Speedcore eFPGA Datasheet (DS012)

Speedcore eFPGA

Achronix
Data Acceleration

Copyrights, Trademarks and Disclaimers

Notice of Disclaimer

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 eFPGA

Achronix Speedcore™ embedded FPGA (eFPGA) IP includes look-up-table, memory, DSP, and MLP building blocks. Each of these blocks are designed to be modular to allow the definition of any mix of resources required for a custom end system.

Achronix delivers the Speedcore IP in OASIS format along with all files and documentation required for the integration of the Speedcore eFPGA instance into a custom ASIC. Achronix also delivers the supporting ACE design tools that are used to compile designs targeting the Speedcore eFPGA.

Feature Summary

Because the Speedcore eFPGA is an embedded IP, it is designed to be completely surrounded by the custom 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)

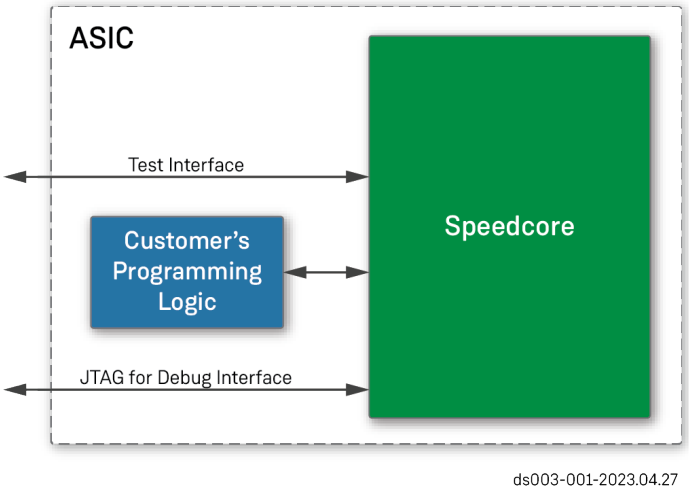

Figure 1 • Embedded Speedcore

Functionality

The functionality of the Speedcore eFPGA is defined by choosing the quantity of each of the resources listed below:

- **Logic** – 6-input look-up-tables (LUTs) plus integrated fast adders
- **Local RAMs** – (two options):
  - Up to 2Kb per memory block for LRAM2k (included as tightly-coupled memory with MLP)
  - Up to 4Kb per memory block for LRAM4k
- **Block RAM** – (two options):
  - Up to 20Kb per memory block for BRAM20k
  - Up to 72Kb per memory block for BRAM72k (optionally included as tightly-coupled memory with MLP)
- **DSP64** – each block has a 18 × 27 multiplier, 64-bit accumulator and 27-bit pre-adder
- **MLP** – machine learning processor (MLP) blocks containing multipliers, adders, accumulators, and tightly-coupled memory (includes BRAM72k and LRAM2k)

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

Programming

The programming interface can be selected 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

Chapter 2 : Speedcore eFPGA 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, MLPs, 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, with all connections between tiles 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.

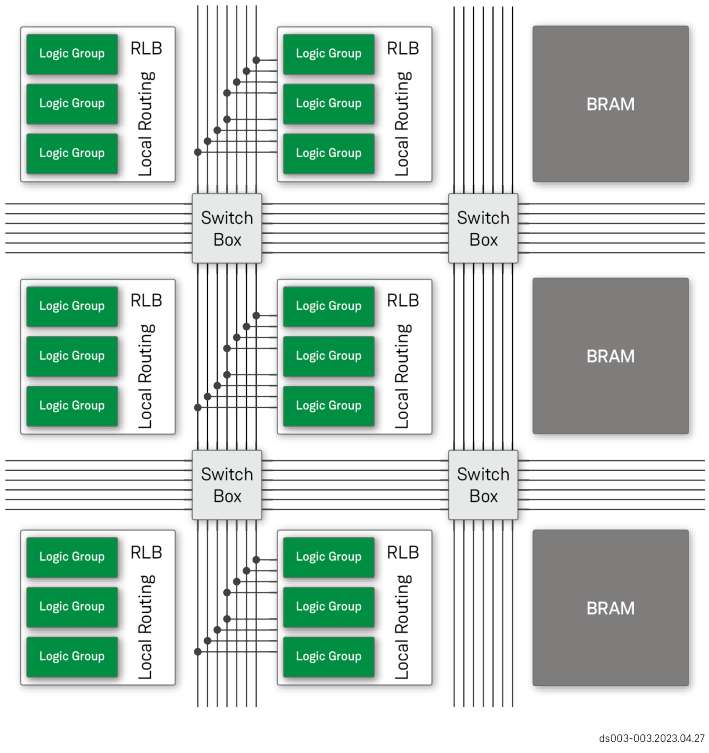

Figure 2 • Speedcore eFPGA Interconnect

Speedcore eFPGA Clock Network

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

The hierarchical global clock network feeds resources within the eFPGA fabric. The global clock trunk runs vertically up and down the center of the core (the gray stripe in the following figure), sourced by global clock muxes at the top and bottom of the global trunk. The sources driven on the trunk are then distributed to all clock regions on both the left and right halves of the core.

Within Speedcore eFPGAs there is a second clock network, the interface clock network, available at the periphery of core. 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 surrounding 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.

The two clock networks are detailed in the following two figures:

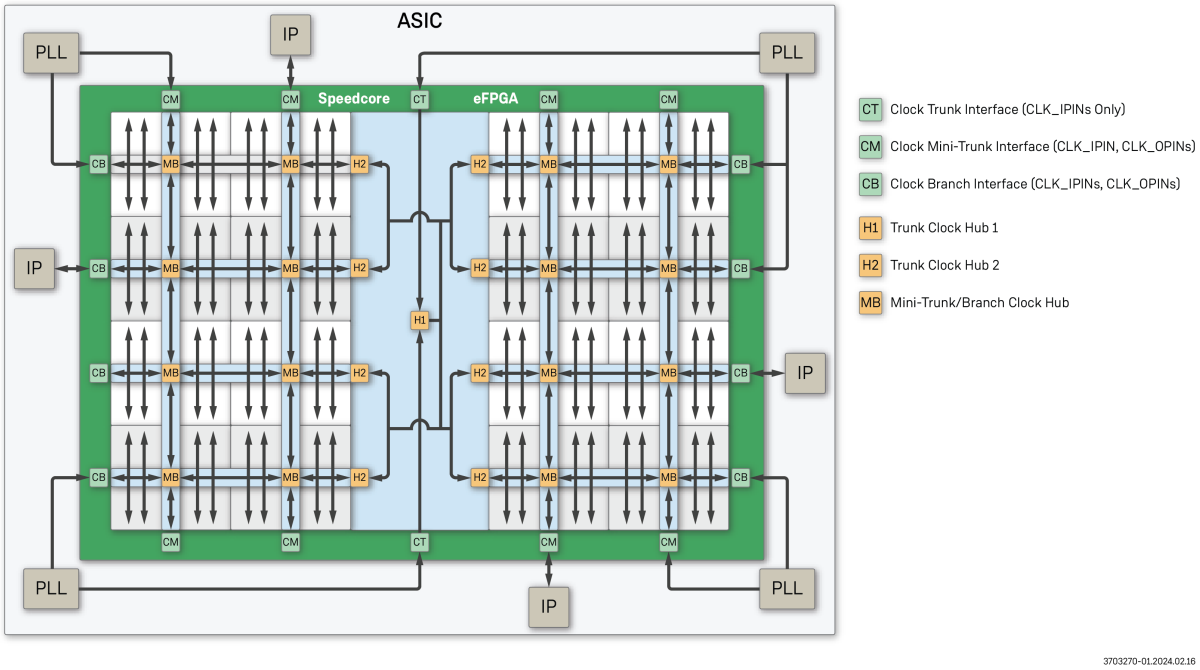

Figure 3 • Global Core Clock Network

In the following figure, the dots represent the clock interface clusters. In the clock interface cluster, these four clocks can be routed on up to eight clock signals going into the fabric. For further details, refer to the section on the **Speedcore Interface Cluster** (page 5).

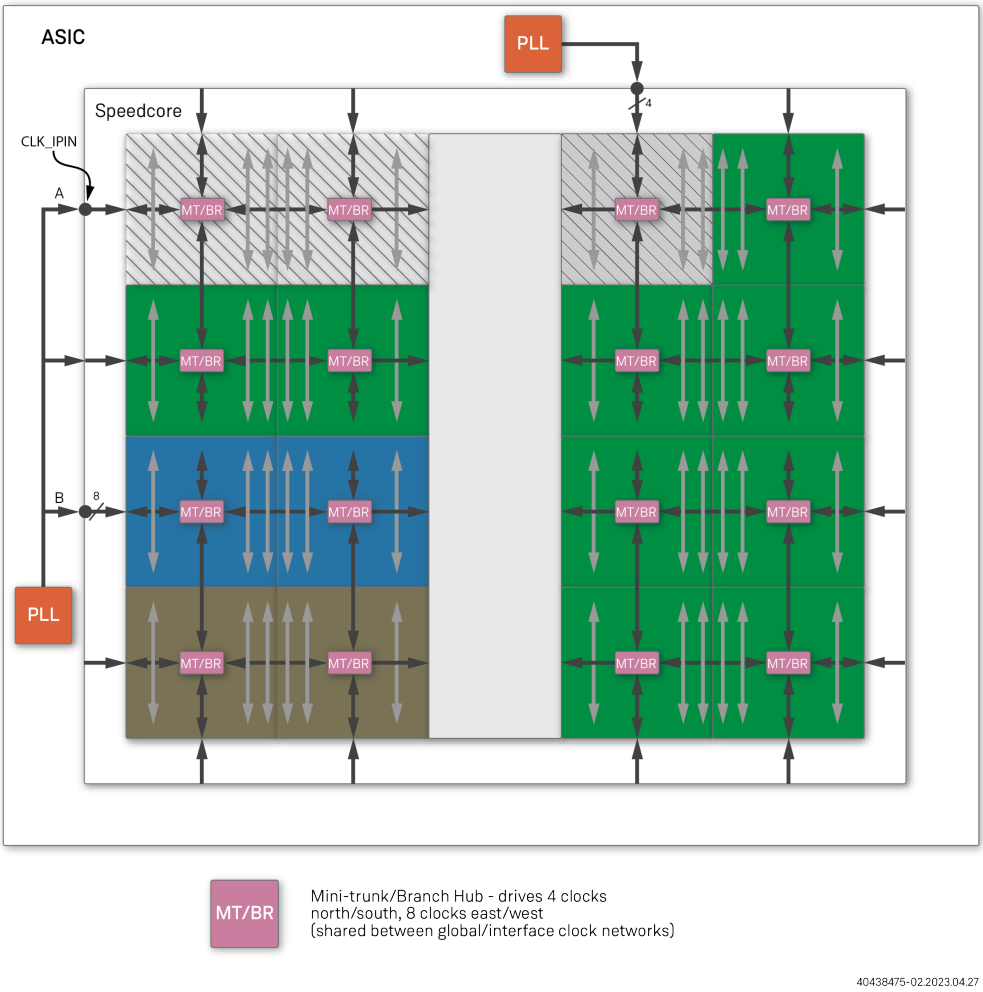

Figure 4 • Interface Clock Network

Speedcore eFPGA Interface Cluster

Each Speedcore instance has a boundary ring, composed of interface clusters, between the Speedcore fabric logic and the surrounding ASIC logic. The interface cluster is the portion of the Speedcore boundary ring that contains the IPINs and OPINs, configuration memory logic, and the connectivity between the Speedcore top-level pins which connect to ASIC logic and the Speedcore fabric. IPINs and OPINs are bypassable registers which represent the connection point between the ASIC logic and the Speedcore logic.

The following figures show a device-specific example 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 a device-specific example of how clocks enter and exit the fabric, including optional delays.

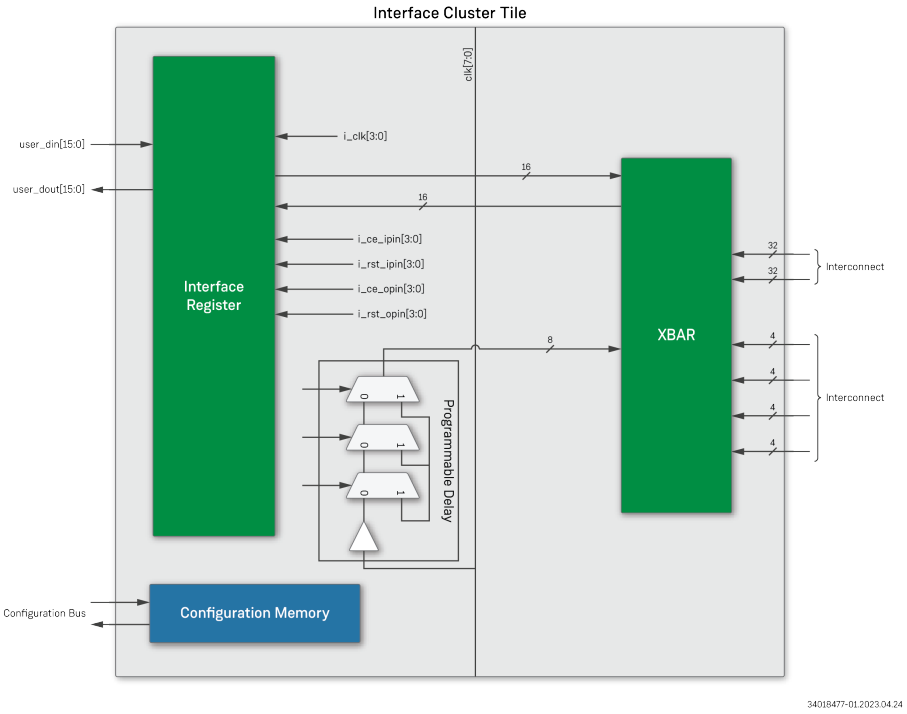

Figure 5 • Interface Cluster Logic Tile

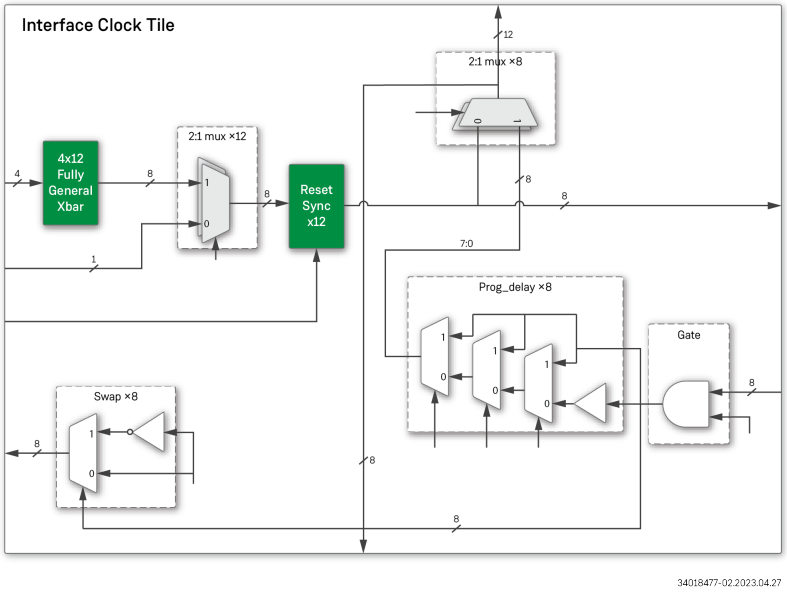

Figure 6 • Interface Cluster Clock Tile

Interface Timing Closure

The Speedcore interface clusters do not have programmable I/O that connect directly to the pads on the host ASIC. Instead the Speedcore 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 eFPGA instance specified.

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

Speedcore eFPGA Logic Fabric — Reconfigurable Logic Block

An RLB contains multiple 6-input look-up-tables (LUT6), a number of registers, and an 8-bit fast arithmetic logic unit (ALU8). The resource counts are however device and architecture/family specific.

The following device-specific features are available using the resources in the RLB:

- 8-bit ALU for adders, counters, and comparators
- MAX function that efficiently compares two 8-bit numbers and chooses the maximum or minimum result
- 8-to-1 MUX with single-level delay
- Support for LUT chaining within the same RLB and between RLBs
- Dedicated connections for high-efficiency shift registers
- Ability to fan-out a clock enable or reset signal to multiple tiles without using general routing resources
- 6-input LUT configurable to function as two 5-input LUTs using shared inputs and two outputs
- Support for combining two 6-input LUTs with a dynamic select to provide 7-input LUT functionality

Speedcore eFPGA 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 following table:

Table 1 • BRAM20k Key Features

Feature	Value
Block RAM Size	20Kb

Feature	Value
Organization [1]	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

Table Notes

1. 512 × 40 organization is only available as a simple dual-port function.

The BRAM20k ports are illustrated in the following figure:

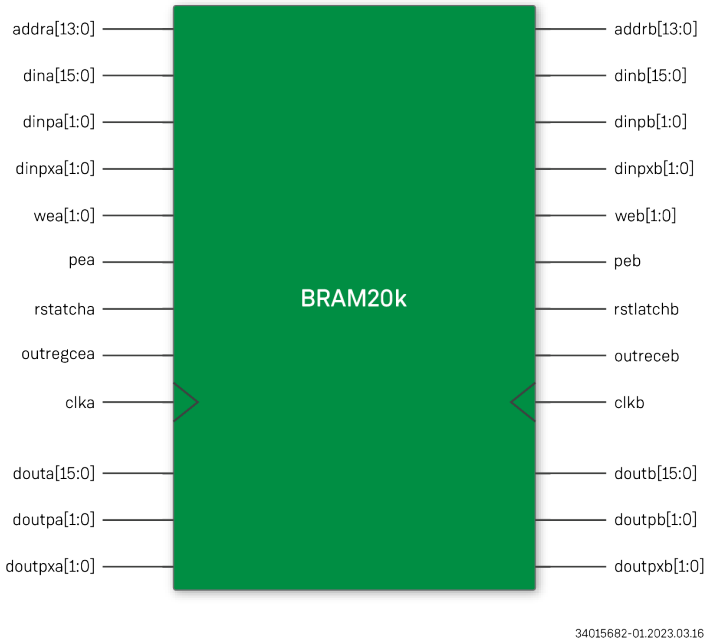

Figure 7 • BRAM20k Ports

Block RAM 72k

The BRAM72K primitive implements a 72Kb 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 following table:

Table 2 • BRAM72K Key Features

Feature	Value
Block RAM size	72Kb
Organization	512 × 144, 128 × 512, 1024 × 72, 1024 × 64, 2048 × 36, 2048 × 32, 4096 × 18, 4096 × 16, 8192 × 9, 8192 × 8, or 16384 × 4
Physical Implementation	Columns throughout device
Number of Ports	Simple Dual Port (independent read and write)
Port Access	Synchronous writes, synchronous reads (write and read clocks 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:

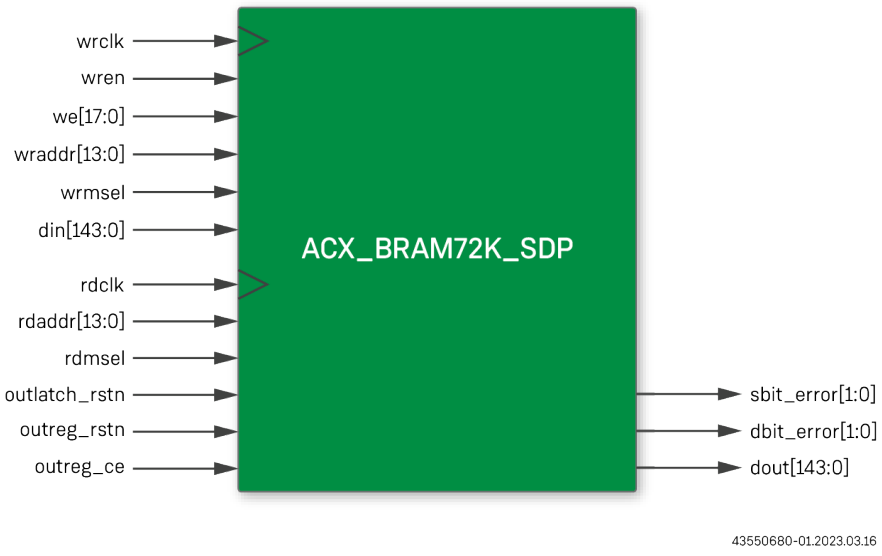

Figure 8 • BRAM72K Block Diagram

Speedcore eFPGA Local RAM

Local 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. A summary of LRAM2K features is shown in the following table:

Table 3 • LRAM2K Key Features

Feature	Value
Local RAM size	2,304 bits
Organization	16 × 144, 32 × 72 or 64 × 36 (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

The LRAM2K ports are shown in the following figure:

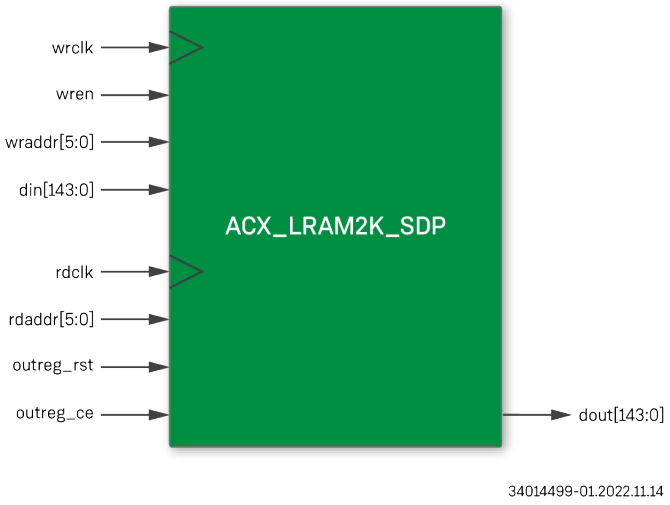

Figure 9 • LRAM2K Block Diagram

Local 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 following table:

Table 4 • LRAM4k Key Features

Feature	Value
Local 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:

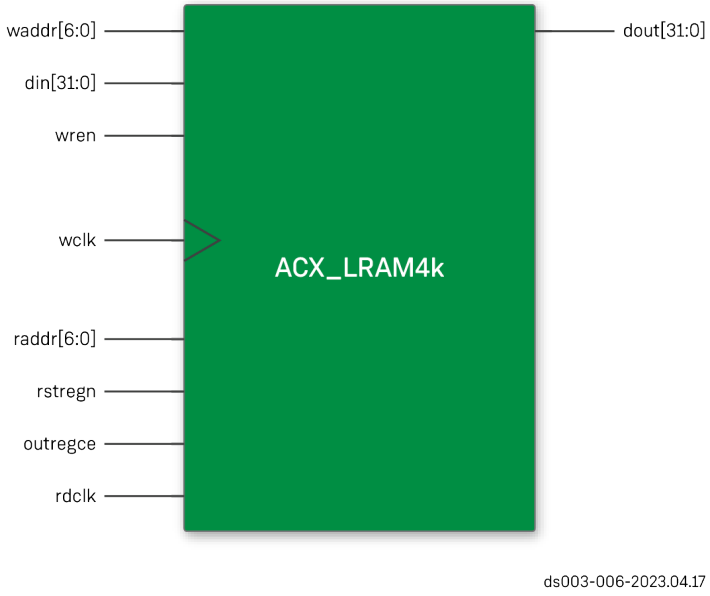

Figure 10 • LRAM4k Ports

Speedcore eFPGA 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 Component Library User Guide*** **(UG065)**[1] for more details.

The DSP64 blocks have the following functions:

- 27-bit pre-adder
- 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

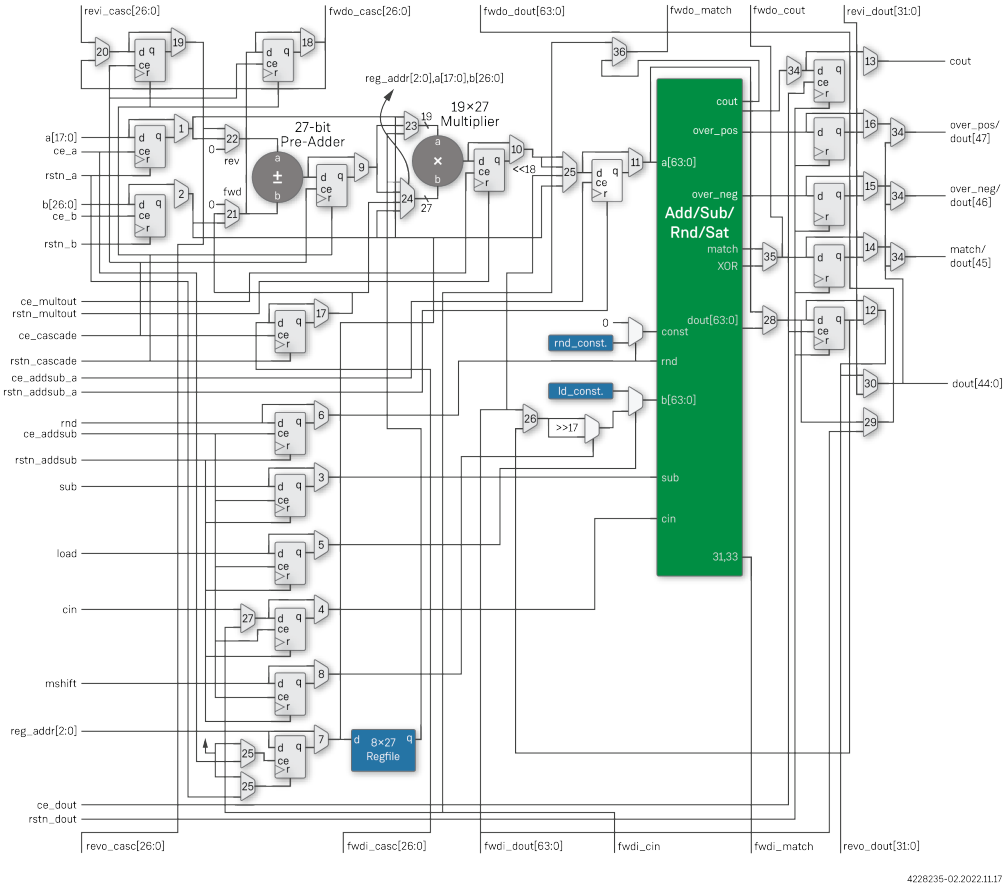

Figure 11 • DSP64 Block

1 https://www.achronix.com/documentation/speedcore-component-library-user-guide-ug065

Speedcore eFPGA Machine Learning Processor (MLP) Block

The machine learning processor block (MLP) is an array of up to 32 multipliers, followed by an adder tree, and an accumulator. The MLP is also tightly coupled with two memory blocks, a BRAM72k and LRAM2k. These memories can be used individually or in conjunction with the array of multipliers. The number of multipliers available varies with the bit width of each operand and the total width of input data. When the MLP is used in conjunction with a BRAM72k, the number of data inputs to the MLP block increases, enabling the use of additional multipliers.

The MLP offers a range of features:

- Configurable multiply precision and multiplier count. Any of the following modes are available:
  - Up to 32 multiplies for 4-bit integers or 4-bit block floating-point values in a single MLP
  - Up to 16 multiplies for 8-bit integers or 8-bit block floating-point values in a single MLP
  - Up to 4 multiplies for 16-bit integers in a single MLP
  - Up to 2 multiplies for 16-bit floating point with both 5-bit and 8-bit exponents in a single MLP
  - Up to 2 multiplies for 24-bit floating point in a single MLP
- Multiple number formats:
  - Integer
  - Floating point 16 (including B float 16)
  - Floating point 24
  - Block floating point, a method that combines the efficiency of the integer multiplier-adder tree with the range of the floating point accumulators
- Adder tree and accumulator block
- Tightly-coupled register file (LRAM) with an optional sequence controller for easily caching and feeding back results
- Tightly-coupled BRAM for reusable input data such as kernels or weights
- Cascade paths up a column of MLPs
  - Allows for broadcast of operands up a column of MLPs without using up critical routing resources
  - Allows for adder trees to extend across multiple MLPs
  - Broadcast read/write to tightly-coupled BRAMs up a column of MLPs to efficiently create large memories

Along with the numerous multiply configurations, the MLP block includes optional input and pipelining registers at various locations to support 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. In addition, a feedback path allows for accumulation within the MLP block.

The following block diagrams show the MLP using the fixed or floating-point formats:

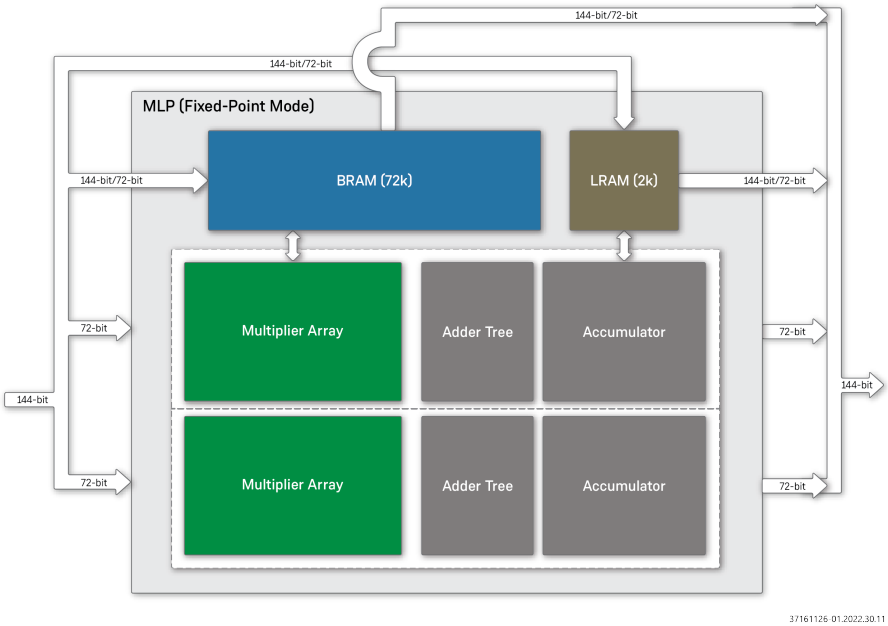

Figure 12 • MLP Using Fixed-Point Mode

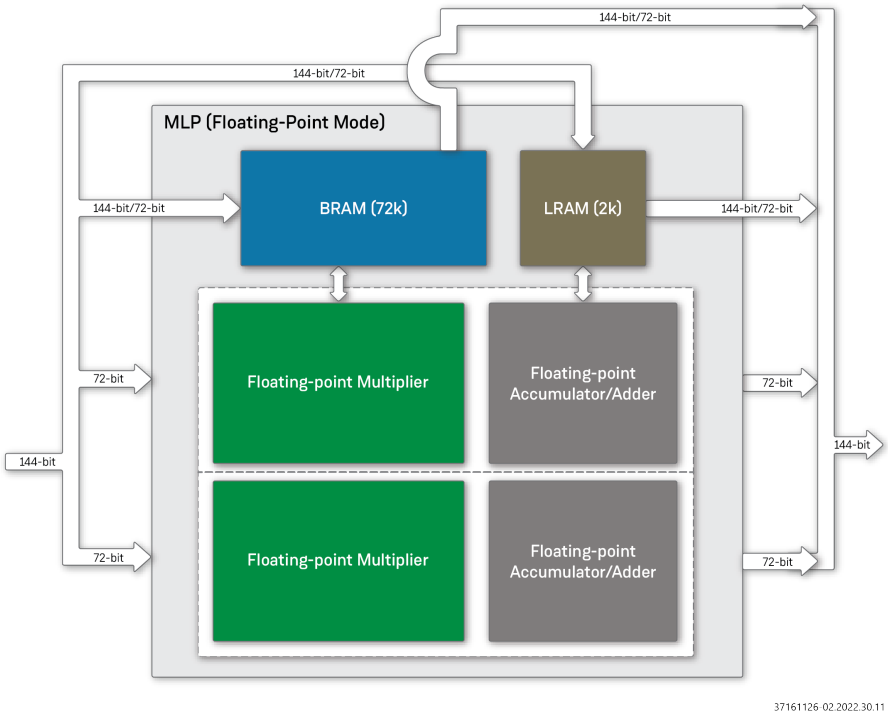

Figure 13 • MLP Using Floating-Point Mode

A powerful feature available in the Achronix MLP is the ability to connect several MLPs with dedicated high-speed cascade paths. The cascade paths allow for the adder tree to extend across multiple MLP blocks in a column without using extra fabric routing resources, and a data cascade/broadcast path is available to send operands across multiple MLP blocks. Cascading input or result data to multiple MLPs in parallel allows for complex, multi-element operations to be performed efficiently without the need for extra routing. The following diagram shows the cascade paths across MLPs:

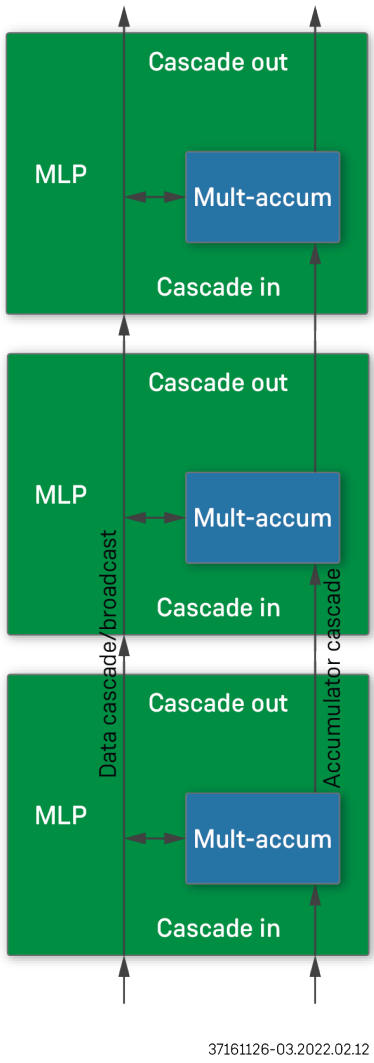

Figure 14 • MLP Cascade Path

Chapter 3 : Speedcore eFPGA IP Interface

Interfaces

There are three sets of interfaces to a Speedcore instance, data, clock and programming interface (see the figure below).

Data Signals

Data inputs and outputs can be on all four edges of a Speedcore instance. There is an option to either register the signals at the Speedcore interface or send them directly to the programmable logic core.

Clock Inputs

Global clock inputs are available on the north and south edges. Interface clock inputs and outputs are available on the north and south edges (one set per cluster column), as well on the east and west edges (one set per cluster row).

> **Caution!**
>
> The global clock inputs may not be available on certain small Speedcore instances.

Programming Interface

There is a dedicated set of signals for programming the eFPGA instance. The number of these signals depends on the programming options selected. The table following the figure lists the interface signals of an eFPGA instance.

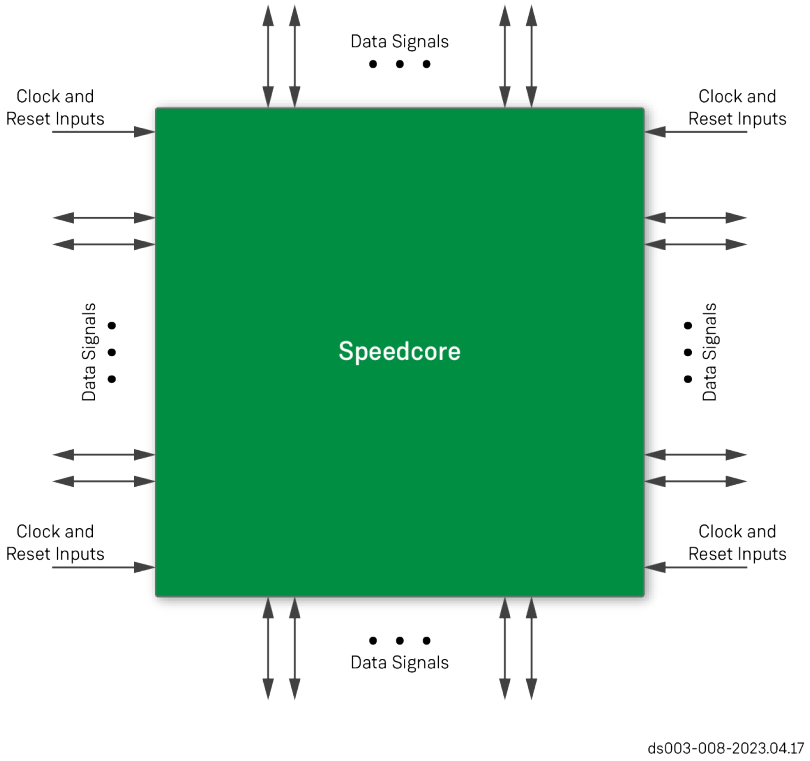

Figure 15 • Speedcore eFPGA Interfaces

Pins

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

Table 5 • Speedcore eFPGA Pins

Pin Name	Direction	Description
`i_user_*_lut/bram/lram/ dsp_*[n:0]`	Input	Data inputs to the programmable core. The bit width depends on size and customer requirements.
`o_user_*_lut/bram/lram/ dsp_*[m:0]`	Output	Data outputs from the programmable core. The bit width depends on size and customer requirements.
`i_user_*_trunk/mt/ branch_*[c:0]`	Input	Clock inputs to the programmable core.
`o_user_*_mt/branch_*[d:0]`	Output	Clock outputs from the programmable core.

Pin Name	Direction	Description
`i_config_*[x:0]`	Input	Bitstream data, control and configuration setting selection pins for a Speedcore instance. The width of these signals depends on the selected programming option.
`o_config_*[y:0]`	Output	Status output and signaling pins for a Speedcore instance.

Chapter 4 : Speedcore eFPGA In-System Debug

Snapshot is the real-time design debugging tool for Achronix FPGAs and eFPGA cores. The Snapshot debugger, which is embedded in ACE software, delivers a practical platform to observe the signals of a user design in real-time. To use the Snapshot debugger, the Snapshot macro must be instantiated inside the user RTL. After instantiating the macro and programming the FPGA, the design may be observed 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, allowing 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 built-in waveform viewer. The following figure shows the components involved in a Snapshot debug session:

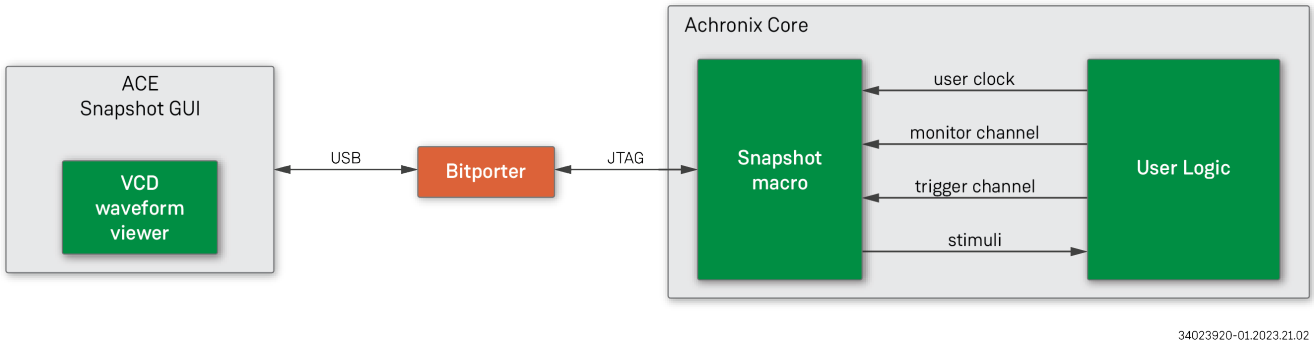

Figure 16 • Snapshot Overview

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-sensitive (rise/fall) or level-sensitive triggers.
- The ACE Snapshot GUI allows the 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 driving 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 prior to the trigger event. 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 eFPGA 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 is 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

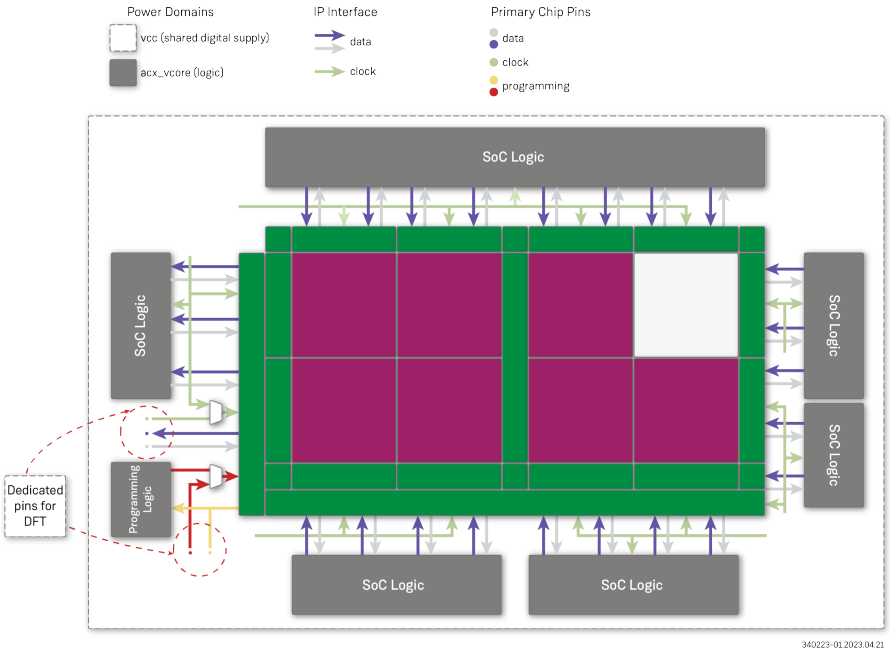

Figure 17 • Sample eFGPA Instantiation

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 are provided in the *Design and Integration Manual*.

Simulation and Validation

The Speedcore eFPGA is supplied with ACE 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 simulation/verification flow.

Chapter 6 : Speedcore eFPGA Datasheet Revision History

Revision History

The following table lists the revision history of this document.

Version	Date	Description
1.0	09 Feb 2019	• Initial release.
1.1	29 Jul 2019	• First public release: removed confidential markings. • Speedcore eFPGA Architecture (page 3): updated the key feature table for the LRAM2k.
2.0	02 Oct 2020	• Updates for new Speedcore devices.
2.0.1	27 Apr 2021	• Document made public (removed confidential marking).
2.1	24 Jan 2023	• Added the following sections: ◦ Local RAM 2k (page 0) ◦ Block RAM 72k (page 9) ◦ Machine Learning Processor (MLP) Block (page 13) • Updated the section Functionality (page 2)
2.2	23 Oct 2024	Various minor updates and corrections.