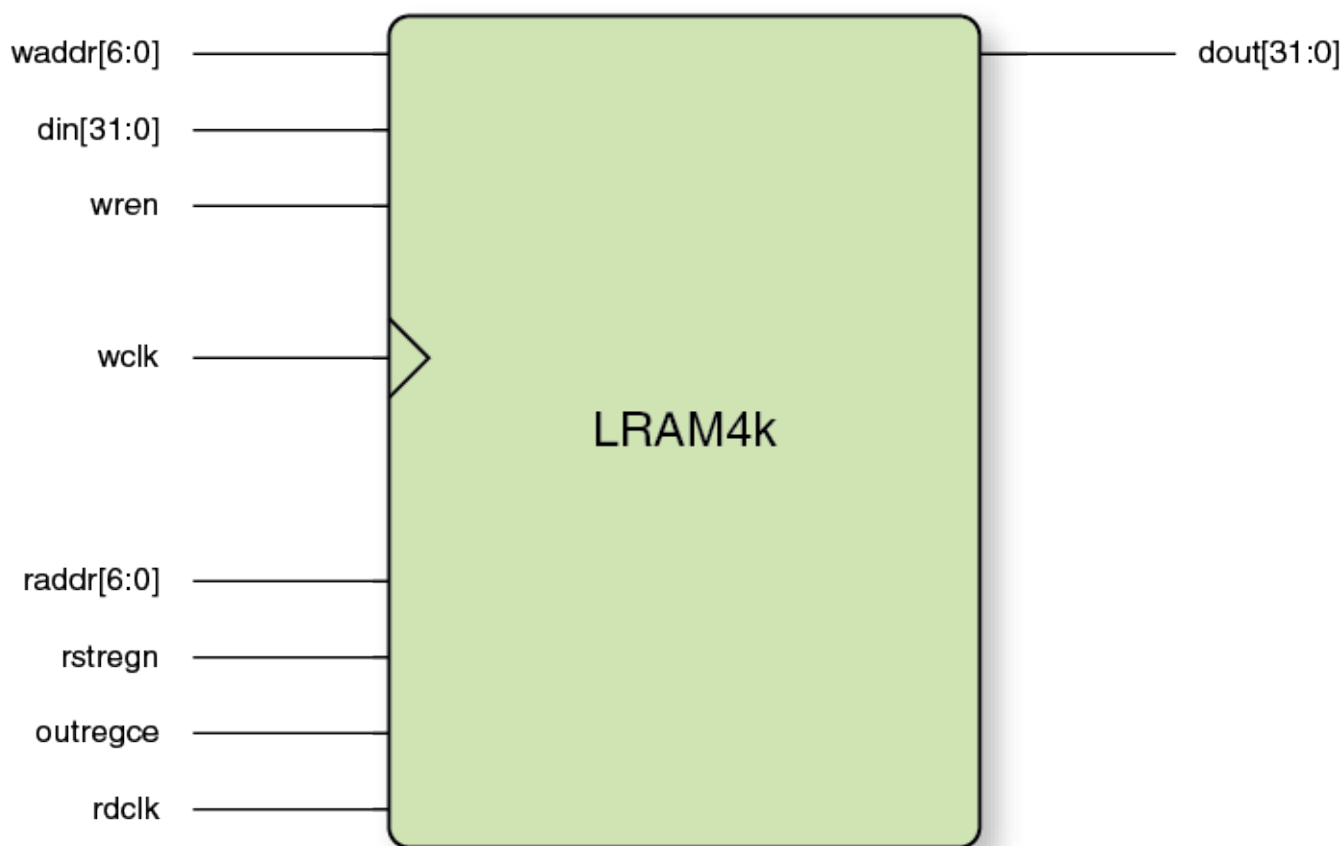
The LRAM4k implements a 4,096-bit memory block configured as a 128 × 32 simple dual-port (one write port, one read port) RAM. The LRAM4k has a synchronous write port. The read port is configured for asynchronous read operations with an optional output register. This memory block is distributed in the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

Table 3: LRAM4k Key Features

| Feature | Value |
|-------------------------|-------------------|
| Logic RAM size | 4,096 bits |
| Organization | 128 × 32 |
| Physical implementation | Dedicated columns |

| Feature | Value |
|-----------------|----------------------------------------|
| 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:



ds003-006-2019.02.06

Figure 12: 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

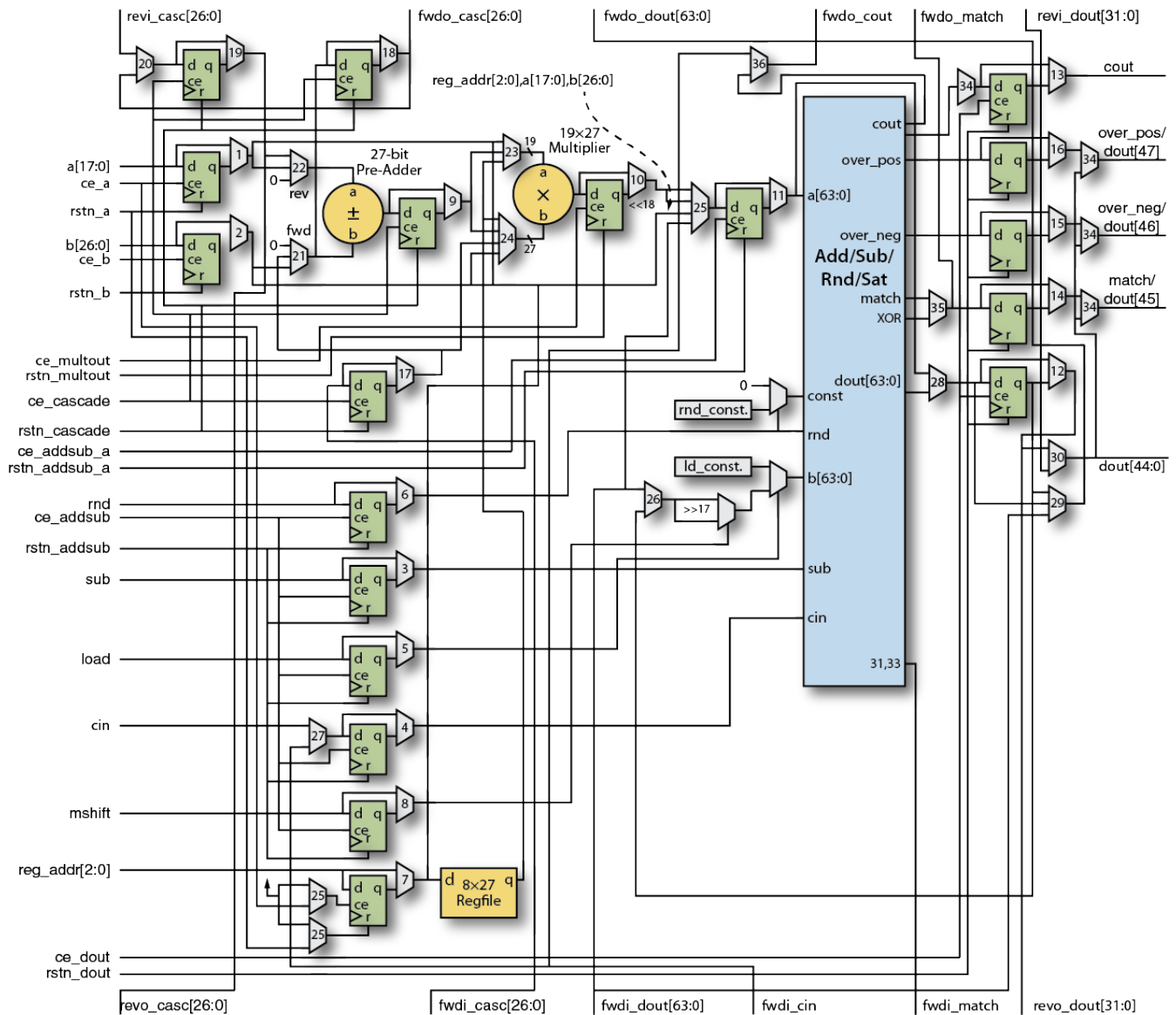
By default, the contents of the LRAM4k memory are undefined. If the user wants the initial contents to be defined, he may assign them from either a file pointed to by the `mem_init_file` parameter or assign them from the value of the `mem_init` parameter.

Speedcore DSP64 Blocks

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 $(A \pm B)^2 + \text{constant}$
- Output rounding



4228235-02.2018.01.24

Figure 13: DSP64 Block

Chapter - 3: Speedcore 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.

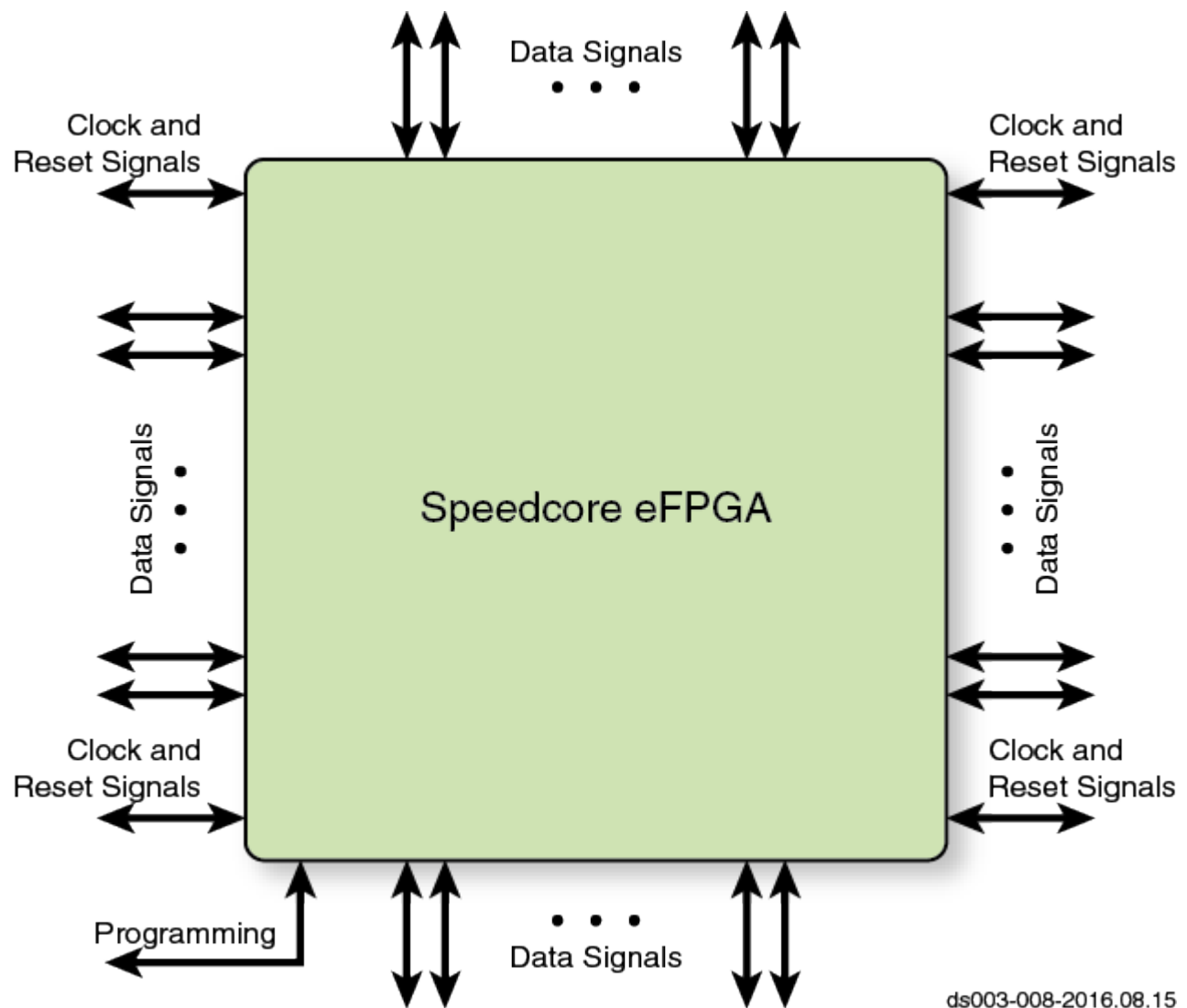


Figure 14: *Speedcore eFPGA Interfaces*

Pins

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

Table 4: *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. |

| Pin Name | Direction | Description |
|---------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| i_clock[c:0] | Input | Clock inputs to the programmable core. |
| 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. |

Chapter - 4: Speedcore 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.

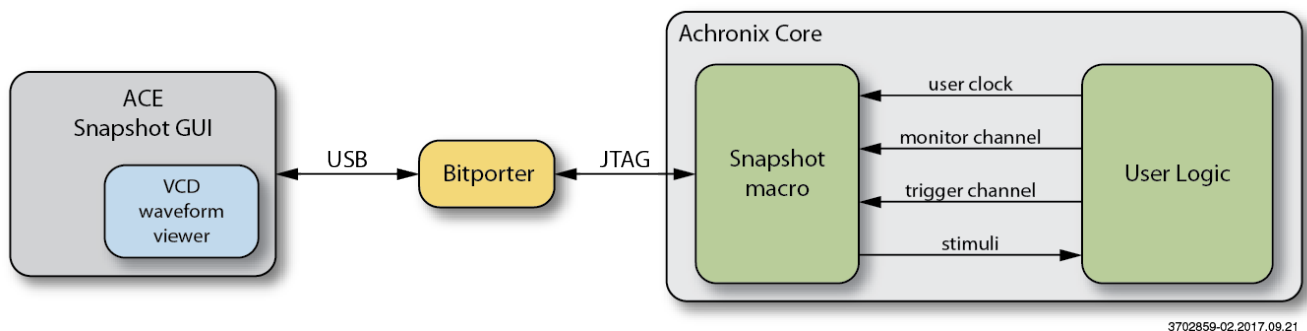


Figure 15: Snapshot Overview

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 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

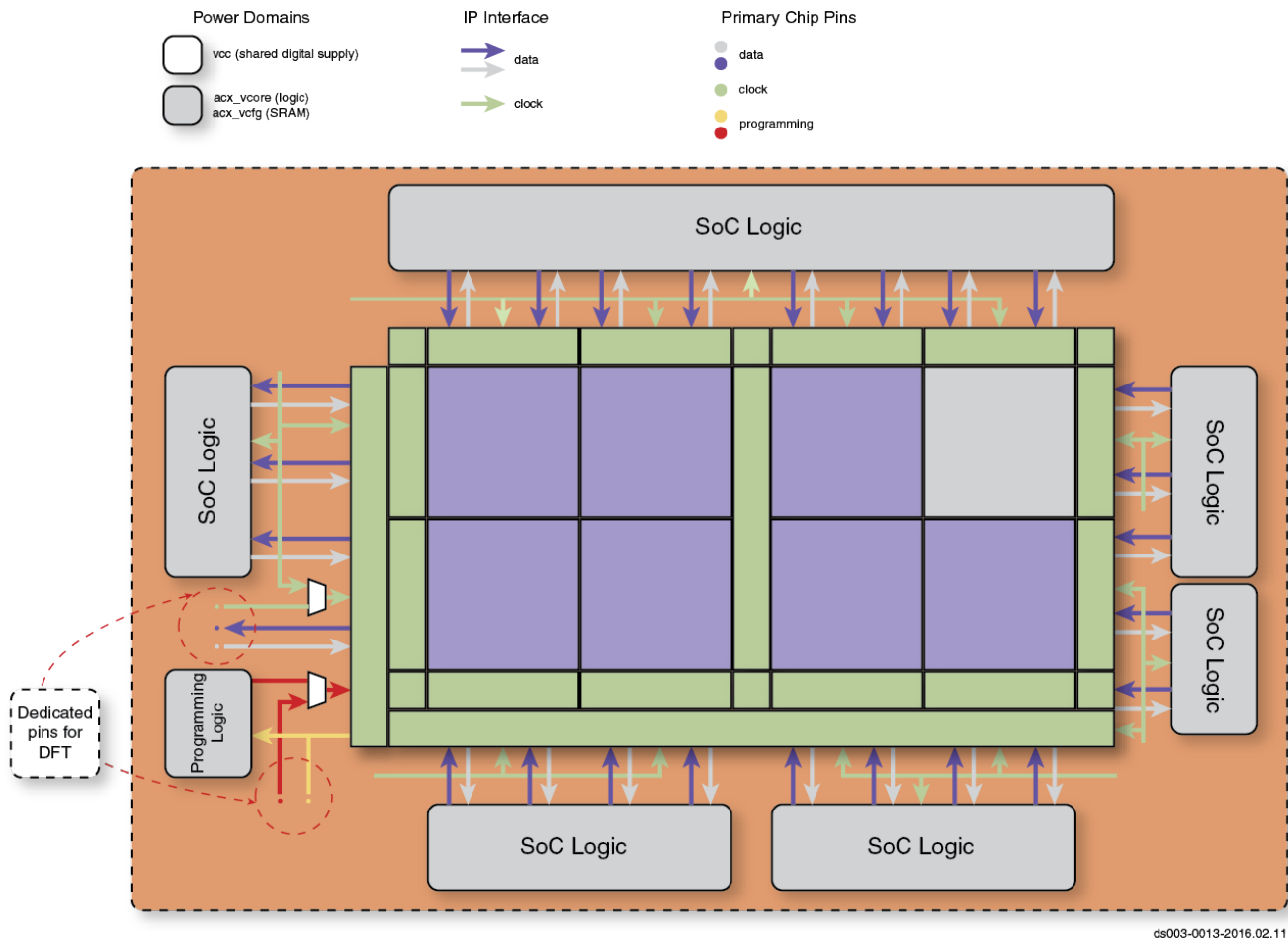


Figure 16: Sample eFPGA Instantiation

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, standard Liberty timing models of the user logic can be emitted and integrated into the user's system-level timing validation flow.

Revision History

The following table lists the revision history of this document.

| Version | Date | Description |
|---------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.0 | February 1, 2016 | <ul style="list-style-type: none"> Initial released version. |
| 1.1 | February 8, 2016 | <ul style="list-style-type: none"> Reorganized the structure: <ol style="list-style-type: none"> Sections Highlights and Introduction combined into a new chapter, Chapter 1 – "Overview" . Sections Logic Fabric, Memory Sources and DSP18x10 Blocks moved to the chapter, Chapter 2 – "Architecture" . Section Speedcore IP Interface moved to a separate chapter, Chapter 3 – "Speedcore IP Interface" . Added Chapter 6 – "Product Specification" DSP block nomenclature changed from DSP18x19 to DSP64. All figures updated. Other minor edits and corrections. |
| 1.2 | February 21, 2016 | <ul style="list-style-type: none"> Update voltages listed in "Power Supplies" in Chapter 1 – Overview. Added section "Temperature Range" in Chapter 1 – Overview. BRAM size updated from 80 kb to 20 kb; port widths adjusted accordingly. Other updates and edits. |
| 1.3 | March 8, 2016 | <ul style="list-style-type: none"> Converted to Confluence |
| 1.4 | March 31, 2016 | <ul style="list-style-type: none"> Clarification regarding ECC mode. Minor edits and corrections. |
| 1.5 | April 4, 2016 | <ul style="list-style-type: none"> Updated DSP multiplier size to 18×27. Clarified information on process technology. Corrected information on clock counts. |
| 1.6 | April 6, 2016 | <ul style="list-style-type: none"> Changed the process description from "TSMC 16FPGL(FF16+ GL option)" to "TSMC 16FFplus-GL". |

| Version | Date | Description |
|---------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.7 | April 6, 2016 | <ul style="list-style-type: none"> Made corrections for BRAM/LRAM organization and operation. |
| 1.8 | July 25, 2016 | <ul style="list-style-type: none"> Re-structured content and moved device specific information, including resource counts, into an Appendix section. |
| 1.9 | August 5, 2016 | <ul style="list-style-type: none"> Corrected naming convention for LRAM and a connection in the RLB Details diagram. |
| 1.10 | August 19, 2016 | <ul style="list-style-type: none"> Updated wording for DSP64 pre-adder in text and block diagram. Put in sections on the Clock Architecture and Interface Cluster. Corrected eFPGA interfaces figure and modified description of eFPGA pins. Standardized configuration memory cell power supply name to V_{CFG}. |
| 1.11 | October 31, 2016 | <ul style="list-style-type: none"> Updated document template to include the confidentiality note. |
| 1.12 | November 6, 2016 | <ul style="list-style-type: none"> Added details, a new figure and resource counts for the RLB section of the Speedcore Architecture chapter. Adjusted device specs in the customer-specific Appendix. |
| 1.13 | November 28, 2016 | <ul style="list-style-type: none"> Appendix: Updated name and resource counts, and put in actual figures for the cores. |
| 1.14 | December 9, 2016 | <ul style="list-style-type: none"> Overview (see page 4): Added a section on IP Nomenclature to decode the Speedcore IP part number. |
| 1.15 | July 9, 2017 | <ul style="list-style-type: none"> Speedcore Architecture (see page 6): Updated Figure: DSP64 Block (see page). Speedcore Architecture (see page 6): Updated Figure: Logic Group Details (see page). |
| 1.16 | July 11, 2017 | <ul style="list-style-type: none"> Speedcore Architecture: Annotated Interconnect figure to clarify RLB locations. |

| Version | Date | Description |
|---------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.17 | October 16, 2017 | <ul style="list-style-type: none">• Speedcore Architecture: Small edit to clarify that each logic group has four 4-LUTs. |
| 1.18 | February 19, 2018 | <ul style="list-style-type: none">• Appendix: Updated tables and figures to provide more accurate die area information for the cores. |
| 1.19 | May 3, 2018 | <ul style="list-style-type: none">• Appendix: Generated a section to provide details on the Speedcore evaluation devices, and corrected some of the details in the customer-specific devices. |
| 1.20 | May 22, 2018 | <ul style="list-style-type: none">• Appendix: Corrected numbers for total BRAM memory in the different devices. |
| 1.21 | January 14, 2019 | <ul style="list-style-type: none">• Re-branded entire document with new formatting, logistical corrections and logos.• Overview (see page 4): Removed information on power supplies.• Speedcore Architecture (see page 6): Updated Figure: Sample Speedcore eFPGA Floorplan (see page 6).• Appendix: Removed evaluation cores as this information is now available in the Speedcore Device Catalog. |
| 1.22 | September 26, 2019 | <ul style="list-style-type: none">• Modified document name to explicitly call out that it is specific to Gen3.• Appendix: Removed the hyphens from the device names. |