

INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS



Virtual Conference
February 27 - March 1, 2022



MathRAMs: Configurable Fused Compute-Memory Blocks for FPGAs

Aman Arora, Aatman Borda, Tanmay Anand, Bagus Hanindhito, Lizy K. John
Laboratory of Computer Architecture
Department of Electrical and Computer Engineering
The University of Texas at Austin



Motivation

Amount of on-chip BRAM in FPGAs growing very fast (upto 100s of MB in high-end FPGAs)

Underutilization of BRAMs

- BRAMs may be left idle if not required by the application
- Limited heights and widths offered by BRAMs cause underutilization
- *Can we utilize the RAMs that are lying idle?*

Data stored in BRAMs needs to travel through routing wires to reach compute units (Logic Blocks and DSPs)

- *Can we avoid this to save energy and reduce routing congestion?*

Bandwidth inside a BRAM is typically higher than outside the RAM

- Physical number of bitlines is higher than the number of bits available at routing ports
- *Can we utilize this higher bandwidth?*

Applications like Deep Learning need flexible precision

- Int4, BFloat16, Floating-point 8-bit (HFP8 [Hybrid 8-bit Floating Point, Neurips'19])
- Recent: Tesla's Dojo has custom 8-bit and 16-bit floating-point formats
- DSPs only support a few fixed precisions. Only the Logic Blocks provide customizable precision.
- *Can we increase the Silicon area that provides flexible precision?*

Modern workloads, especially Deep Learning, are very compute intensive and highly parallel

- *Can we increase the compute throughput of the FPGA?*

Introducing MathRAMs

Add configurable processing elements inside BRAM. Convert it into a highly parallel compute unit

- Avoid moving data through the routing wires

Use dual port nature to read out two rows (containing operands) at the same time

- Instead of activating two wordlines on the same port like in Neural Cache, CCB (See Related work)

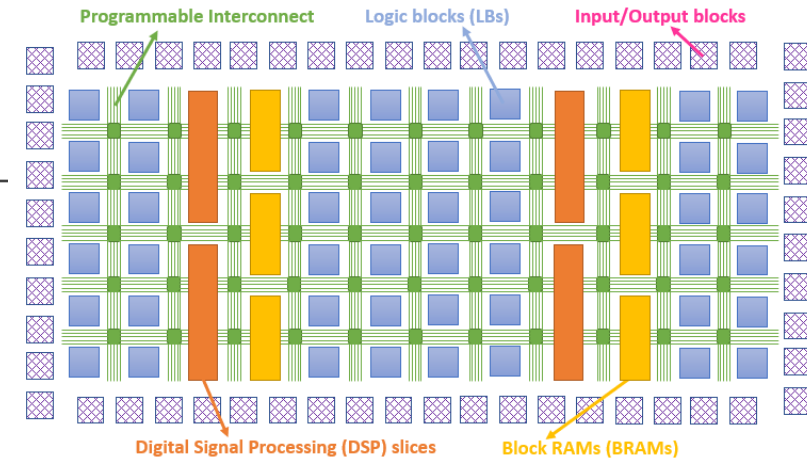
Use bit-serial computing, hence, precision agnostic (great for DL!). Supports floating-precision as well.

- Computation in any precision can be done. Makes these blocks versatile

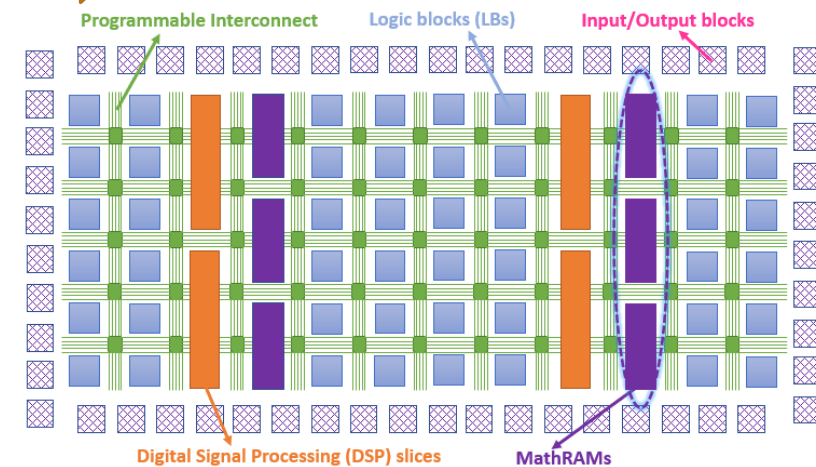
Compute with physical width of BRAM (128 bitlines instead of 32 bits outside the RAM)

- Increase effective on-chip memory bandwidth

Baseline FPGA

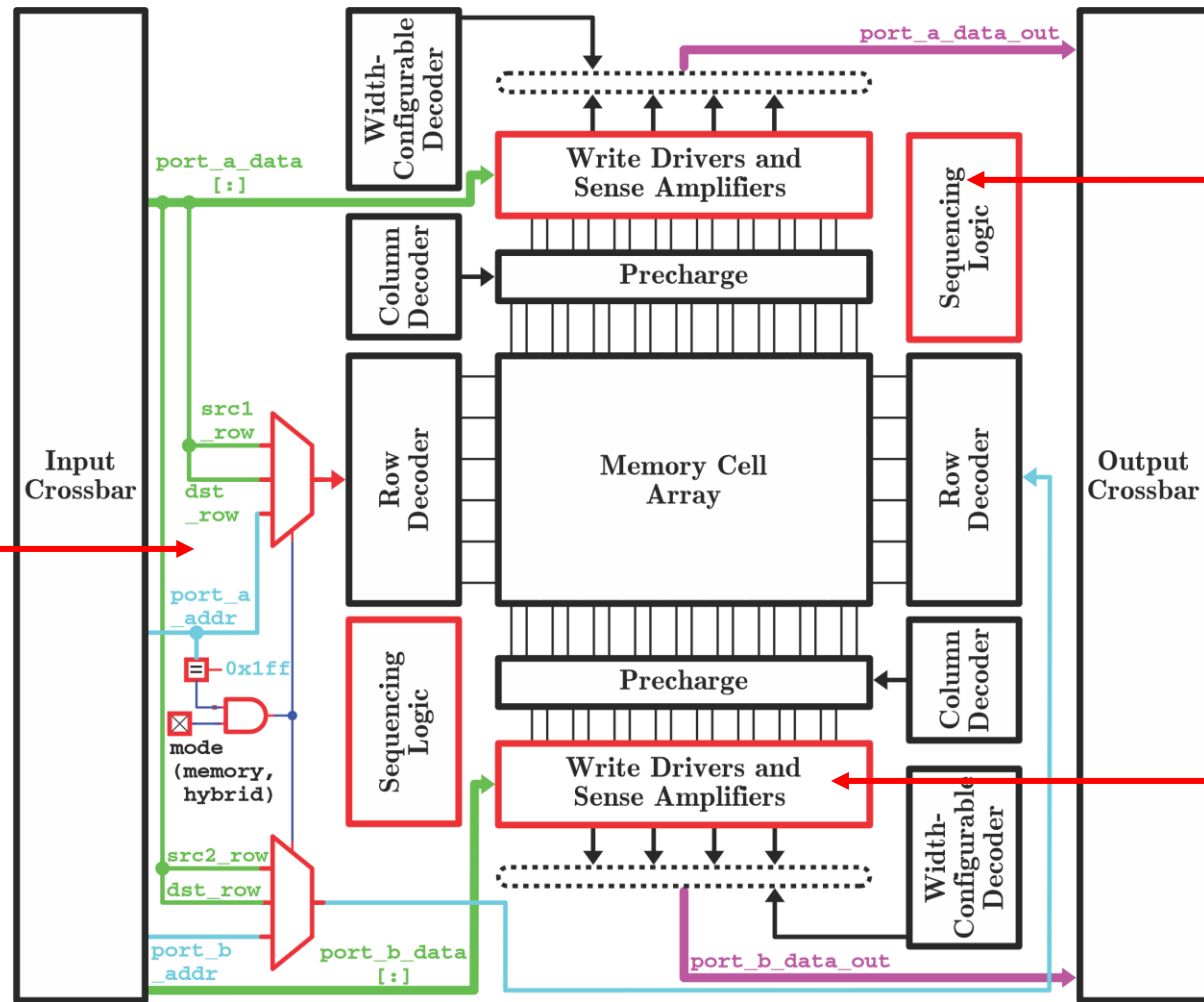


Convert BRAMs to MathRAMs. Can still be used for storage as well.



Modifications Done to the BRAM

Additional logic:
 (1) Comparator and muxes
 (2) SRAM cell to denote the mode



Change in logic to have one long clock cycle that does read, compute, write

Adding processing elements.
 Adding sense amplifiers, to have sense amplifiers on each bitline pair

Modes

A MathRAM can be configured in 2 modes at configuration time

Memory mode

- MathRAM will act as a normal BRAM
 - Only usable for storage
 - Negligible frequency overhead

Hybrid mode

- MathRAM can be used for compute as well as storage
 - Fixed width: 512x32
 - Special address 0x1ff is reserved
 - Data written to this address is treated as an instruction
 - Clock cycle is longer than baseline BRAM

Stages

An operation is typically done in 3 stages

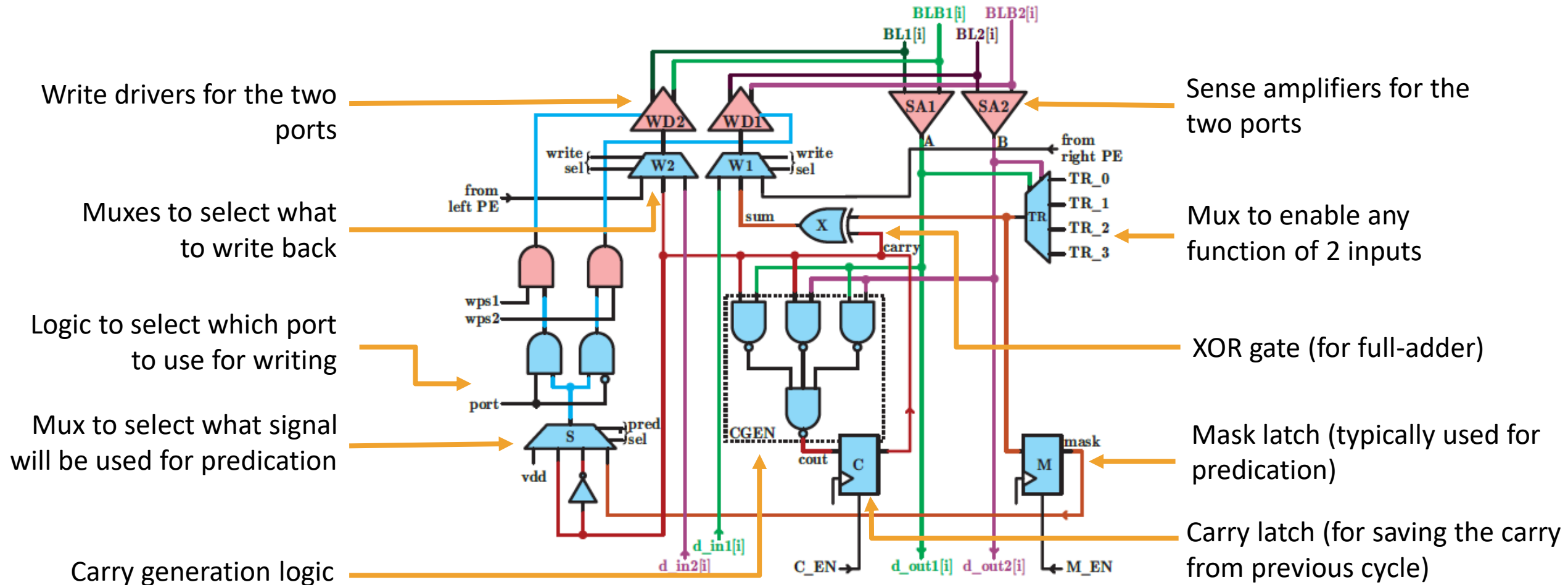
Input data is stored in transposed form into the array.

Instructions are provided to MathRAM by writing to the address 0x1ff. Source operand rows are read, PEs evaluate the results and results are written to the result row. Requires longer clock cycle than normal.

Results can be read out by reading any address

Processing Element

One processing element per bitline pair



Write drivers for the two ports

Sense amplifiers for the two ports

Muxes to select what to write back

Mux to enable any function of 2 inputs

Logic to select which port to use for writing

XOR gate (for full-adder)

Mux to select what signal will be used for predication

Mask latch (typically used for predication)

Carry generation logic

Carry latch (for saving the carry from previous cycle)

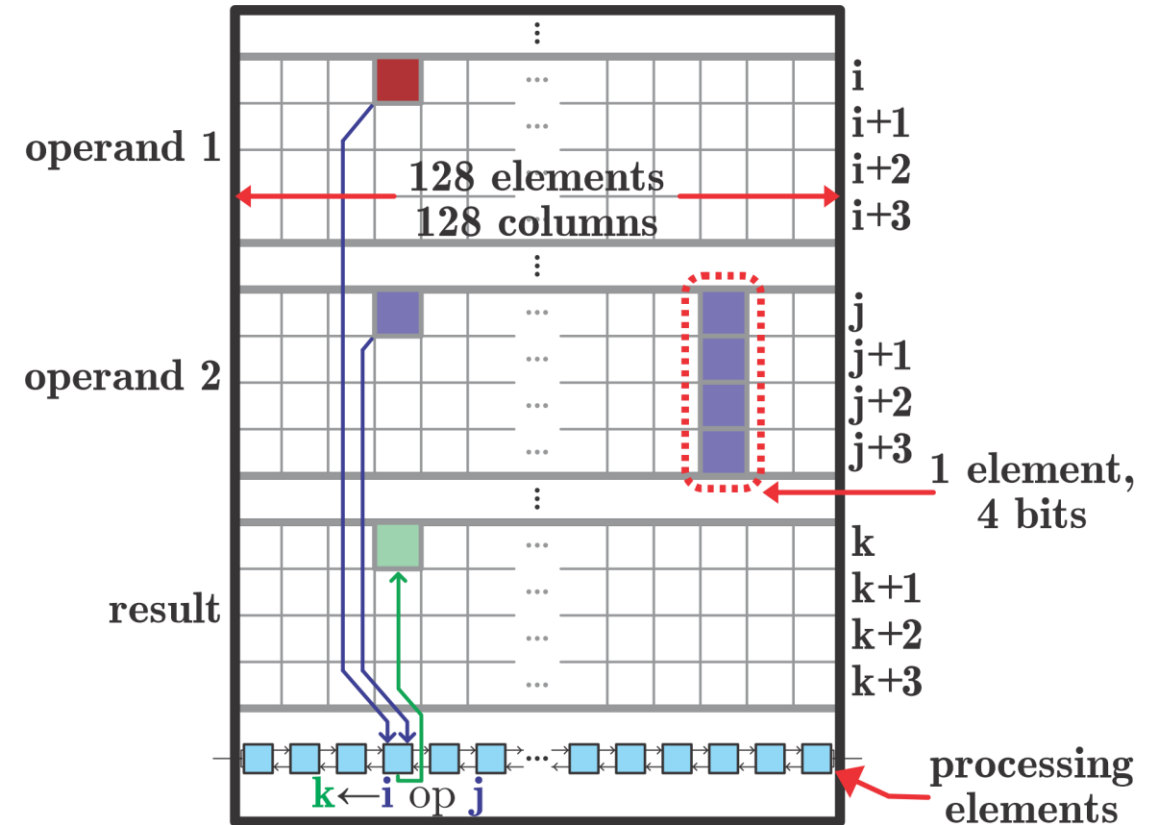
High Level Operation

All computation done in a bit-wise manner using a transposed data layout

Example: 4-bit AND operation between operand 1 and operand 2

```

operand 1 in rows  $i, i+1, i+2, i+3$ 
operand 2 in rows  $j, j+1, j+2, j+3$ 
repeat (m=0 to 3) {
    read bits in row  $i+m$  and row  $j+m$ 
    compute 'AND' in the PEs (all columns)
    write result to row  $k+m$ 
}
result in rows  $k, k+1, k+2, k+3$ 
    
```



Instruction format sent to MathRAM for computation:

31	30	29	28	27	26	25	24	21	20	14	13	7	6	0
predicate		write_sel		port		c_en		m_en		truth_table		dst_row		src1_row

Operations and Precisions Supported

The TR mux enables any 2-bit logical function, making the PE versatile

Addition and multiplication are most common. In-MathRAM reduction is also possible.

- Addition takes $N+1$ cycles
- Multiplication takes $N^2 + 3*N - 2$ cycles
- Algorithms adapted from [*Neural Cache, ISCA'18*]

Any precision can be supported

Floating point is supported through:

- Carry and not-carry used in the predication logic
- Mask is populated from the output of the programmable multiplexer TR instead of just A or B
- Operations like XOR can be performed easily using TR

Floating point algorithms adapted from [*FloatPIM, ISCA'19*]

Transposing Data

Data should be in transposed form for computation inside the MathRAM block

Swizzle logic is designed for transposing data

- Circular FIFO based implementation
- Mapped to soft logic on the FPGA
- Overhead is high (~100 Logic Blocks for each instance)
- Reducing this is future work
- Options to explore:
 - Harden swizzle logic
 - Transpose in the DRAM controller
 - Use TMU [*Neural Cache ISCA'18*]

Shifting and Chaining

Native support for left-shift and right-shift operations

- Each PE connects to neighboring PE within the MathRAM block

Direct connections between neighboring MathRAMs in a column

- Allow left-shift and right-shift operations *across* MathRAMs as well
- Enables more parallelism
- Useful in applications like DSP filters

Evaluation

VTR 8.0 [Univ. of Toronto]

- Evaluating designs on hypothetical FPGA architectures, for example, an FPGA containing MathRAM blocks

COFFE 2.0 [Univ. of Toronto]

- For estimating area and delays of FPGA components, specifically BRAMs and MathRAMs

Synopsys HSPICE

- SPICE level simulations of the MathRAM and the processing element

Synopsys VCS

- Verilog functional simulations for the benchmarks

Area overhead

- ~26% larger compared to a BRAM
- Additional sense amplifiers (128 instead of 32)
- Processing elements
- Overall, 5.7% increase in the area of the baseline FPGA

Frequency overhead

- ~25% slower compared a BRAM in Hybrid mode
- Because of performing read, compute and write in the same cycle
- Negligible degradation in Memory mode

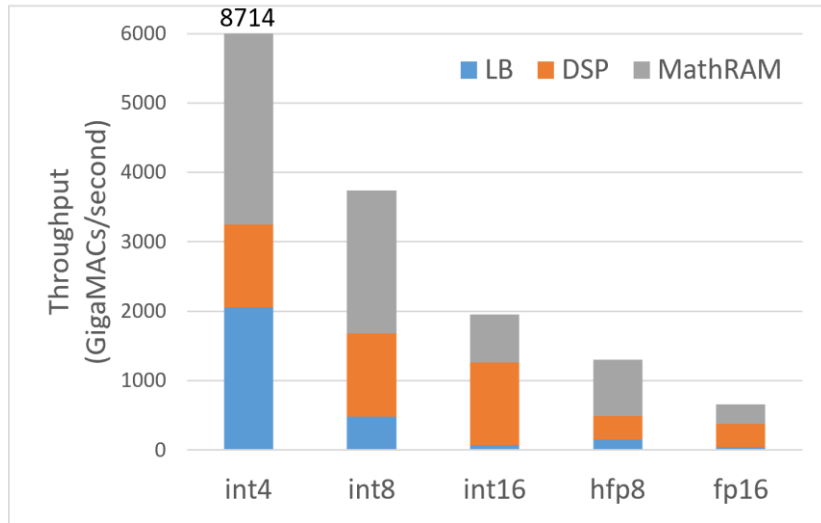
Baseline Architecture

- Island style. Blocks arranged in columns.
- Resource mix similar to Intel Stratix 10 GX 650
- Routing architecture similar to Intel Stratix IV
- 22nm technology node
- BRAM size = 16 Kilobits
- Physical geometry of BRAMs = 128 rows x 128 columns

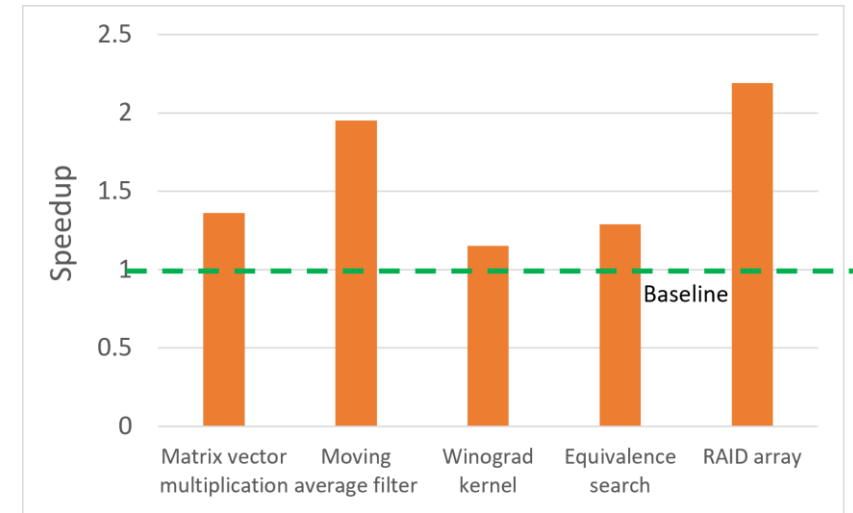
Proposed Architecture

- Replace BRAMs with MathRAMs

Peak throughput for MAC operations for the whole FPGA. *Takeaway: MathRAMs significantly increase the compute throughput of the FPGA*



Speedup obtained using an FPGA with MathRAMs compared to baseline FPGA for various applications



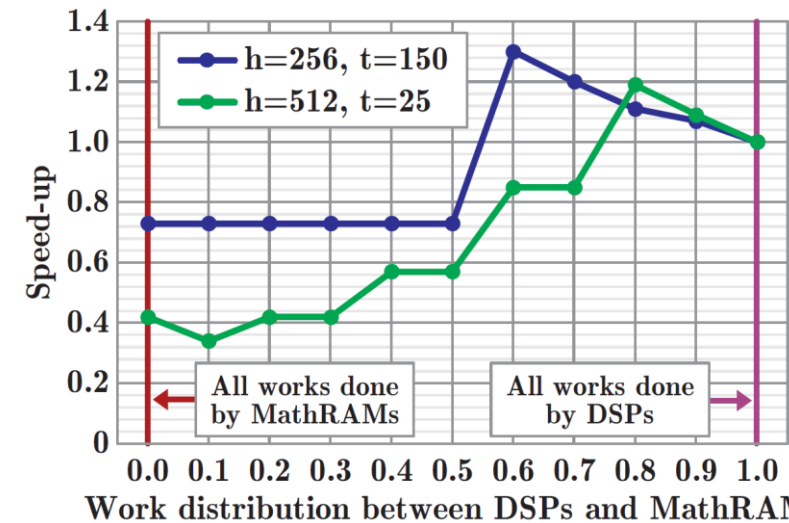
Results

Metric	Taps 128		Taps 256	
	Baseline	Proposed	Baseline	Proposed
LBs	91%	76%	90%	76%
RAMs	3%	78%	2%	78%
Frequency	134MHz	225MHz	146MHz	242MHz
Total Cycles	122880	105648	245760	210628
Total Time	917 μ s	469 μ s	1683 μ s	870 μ s
Speed-up	1	1.95	1	1.93

Moving Average Filter, Precision = 16-bit

Takeaway: Almost 2x speedup obtained by using MathRAMs.

Takeaway: More MathRAMs were available but couldn't be fed with the available DRAM bandwidth. Need more DRAM bandwidth to feed more compute throughput.



Matrix Vector Multiplication from 2 DeepBench LSTM workloads. Using both MathRAMs and DSPs for computation. Precision = int8
Takeaway: Design space exploration is necessary to find the sweet-spot of work distribution between DSPs and MathRAMs

Conclusion



Propose changing BRAMs to MathRAMs on FPGAs, fusing computation and storage capabilities in one block



Speed up multiple applications by increasing compute throughput, at the cost of 5.7% area overhead at the FPGA chip level, compared to an Intel Stratix 10-like baseline FPGA



First work to: (1) Utilize dual port nature of BRAMs for compute, (2) Deploy configurable 1-bit processing element, (3) Apply PIM to FPGAs for non-DL applications