29th IEEE Symposium on Computer Arithmetic – ARITH2022
Virtual conference, September 12-14, 2022

MiniFloat-NN and ExSdotp: An ISA Extension and a Modular Open Hardware Unit for Low-Precision Training on RISC-V Cores

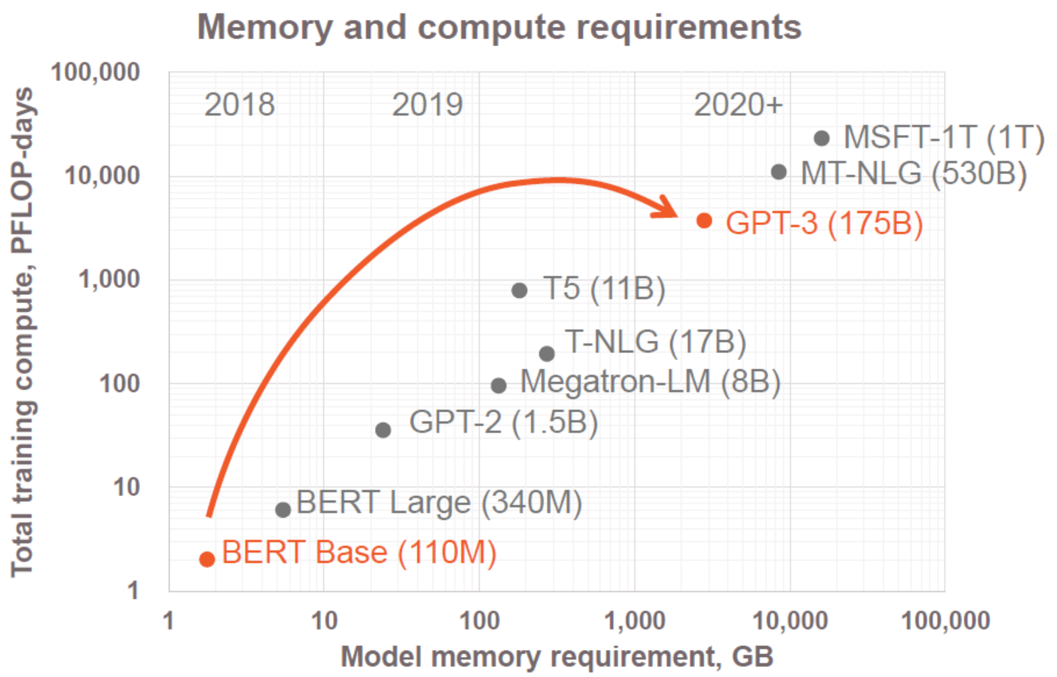
Luca Bertaccini[*], Gianna Paulin[*], Tim Fischer[*], Stefan Mach[†], Luca Benini[*‡]

[*]IIS, ETH Zurich, Switzerland,
[‡]DEI, University of Bologna, Italy
[†]Axelera AI, Switzerland

PULP Platform
Open Source Hardware, the way it should be!

@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

The Exponential Growth of AI



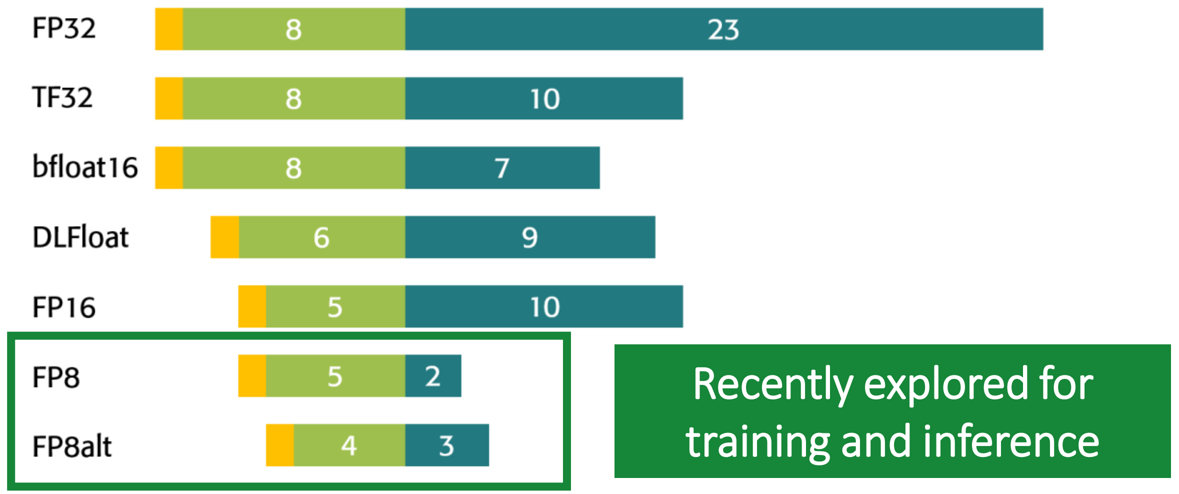
S. Lie, "Thinking outside the die: Architecting the ML accelerator of the future"

- NN models' memory and compute requirements are growing **exponentially**
- Technology scaling is not sufficient

- Required **algorithmic** and **architectural** advancements to keep the progress pace

Algorithmic Advancements

- New low-precision data types:
  - 32-bit → 19-bit → 16-bit → 8-bit floating-point (FP) data types
- New **mixed** and **low-precision training** algorithms have been developed to exploit the resilience of NN models to noise
- Expanding **operations** in which the accumulation is performed in higher precision

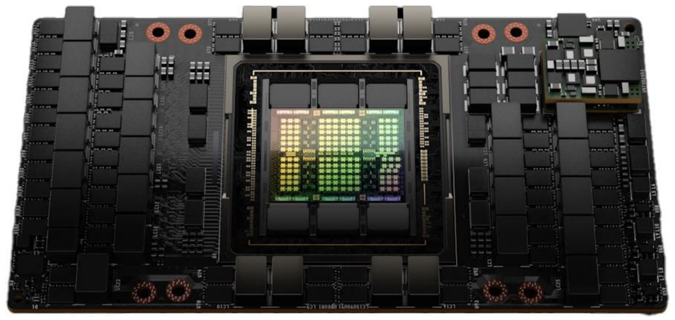


- Lower memory requirements
- Opportunities for more efficient hardware architectures

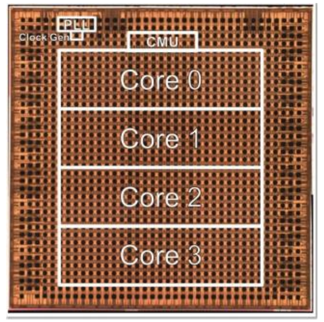
Architecture Advancements

Hardware Specialization

FUJITSU A64FX
(https://www.fujitsu.com)



NVIDIA H100 on SMX5 Module
(https://resources.nvidia.com/en-us-tensor-core)



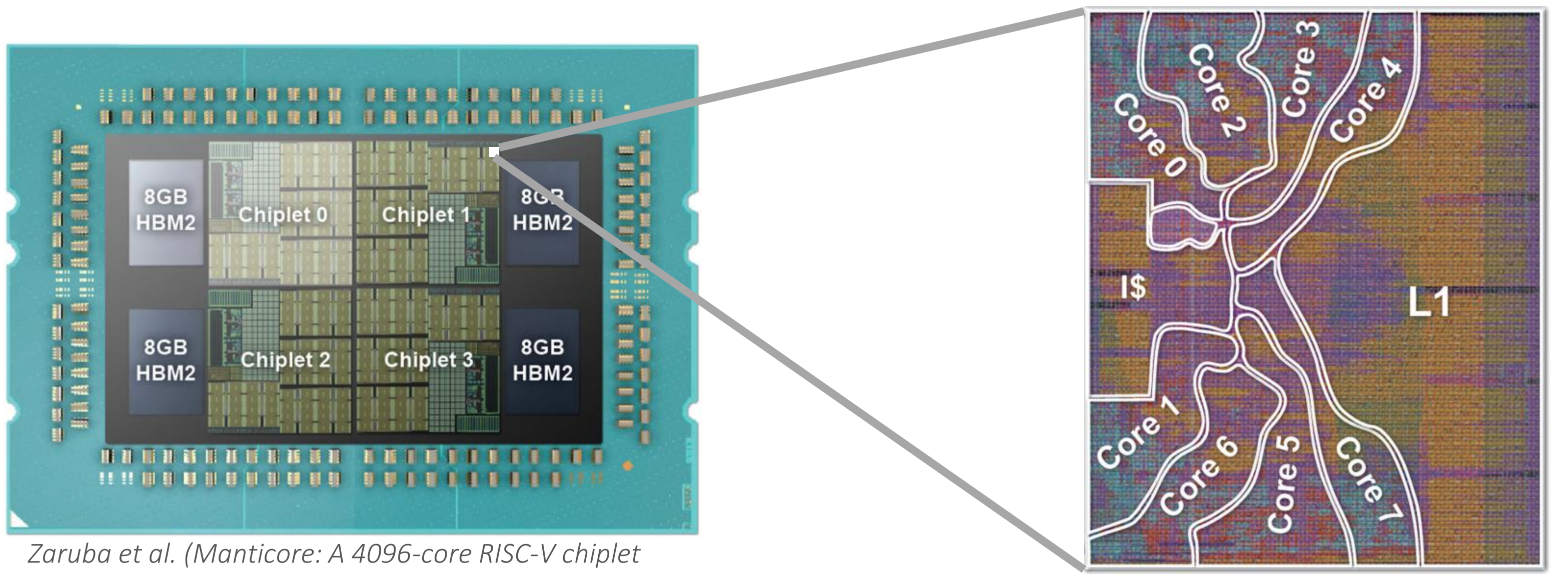
IBM AI Chip
(https://research.ibm.com)

- HPC processor with vector extensions
- 512-bit SIMD FPUs

- GPU architecture based on CUDA cores and tensor cores (each one capable of 1024 dot products per cycle)
- Specialized on data parallel computations

- Fully specialized for NN training and inference
- Based on arrays of mixed precision elements

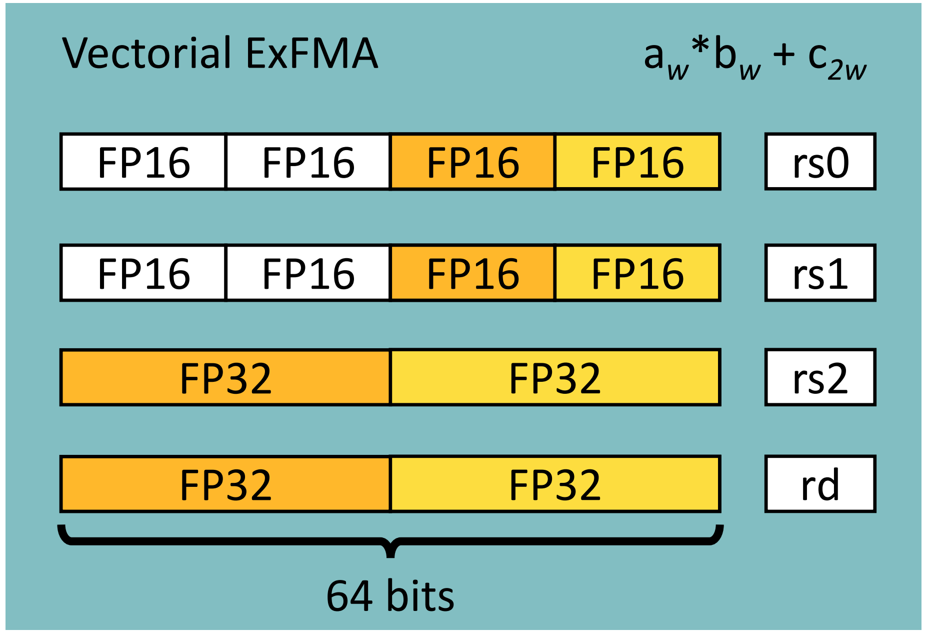
Academic Architecture for HPC - Manticore

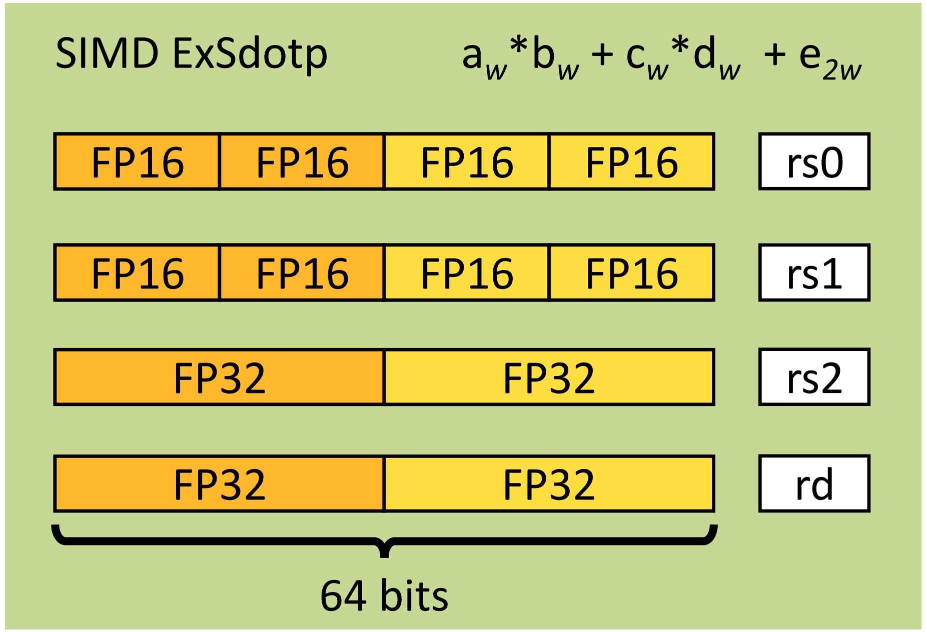


Zaruba et al. (Manticore: A 4096-core RISC-V chiplet architecture for ultraefficient floating-point computing)

- Manticore: a chiplet-based hierarchically-scalable architecture
- Built upon the replication of eight-core **compute clusters:**
  - Area-optimized integer cores coupled with FP64/FP32 FPUs sharing a scratchpad memory.
  - It contains Snitch cores, enhanced with ISA extensions that allows achieving >90% FPU utilization.
- How can we extend such an architecture to exploit new algorithms for low and mixed-precision efficient NN training?

Why are we interested in ExSdotp instructions?

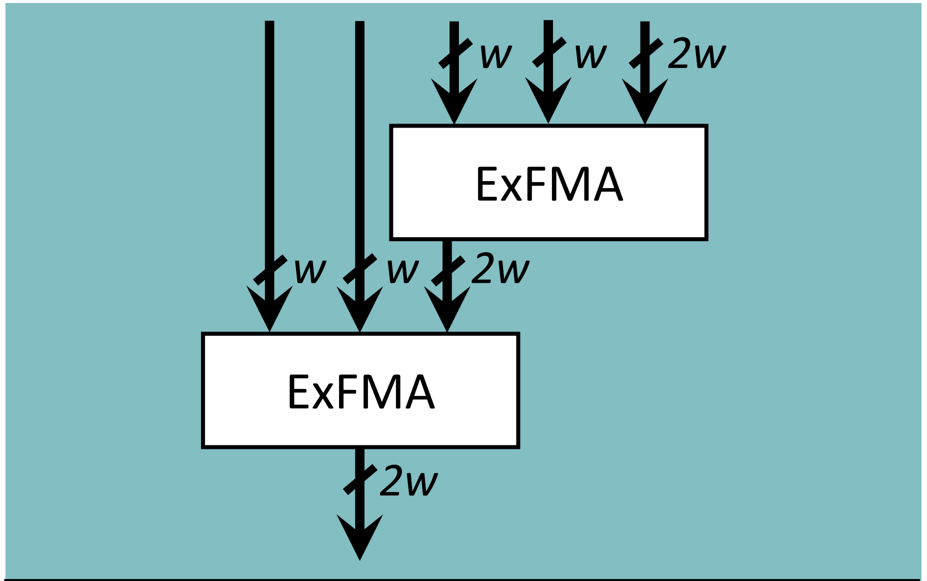




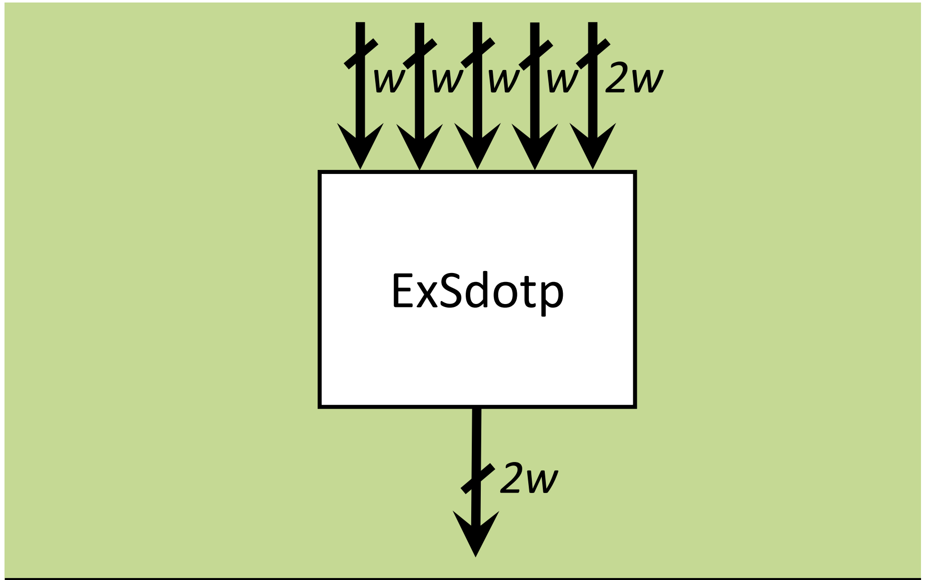
- Vectorial Expanding FMA: **Unbalanced**
- Consumes half of rs0, rs1 and the whole rs2, rd
- Multiple instructions to cover all possible source locations needed to use the full register file space

- Vectorial Expanding SDOTP: **Balanced**
- Consumes the whole rs0, rs1, rs2, rd

Why Fused ExSdotp Units?



$\text{Cascade}_{2w} = a_w * b_w + (c_w * d_w + e_{2w})$



$\text{ExSdotp}_{2w} = a_w * b_w + c_w * d_w + e_{2w}$

- A cascade of two ExFMAs computes an expanding dot product
- Non-distributive FP addition

- Fused ExSdotp unit
- Single normalization and rounding step

Contributions

1. ExSdotp unit:
   1. An **open-source** parameterized **multi-format unit** supporting **expanding sum-of-dot-product** (ExSdotp) instructions (8-to-16-bit and 16-to-32-bit), as well as non-expanding and expanding **three-operand additions**, called Vsum and ExVsum.
   2. **Integration** of ExSdotp unit into an open-source multi-format FPU (**FPnew**[1])

2. MiniFloat-NN:
   1. A **RISC-V ISA extension** for **low-precision FP training** on many-core architectures.
   2. **Integration** of the enhanced FPU into an open-source RISC-V eight-core **cluster** based on **Snitch**[2] cores.

3. Evaluation of the standalone ExSdotp unit and the enhanced compute cluster

[1]https://github.com/pulp-platform/fpnew [2]https://github.com/pulp-platform/snitch

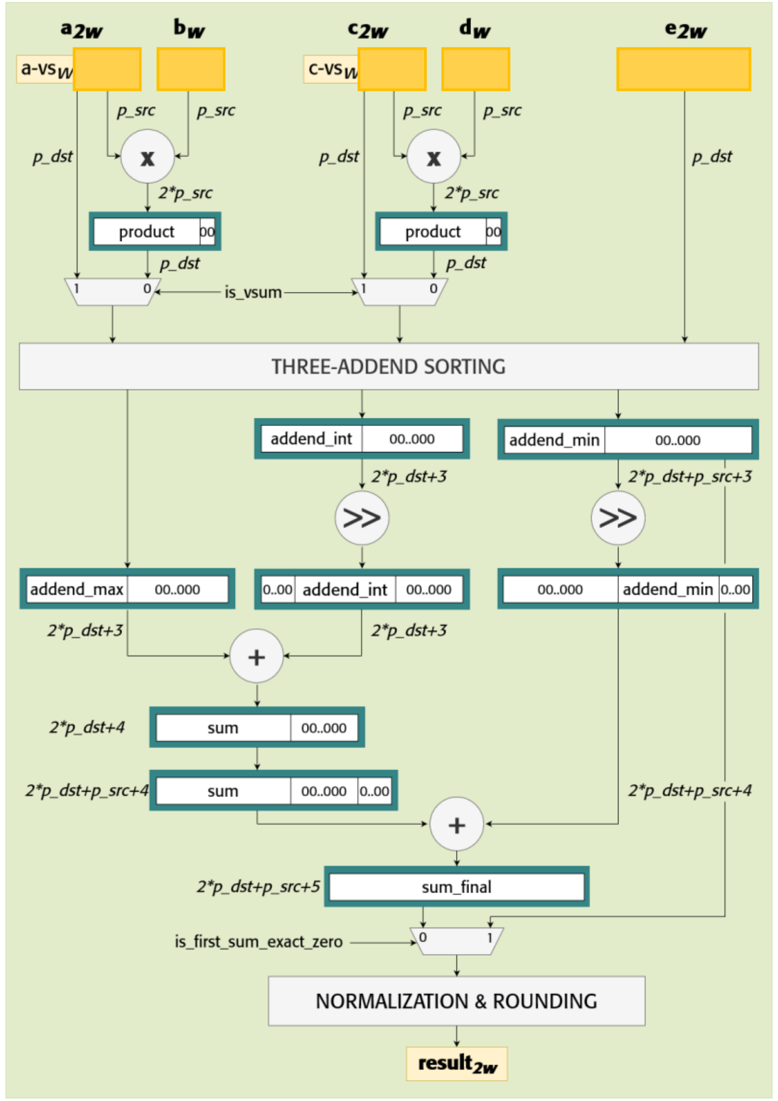
Targeted Floating-Point Formats

- A parametric design to enable fast exploration of new FP formats
- ExSdotp source formats:
  - FP16alt (1, 8, 7) (as bfloat16 but handling subnormals)
  - FP16 (1, 5, 10)
  - FP8alt (1, 5, 2)
  - FP8 (1, 4, 3)
- ExSdotp destination formats:
  - FP32 (1, 8, 23)
  - FP16alt (1, 8, 7)
  - FP16 (1, 5, 10)
- Subnormals handled for all combinations of formats

	Destination Format				
Source Format	**FP32**	**FP16alt**	**FP16**	**FP8alt**	**FP8**
FP32	Vsum	-	-	-	-
FP16alt	ExSdotp/ExVsum	Vsum	Vsum	-	-
FP16	ExSdotp/ExVsum	Vsum	Vsum	-	-
FP8alt	-	ExSdotp/ExVsum	ExSdotp/ExVsum	Vsum	Vsum
FP8	-	ExSdotp/ExVsum	ExSdotp/ExVsum	Vsum	Vsum

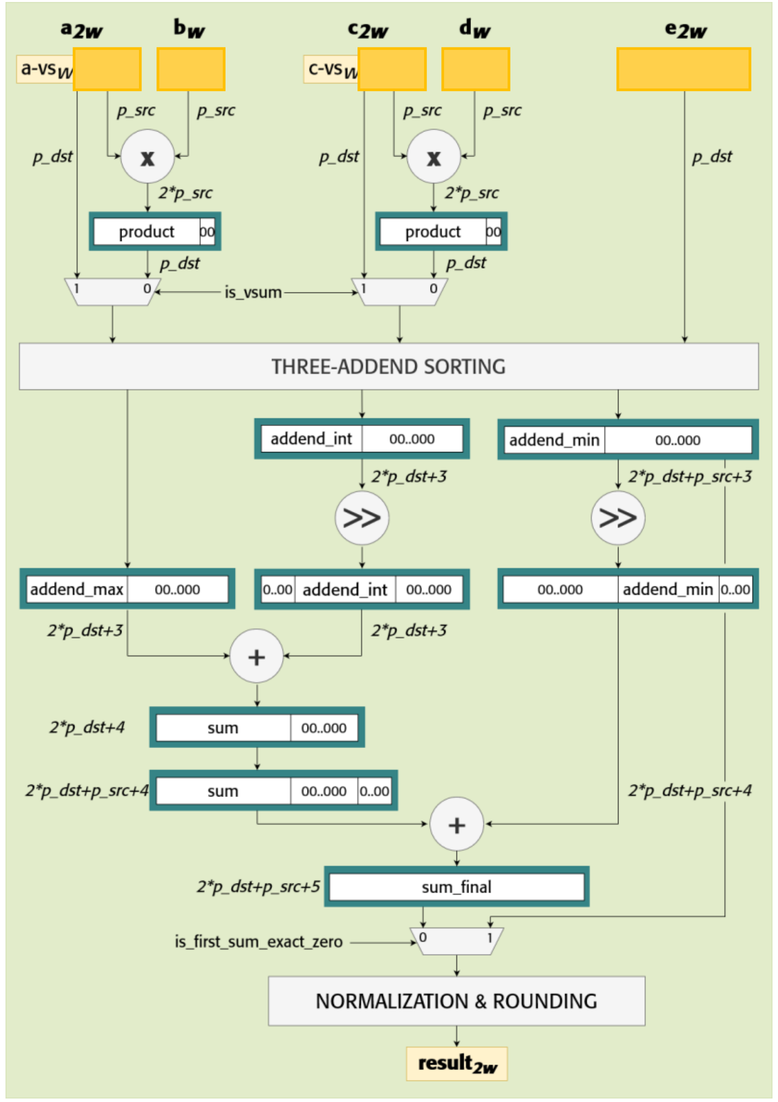
Expanding Sum of Dot Product Unit

- *w = source format bit width*
- *2w = destination format bit width*
- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$



Expanding Sum of Dot Products Unit

- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$

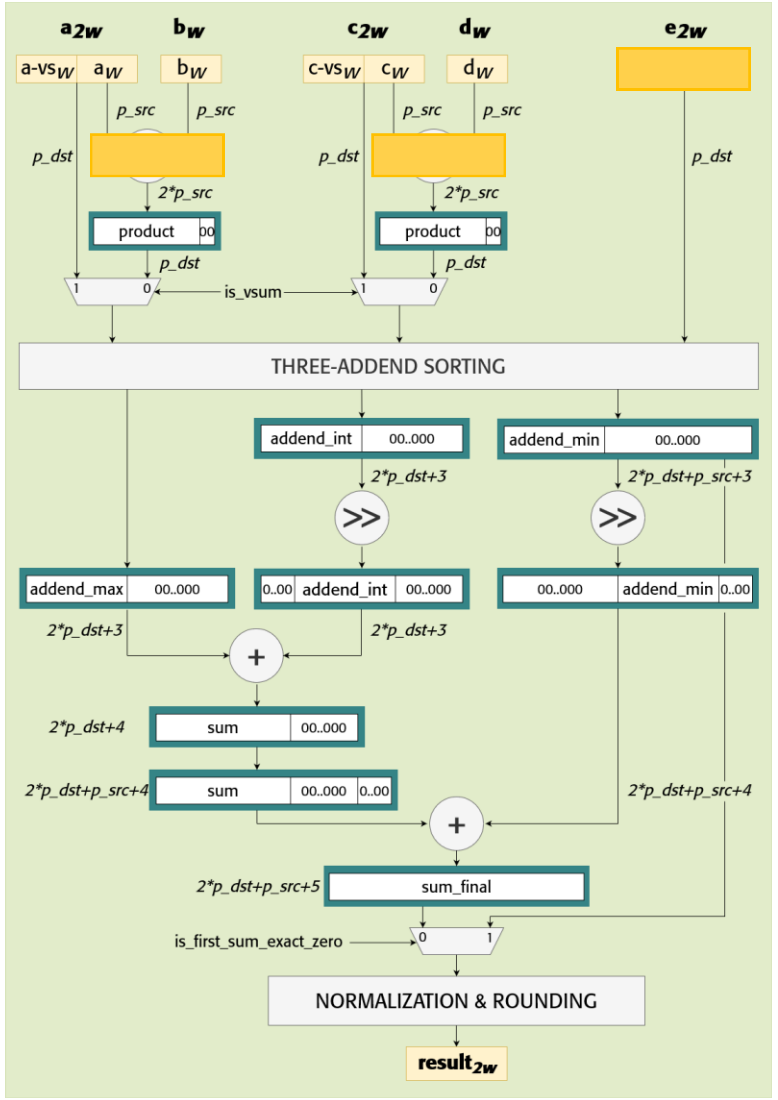


Expanding Sum of Dot Products Unit

- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$

- Product expressed with twice as many bits $= 2*p\_src$ ($\approx p\_dst$)

- Padding to reach *p_dst*

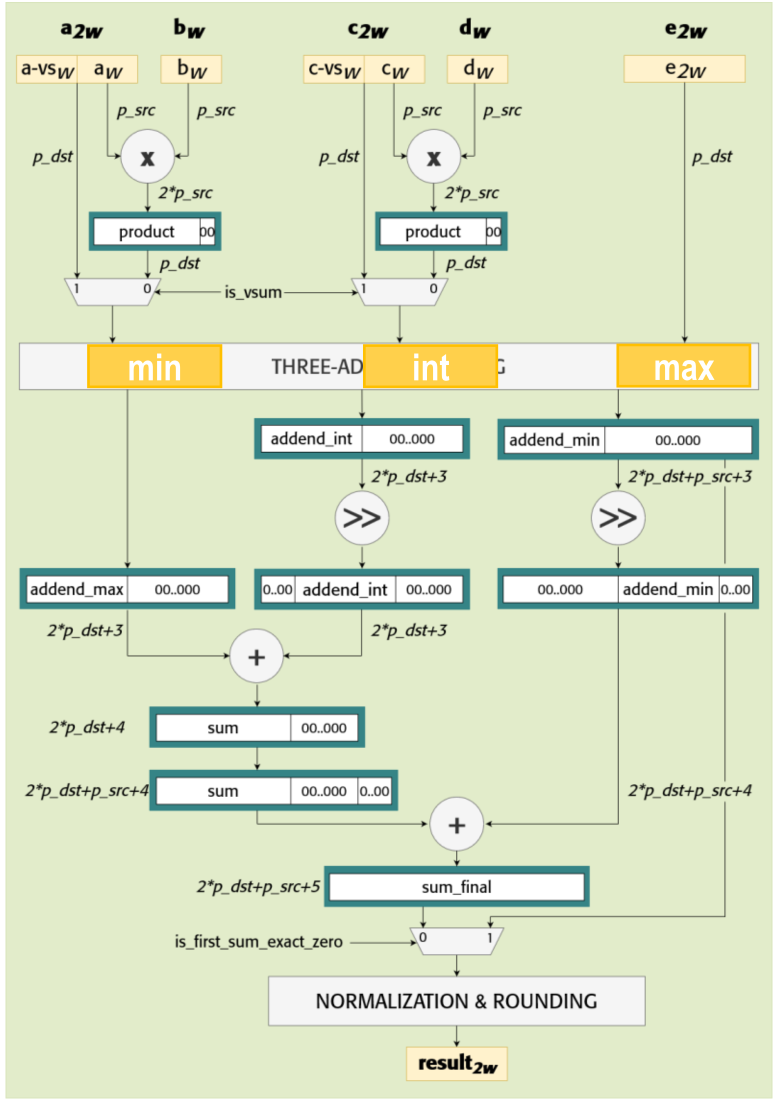


Expanding Sum of Dot Products Unit

- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$

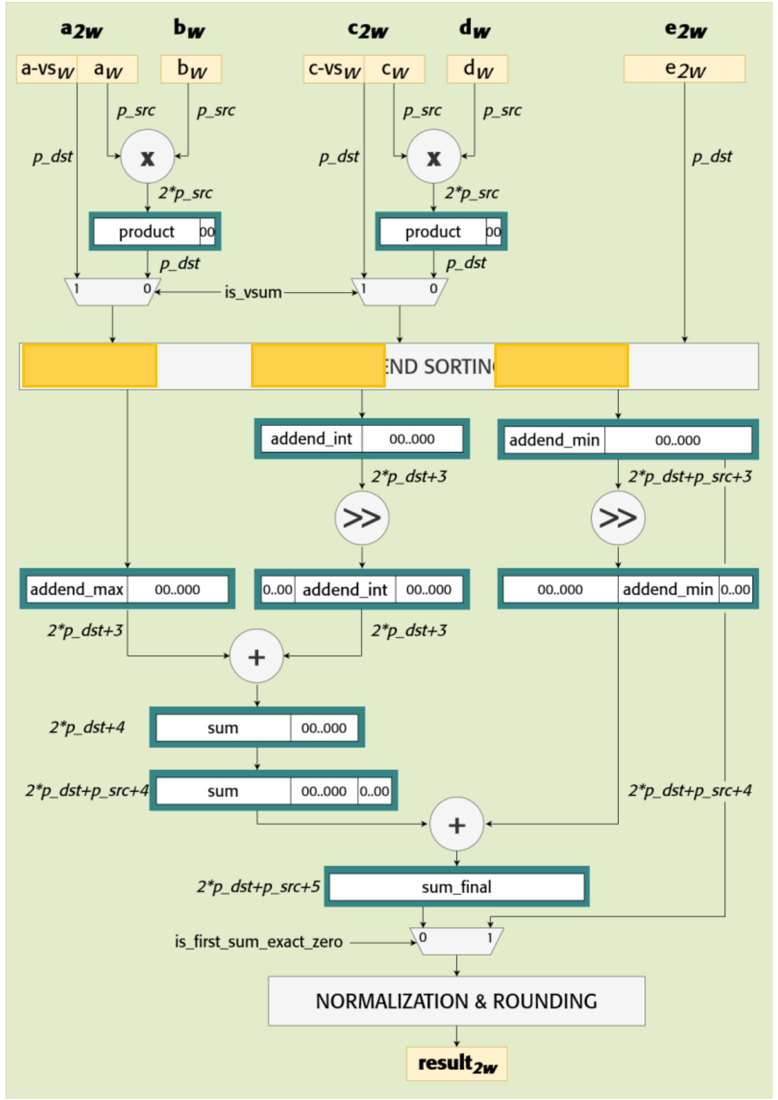
- Three-addend sorting to prevent precision losses due to **cancellation** during the three-term addition:
  - (a + b) – a might return 0 if a is much larger than b
  - Decreasing order (abs value) -> (a – a) + b = b

- Gradually **increasing** the **internal precision** to retain precision



Expanding Sum of Dot Products Unit

- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$
- Three-addend sorting to prevent precision losses due to **cancellation** during the three-term addition:
  - (a + b) – a might return 0 if a is much larger than b
  - Decreasing order (abs value) -> (a – a) + b = b
- Gradually **increasing** the **internal precision** to retain precision

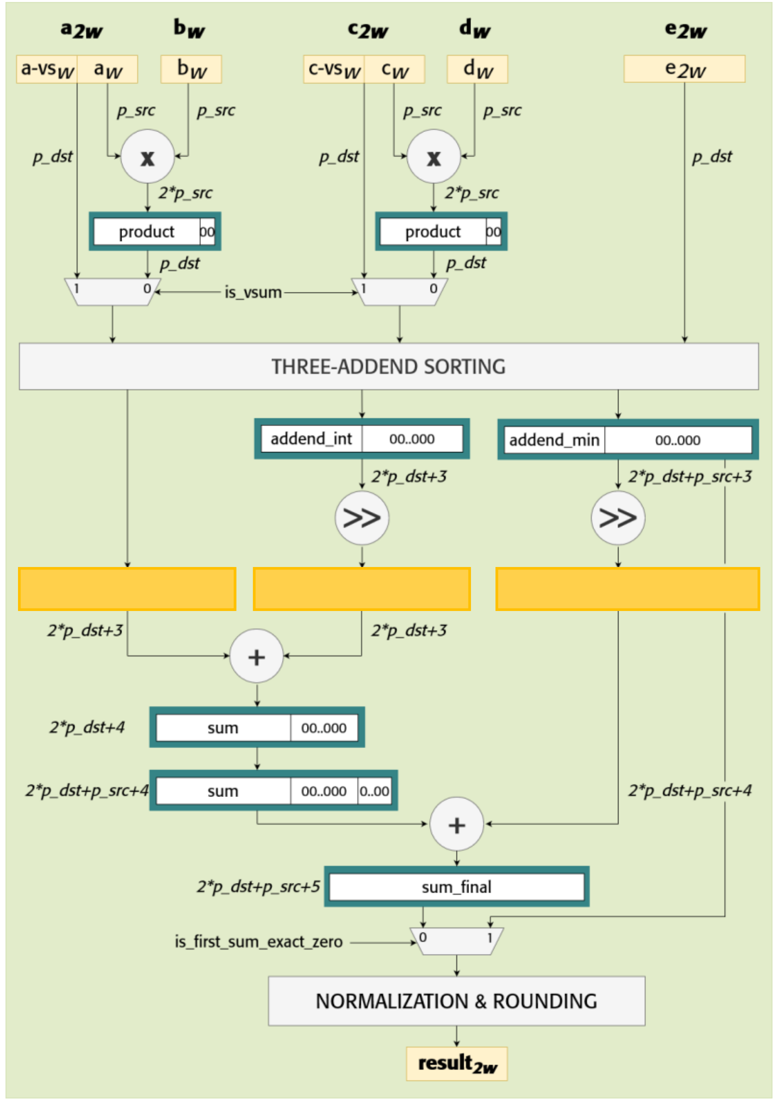


Expanding Sum of Dot Products Unit

- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$

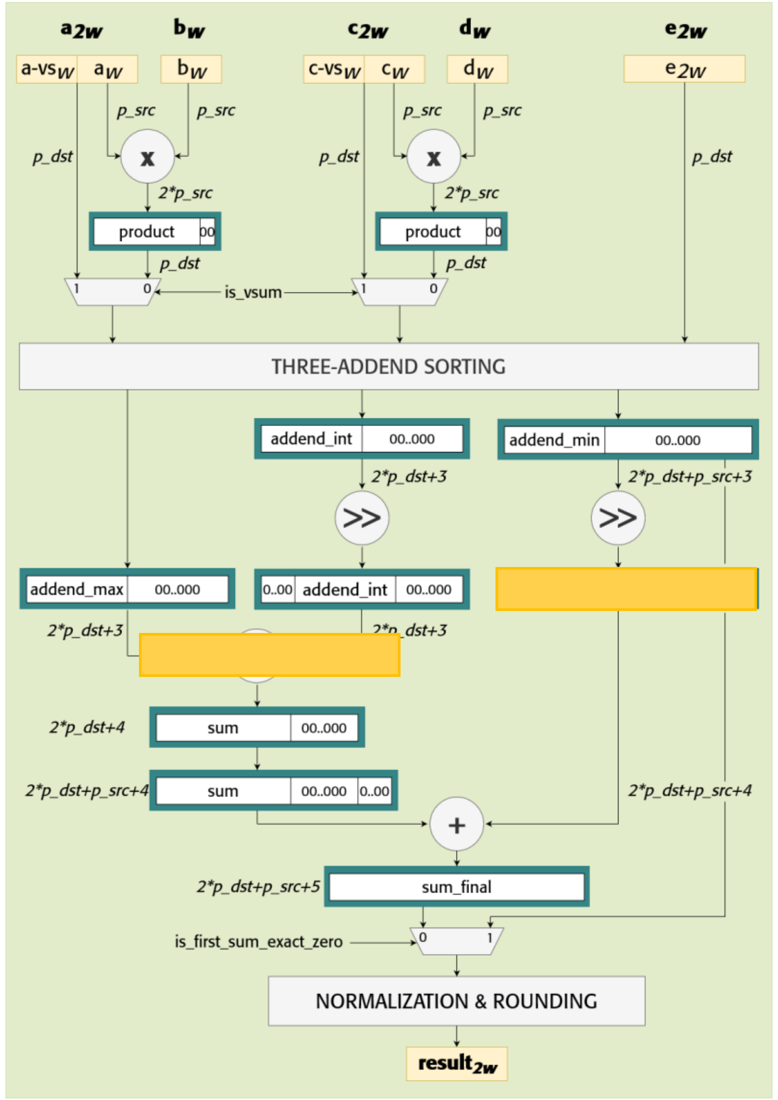
- Three-addend sorting to prevent precision losses due to **cancellation** during the three-term addition:
  - (a + b) – a might return 0 if a is much larger than b
  - Decreasing order (abs value) -> (a – a) + b = b

- Gradually **increasing** the **internal precision** to retain precision



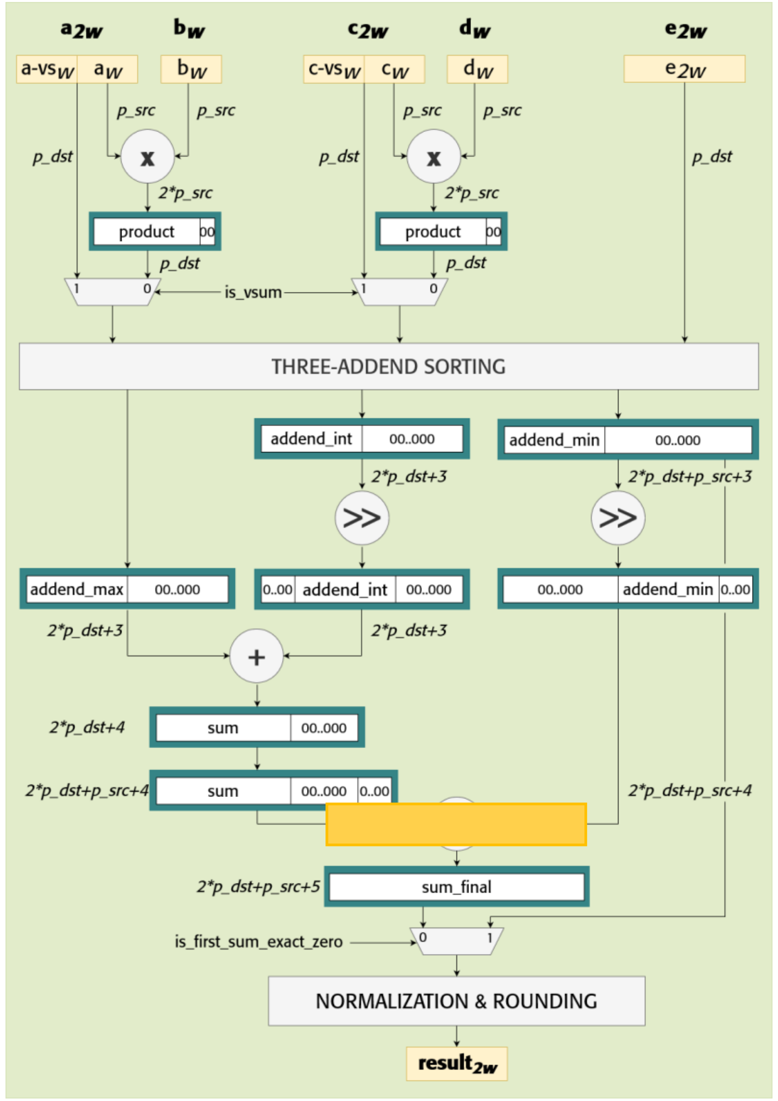
Expanding Sum of Dot Products Unit

- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$
- Three-addend sorting to prevent precision losses due to **cancellation** during the three-term addition:
  - (a + b) – a might return 0 if a is much larger than b
  - Decreasing order (abs value) -> (a – a) + b = b
- Gradually **increasing** the **internal precision** to retain precision



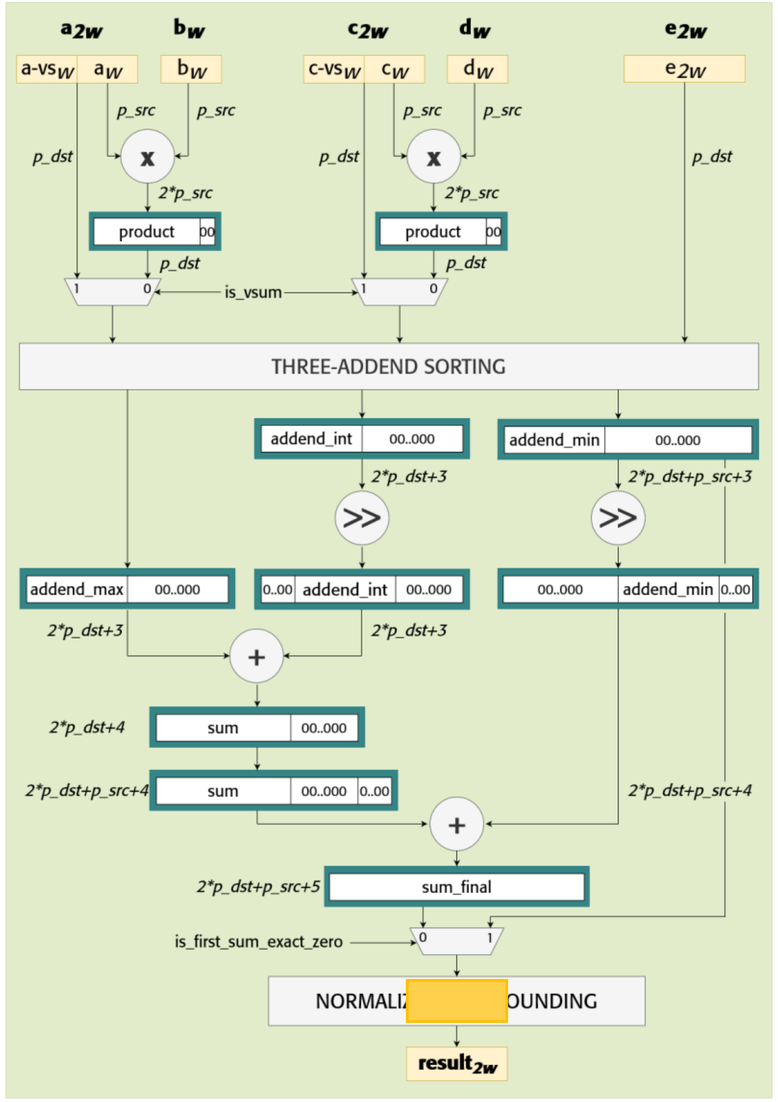
Expanding Sum of Dot Products Unit

- ExSdotp $= a_w * b_w + c_w * d_w + e_{2w}$
- Three-addend sorting to prevent precision losses due to **cancellation** during the three-term addition:
  - (a + b) – a might return 0 if a is much larger than b
  - Decreasing order (abs value) -> (a – a) + b = b
- Gradually **increasing** the **internal precision** to retain precision



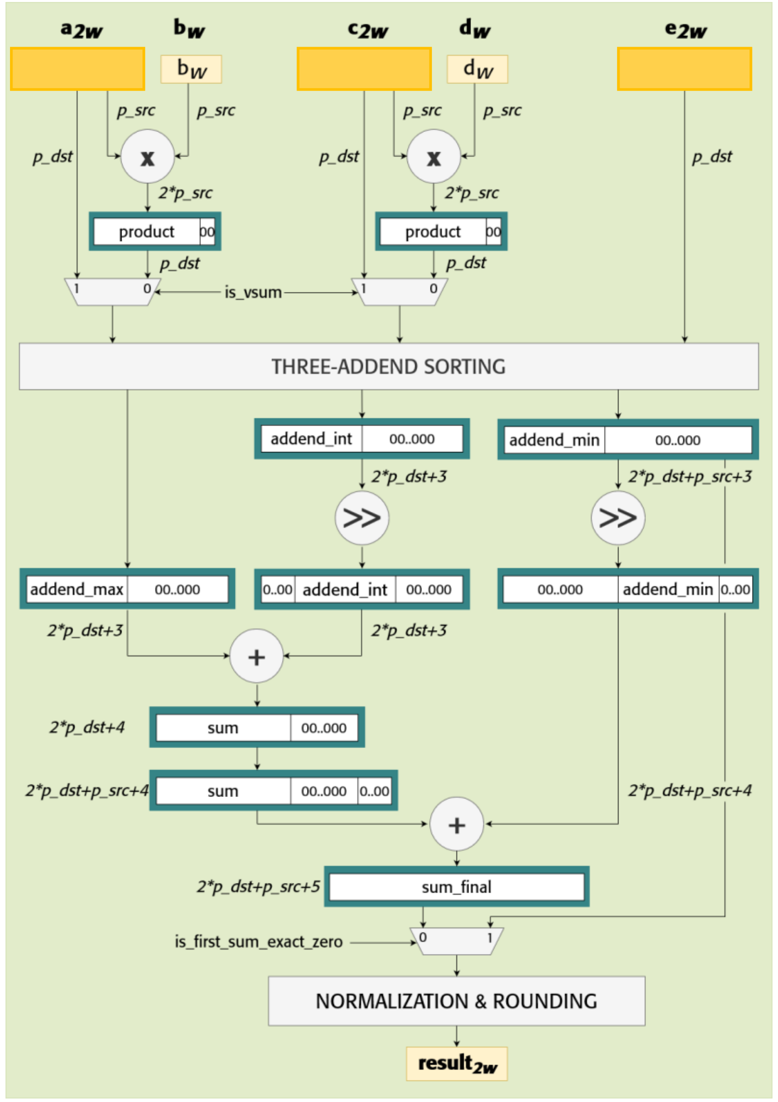
Expanding Sum of Dot Products Unit

- ExSdotp = $a_w*b_w + c_w*d_w + e_{2w}$
- Three-addend sorting to prevent precision losses due to **cancellation** during the three-term addition:
  - (a + b) – a might return 0 if a is much larger than b
  - Decreasing order (abs value) -> (a – a) + b = b
- Gradually **increasing** the **internal precision** to retain precision



Non-expanding three-term addition

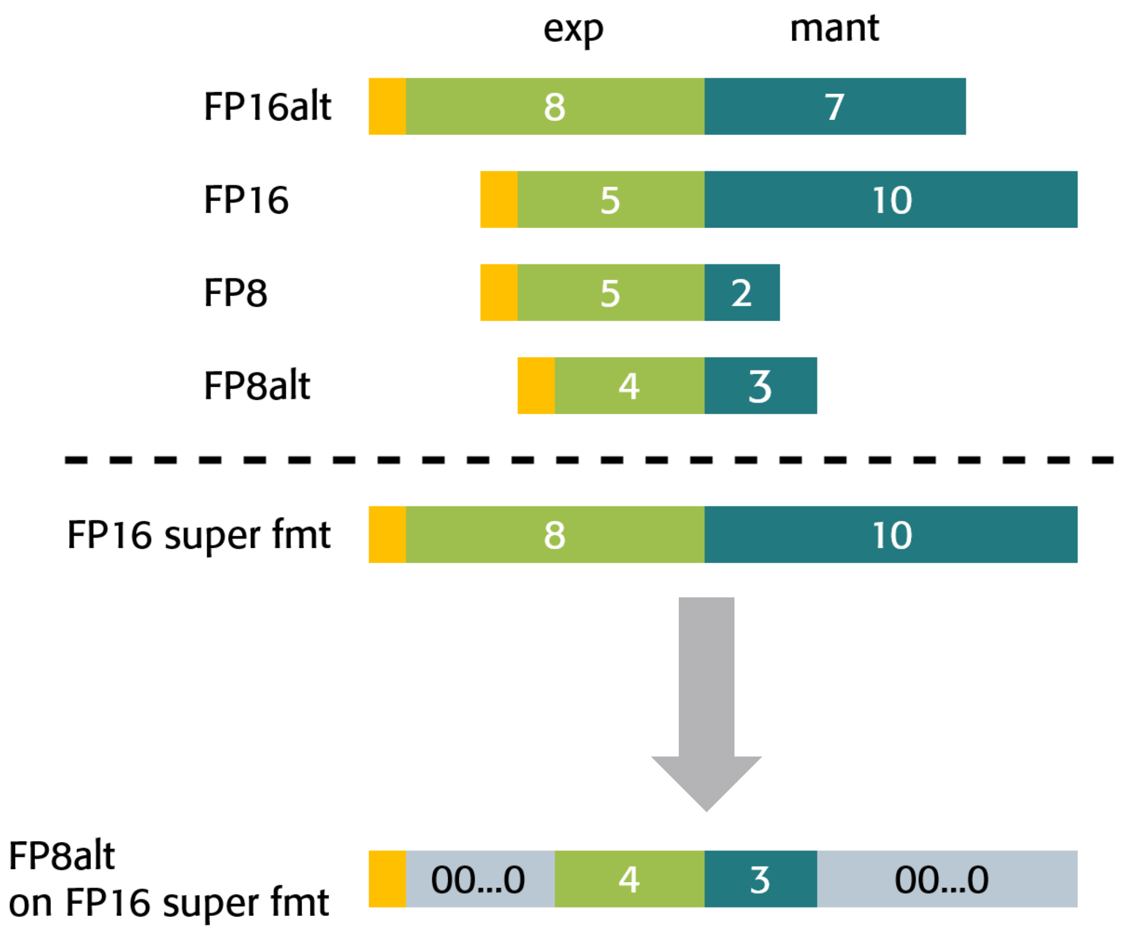
- Vsum = $a_{2w} + c_{2w} + e_{2w}$
- Vsum can be used to reduce and accumulate the results packed in a register after SIMD ExSdotp executions
- Support for non-expanding three-term sum added by **bypassing** the multiplications
- All the necessary logic is already present as the targeted ExSdotp operations were expanding
- ExVsum is computed as an ExSdotp where $b_w = d_w = 1$
  ExVsum = $a_w*1 + c_w*1 + e_{2w}$



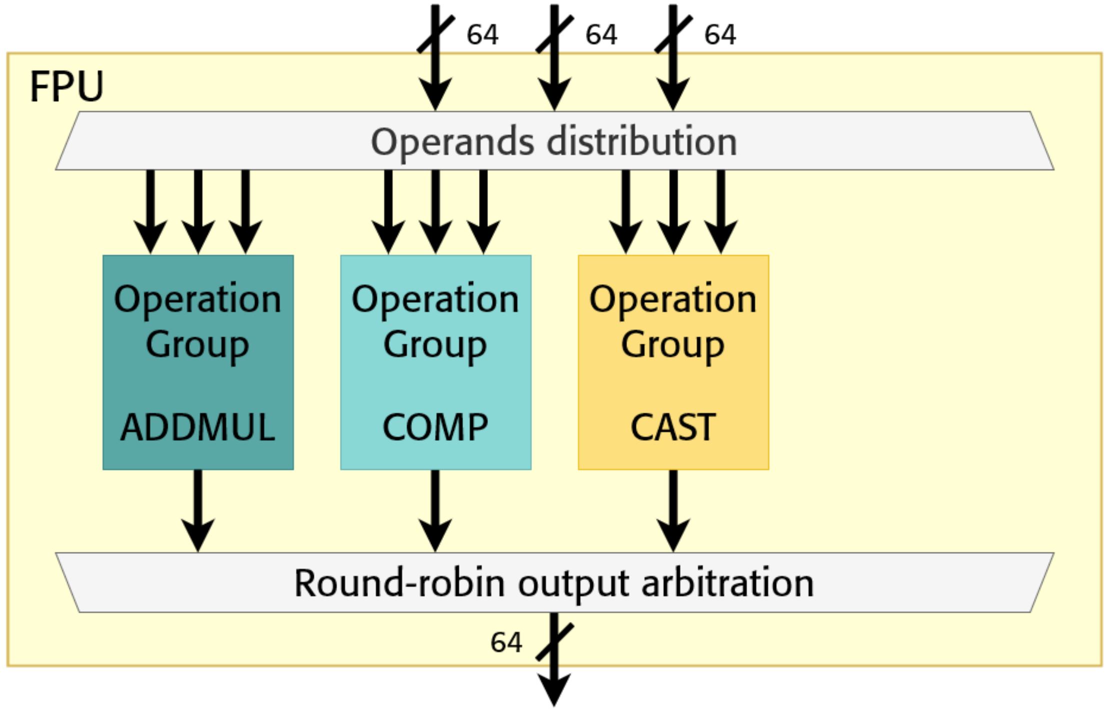
Reusing the Same Hardware for Lower-Precision Formats

- 16-to-32-bit ExSdotp unit:
  - **Super format** between enabled input formats FP16, FP16alt, FP8, FP8alt
  - {1, max(exp_{16}, $\text{exp}_{16\text{alt}}$), max($\text{mant}_{16}$, $\text{mant}_{16\text{alt}}$)}

- Narrower exponent mapped to the lower bits of super format exponent
- Narrower mantissa mapped to the upper bits of super format mantissa

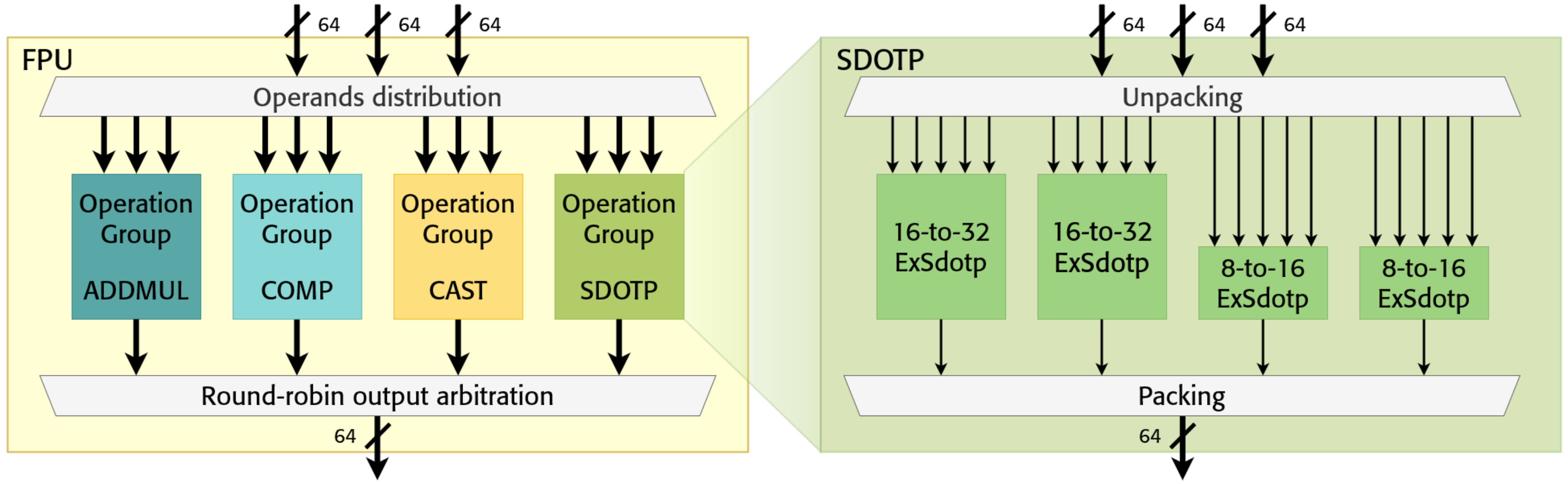


Enhancing FPnew with SIMD ExSdotp



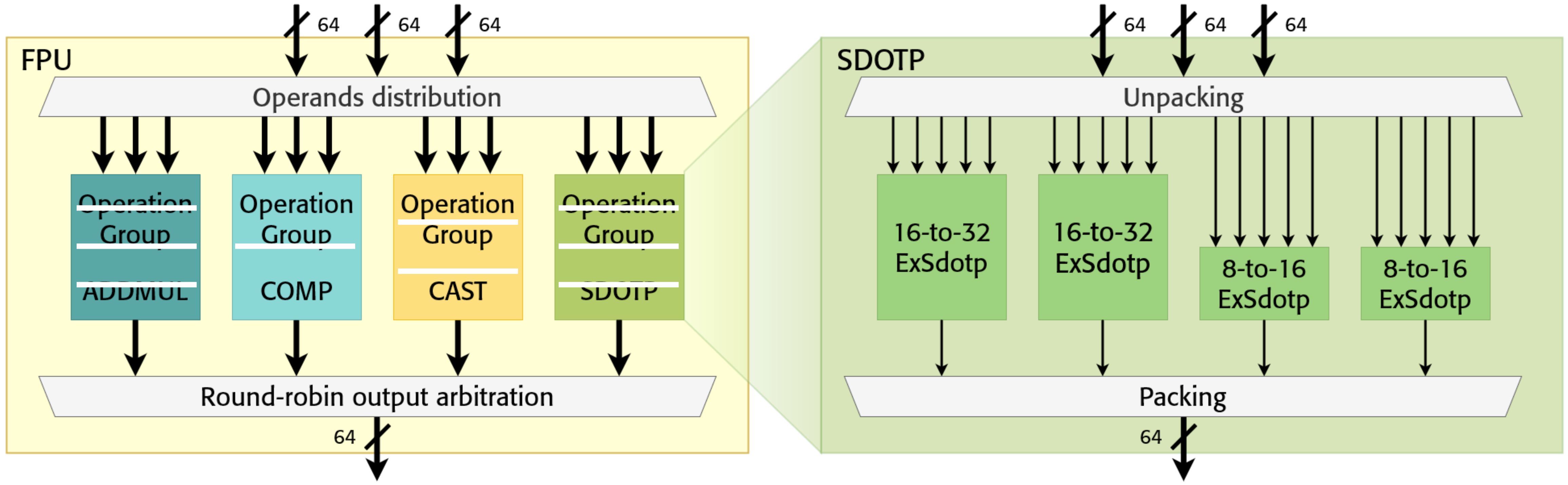
- **FPnew** is a highly-parameterized open-source **modular** energy-efficient multi-format FPU

Enhancing FPnew with SIMD ExSdotp



- **FPnew** is a highly-parameterized open-source **modular** energy-efficient multi-format FPU
- **SIMD ExSdotp** unit integrated into FPnew as a new **operation group** block
- SIMD SDOTP: **two** 16-to-32-bit units and **two** 8-to-16-bit units
- Up to **two** 16-to-32-bit ExSdotp and **four** 8-to-16-bit **ExSdotp per cycle**

Enhancing FPnew with SIMD ExSdotp

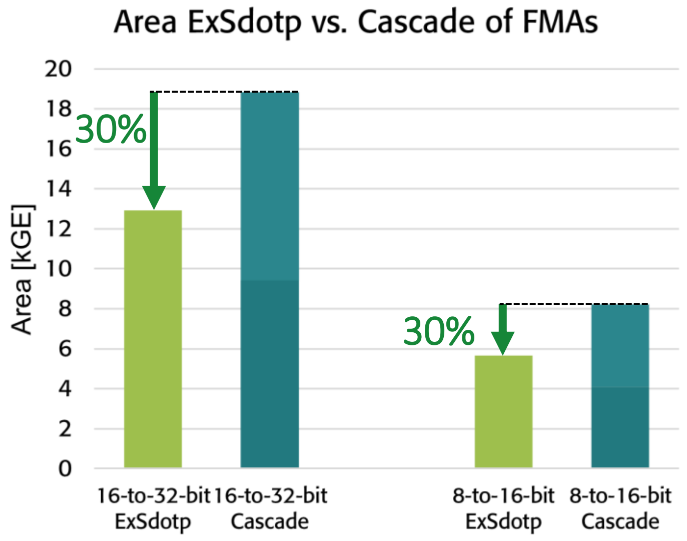


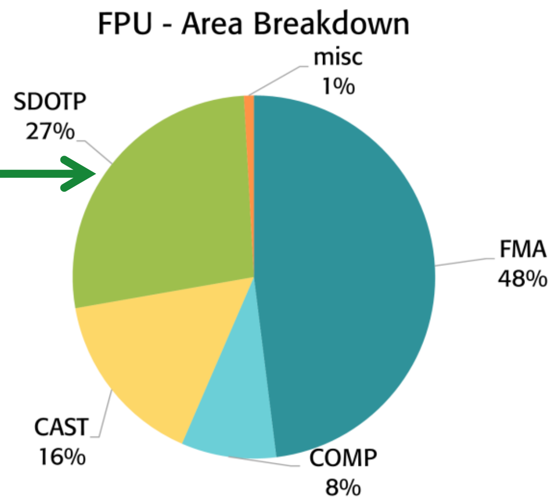
- **Parametrizable number** of **pipeline** levels. In our specific case, we selected:
  - SDOTP: 3 levels of pipeline registers
  - ADDMUL: 3 levels of pipeline registers
  - CAST: 2 levels of pipeline registers
  - COMP: 1 levels of pipeline registers

ExSdotp & FPnew: Area and Timing

- Fused synthesis and PnR with SYNOPSYS FUSION COMPILER using GlobalFoundries 12nm FinFET technology.
- Worst-case corner (0.72V, 125°C)

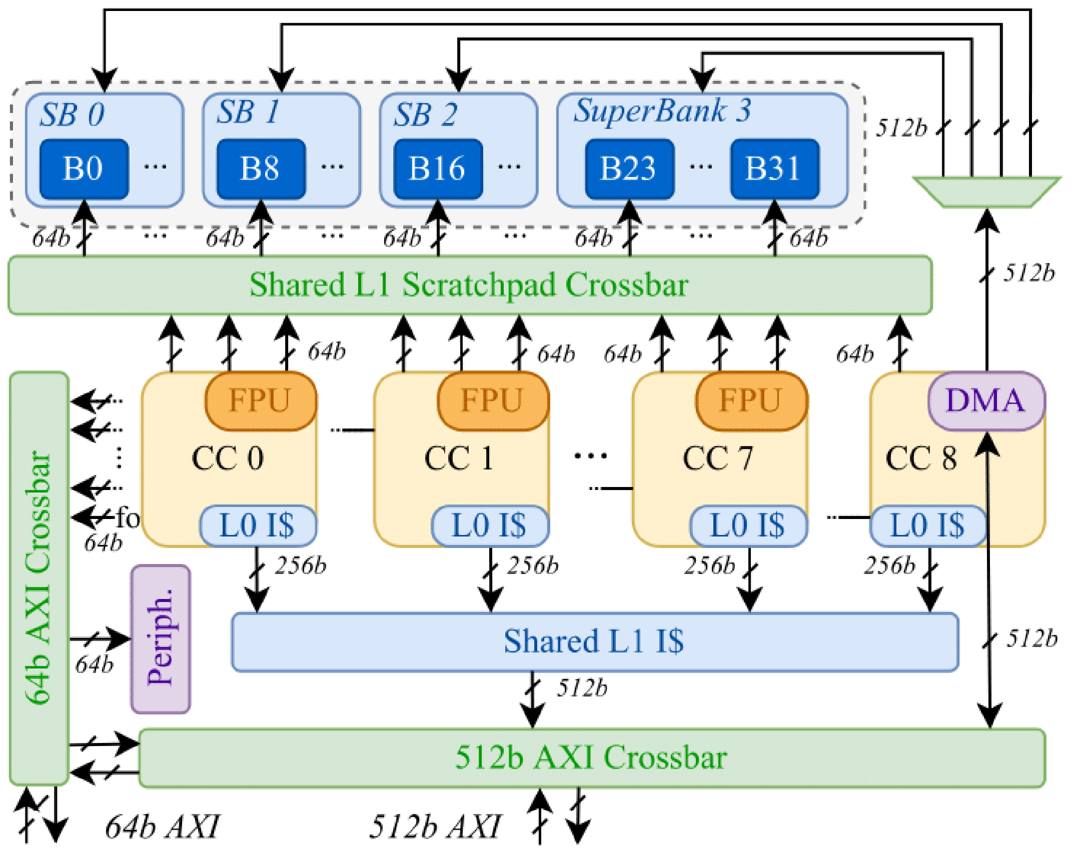
- The fused ExSdotp unit allows for around 30% **area** and **critical path reduction** with respect to a cascade of ExFMA modules.
- The SIMD SDOTP unit occupies 44.5 kGE, amounting to 27% of the enhanced FPU area (overall FPU area = 165kGE). Synthesized targeting **950MHz** in worst-case corner.





MiniFloat-NN Compute Cluster

- Snitch cluster:
  - Integer cores optimized for area and coupled with a large and high-performance FPU, plus ISA extensions to achieve >90% FPU utilization.
- MiniFloat-NN capabilities added by:
  - Including FPnew enhanced with **SIMD ExSdotp**
  - Extending the **decoder**
  - **Alt formats** enabled through a **CSR** to reduce the number of additional instructions

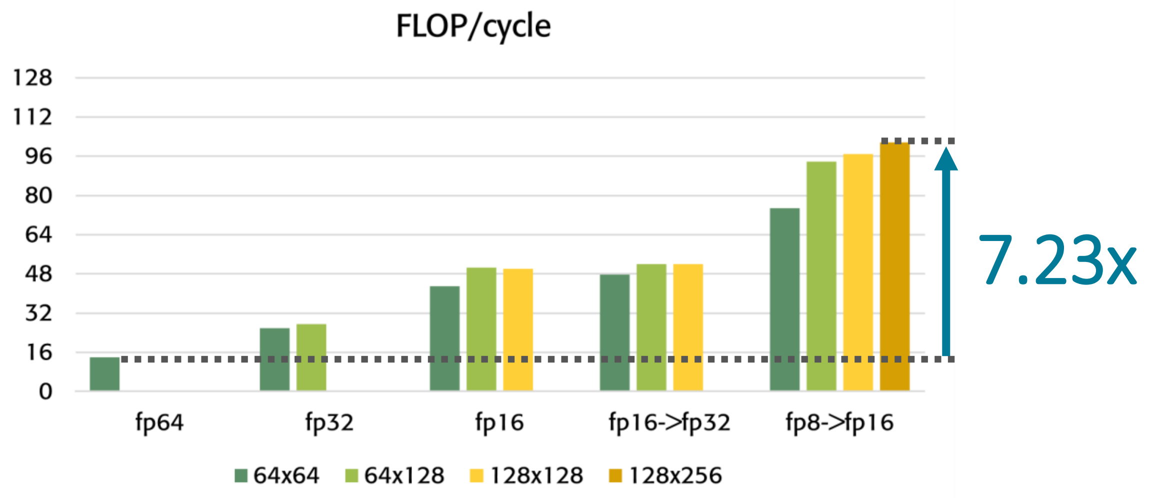


MiniFloat-NN Cluster: Performance

- Only GEMM sizes for which all the data can fit in the local 128 KiB scratchpad memory.
- alt-formats would only require an additional CSR write

GEMM size	FMA-based			ExSdotp-based	
	FP64 [cycles]	FP32 [cycles]	FP16 [cycles]	FP16 to FP32 [cycles]	FP8 to FP16 [cycles]
64 × 64	37.3 k	20.2 k	12.2 k	11.0 k	7.0 k
64 × 128	-	38.0 k	20.7 k	20.2 k	11.1 k
128 × 128	-	-	83.9 k	80.7 k	43.2 k
128 × 256	-	-	-	-	82.5 k

- Up to 7.23x performance increase wrt FP64 thanks to SIMD ExSdotp and low-precision formats



Energy Efficiency, Accuracy, SoA Comparison

Design	Tech	Voltage	Frequency	Area	**Performance** [FLOP/cycle] FP16alt	FP16	FP8	FP8alt	**Peak Throughput** [GFLOPS]	**Efficiency** [GFLOPS/W]
ExSdotp FPU	**12 nm**	**0.8 V**	**1.26 GHz**	**0.019 mm^2**	**8/8**	**8/8**	**16/16**	**16/16**	**20.2 (exFP8)**	**1631 (exFP8)**
Fpnew [1]	22 nm	0.8 V	0.923 GHz	0.049 mm^2	4/8	4/8	8/16	-/-	14.8 (FP8)	1245 (FP8)
Mao *et al.* [2]	28 nm	1.0 V	1.43 GHz	0.013 mm^2	-/-	-/20	-/-	-/-	28.6 (FP16)	975 (FP16)
Zhang *et al.* [3]	90 nm	1.0 V	0.667 GHz	0.191 mm^2	-/-	8/8	-/-	-/-	5.3 (FP16)	113 (FP16)
MiniFloat-NN Snitch	**12 nm**	**0.8 V**	**1.26 GHz**	**0.52 mm^2**	**8/8**	**8/8**	**16/16**	**16/16**	**160 (exFP8)**	**575 (exFP8)**
Snitch [4]	22 nm	0.8 V	1 GHz	0.66 mm^2	-/-	-/-	-/-	-/-	16 (FP64)	80 (FP64)

7.2x

- 1 ExSdotp = 2 FMA = 4 FLOP
- Peak efficiency achieved computing GEMM
- Reported only low-precision formats. FP32 and FP64 are supported by all the designs
- MiniFloat-NN Snitch FP8-to-FP16: 7.2x more efficient than Snitch on FP64

[1] S. Mach et al., *"Fpnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing"*

[2] W. Mao et al., *"A configurable floating-point multiple-precision processing element for hpc and ai converged computing"*

[3] H. Zhang et al., *"Efficient multiple-precision floating-point fused multiply-add with mixed-precision support"*

[4] F. Zaruba et al., *"Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads"*

Energy Efficiency, Accuracy, SoA Comparison

Design	Tech	Voltage	Frequency	Area	Performance [FLOP/cycle]				Peak Throughput [GFLOPS]	Efficiency [GFLOPS/W]
					FP16alt	FP16	FP8	FP8alt		
ExSdotp FPU	**12 nm**	**0.8 V**	**1.26 GHz**	**0.019 mm^2**	**8/8**	**8/8**	**16/16**	**16/16**	**20.2 (exFP8)**	**1631 (exFP8)**
Fpnew [1]	22 nm	0.8 V	0.923 GHz	0.049 mm^2	4/8	4/8	8/16	-/-	14.8 (FP8)	1245 (FP8)
Mao *et al.* [2]	28 nm	1.0 V	1.43 GHz	0.013 mm^2	-/-	-/20	-/-	-/-	28.6 (FP16)	975 (FP16)
Zhang *et al.* [3]	90 nm	1.0 V	0.667 GHz	0.191 mm^2	-/-	8/8	-/-	-/-	5.3 (FP16)	113 (FP16)
MiniFloat-NN Snitch	**12 nm**	**0.8 V**	**1.26 GHz**	**0.52 mm^2**	**8/8**	**8/8**	**16/16**	**16/16**	**160 (exFP8)**	**575 (exFP8)**
Snitch [4]	22 nm	0.8 V	1 GHz	0.66 mm^2	-/-	-/-	-/-	-/-	16 (FP64)	80 (FP64)

- 1 ExSdotp = 2 FMA = 4 FLOP
- Peak efficiency achieved computing GEMM
- Reported only low-p
  and FP64 are suppo

Improved Precision

- MiniFloat-NN Snitch FP8-to-FP16:
  7.2x more efficient than Snitch on FP64

Operation	Format	Relative error vs. FP64		
		n = 500	**n = 1000**	**n = 2000**
ExSdotp	**FP16-to-FP32**	**0**	**1.1×10^{-7}**	**5.4×10^{-7}**
ExFMA	FP16-to-FP32	7.6×10^{-7}	1.8×10^{-6}	9.9×10^{-7}
ExSdotp	**FP8-to-FP16**	**5.9×10^{-4}**	**2.7×10^{-3}**	**3.9×10^{-3}**
ExFMA	FP8-to-FP16	5.9×10^{-4}	8.2×10^{-3}	1.2×10^{-2}

Conclusion

- We presented **MiniFloat-NN,** an ISA extension for low-precision floating-point training on RISC-V cores.

- The extension is based on an open-source SIMD **ExSdotp** unit[1] that has been integrated into an open-source FPU.

- A MiniFloat-NN computing cluster achieves **7.23x performance speedup** when calculating on FP8-to-FP16 data with respect to FP64 computations, and **7.2x more energy-efficient** than the baseline Snitch cluster (FP64).

[1]https://github.com/pulp-platform/fpnew/tree/feature/expanding dotp

Thank you for your attention!

@pulp_platform

pulp-platform.org

youtube.com/pulp_platform