In-Memory Computation

Hardware for Machine Learning

Mark Vesterbacka
2020-03-16

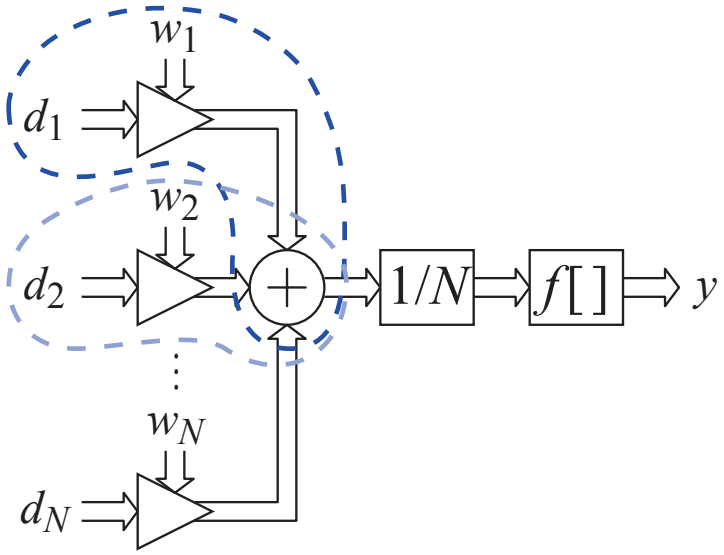
LINKÖPING UNIVERSITY

Outline

- Introduction
- Embedding in memory
- Circuits and functionality
- Published results
- Conclusion

Inference

- Typical dataflow



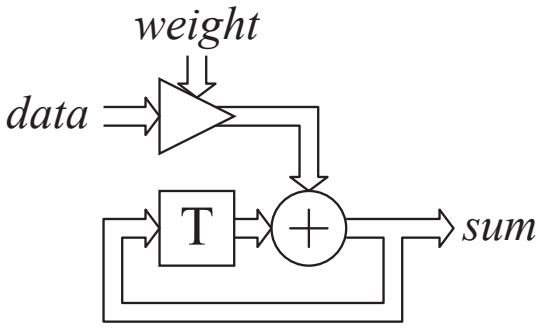
- Computations [1]

Algorithm	Distance	$f[\,]$
SVM	Dot product	sign
TM	Manhattan distance	min
k-NN	Manhattan distance	majority vote
MF	Dot product	max

- Hardware: *multiply-and-accumulate*

Memory access bottleneck

- Memory accesses per MAC
  - Two data reads (*data* and *weight*)
  - One data write (partial *sum*)
- GMACs per image, examples [2]
  - LeNet-5: 0.00034
  - AlexNet: 0.72
  - Overfeat fast: 2.8
  - VGG-16: 15.5
  - GoogLeNet v1: 1.43
  - ResNet-50: 3.9



Memory access cost

- Energy cost of memory access compared with MAC [3]
  - Accessing a *register* costs ~ E_{MAC}
  - Accessing other *MAC* costs ~ $2E_{\text{MAC}}$
  - Accessing *cache* costs ~ $6E_{\text{MAC}}$
  - Accessing *external* memory costs ~ $200E_{\text{MAC}}$

In-memory computing

- GMACs, Greads, Gwrites costs energy…

  …can we do this cheaper?

- Inference can often be done with low precision

  …we could try *approximate analog computing*

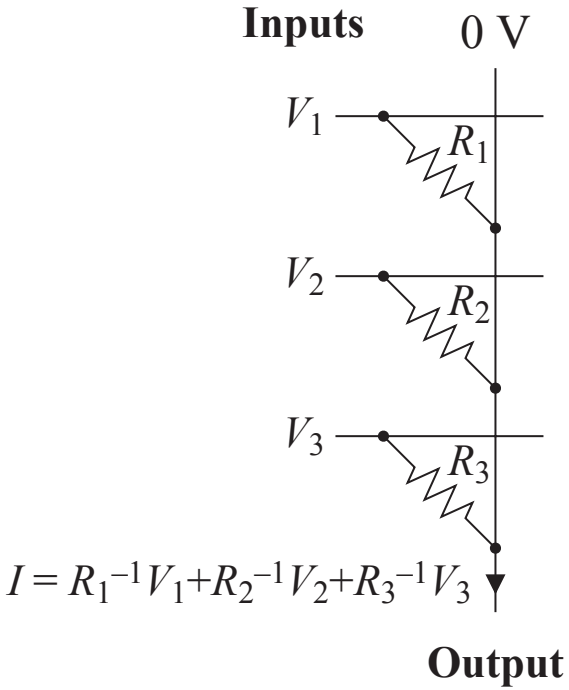
- Communication with memory is expensive

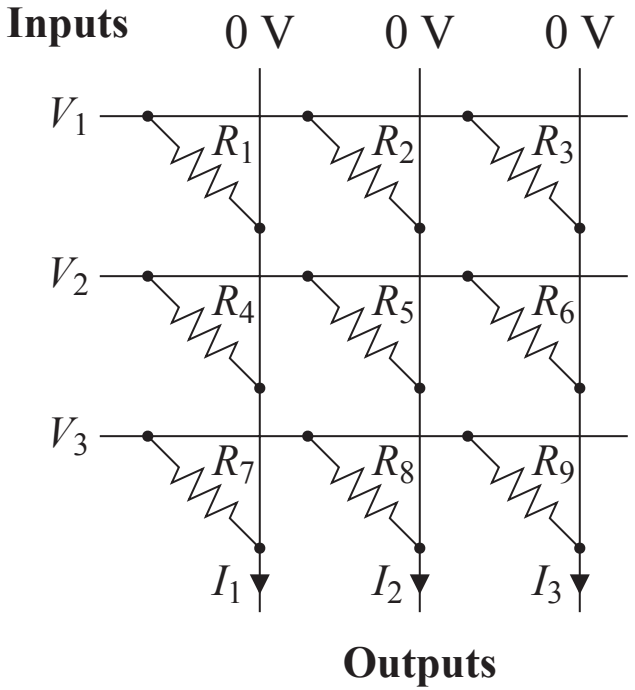
  …let us minimize distance to memory

A solution could be *analog computation inside the memory*

Analog multiplication

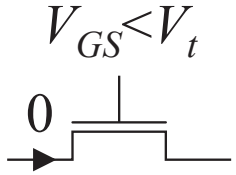
- Ohm's law ⇒ multiplicate, KCL ⇒ accumulate
- Vector-matrix multiplication

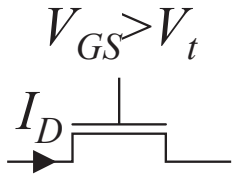


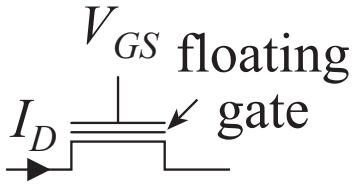


Programmable resistors

- Memristor (memory resistor) 
  - Thought device for which *R* can be programmed with *I*
  - Remembers *R* when $I = 0$ or $I(t) = \hat{I}\sin(\omega t)$
- Field-effect transistor (FET)
  - Program V_{GS} from memory cell
  - Easiest to get linear operation with constant V_{GS}
- Floating gate transistor
  - Used in flash memory
  - Program V_t by storing charge in extra gate

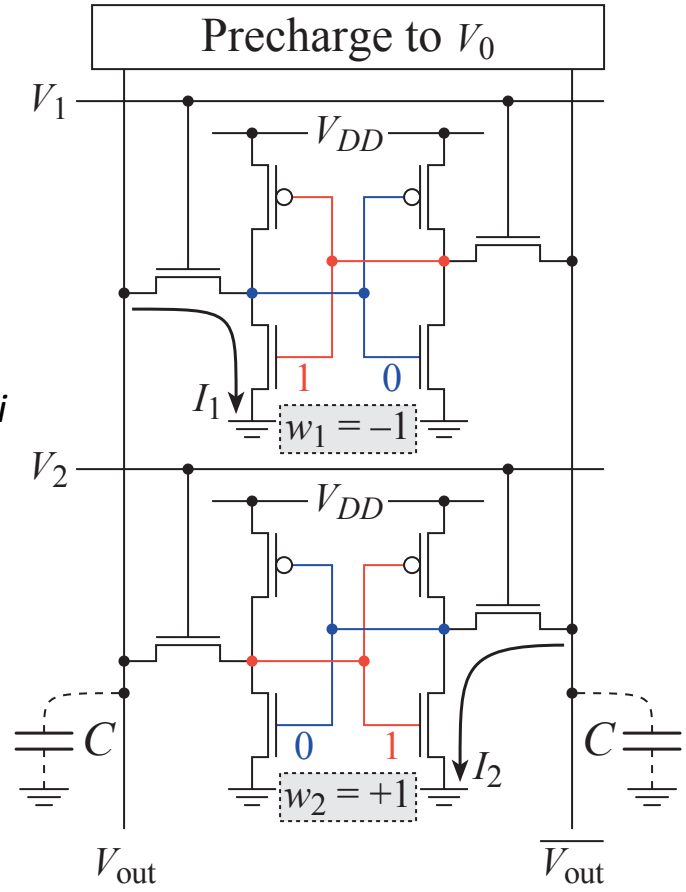






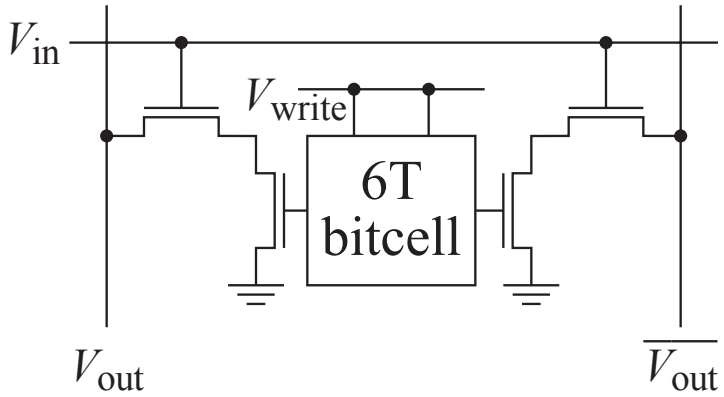
Standard 6T bitcell

- A 6T memory cell can control *I*
  - *C* is precharged to V_0
  - V_i:s are pulsed in amplitude and width
  - Output is discharged to resulting V_{out}
- Differential output can represent $\pm w_i$
  - –1 is stored in top and +1 in bottom cell
- We can obtain a linear classifier with
  - $y = \text{sign}\left(\sum_{i=1}^{N} w_i x_i\right)$



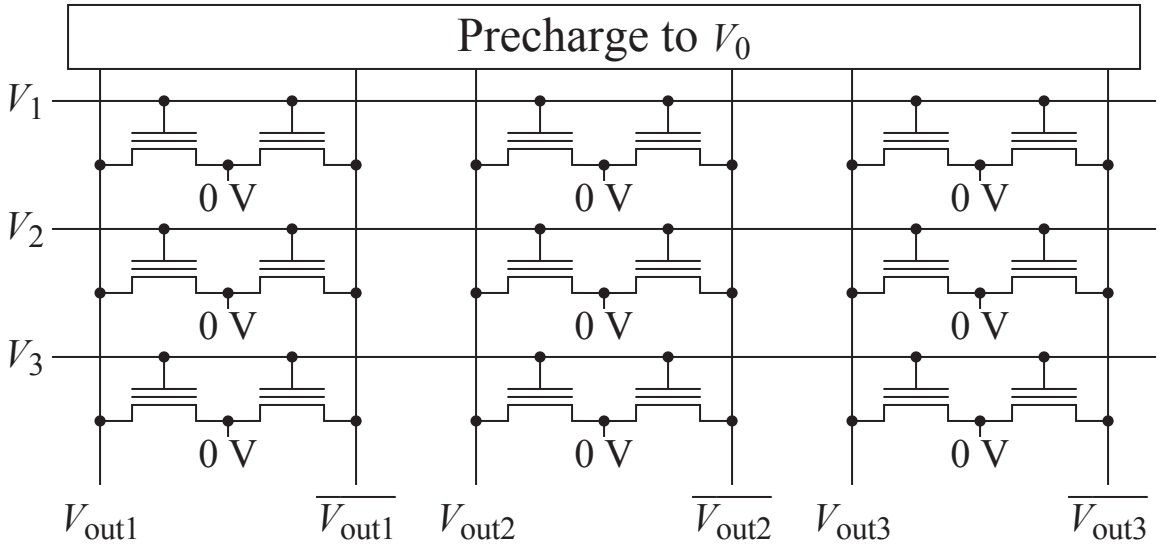
10T bitcell

- A 6T cell must have low V_{out} to not ruin the stored bit
- A 10T cell buffers V_{out} from the bitcell during access [4]
  - Pseudo-writes from multiple bit-cells discharge are avoided
  - Almost full rail swing can be used for analog computation

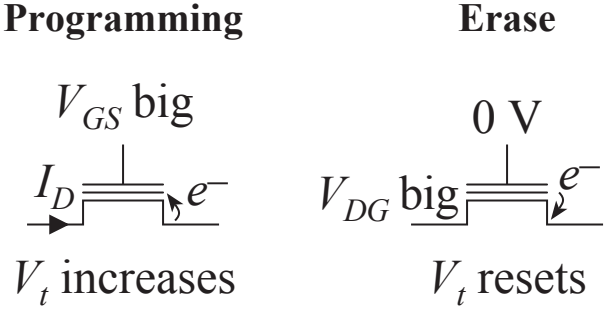


Floating gate transistors

- Array
  - 1T per bit

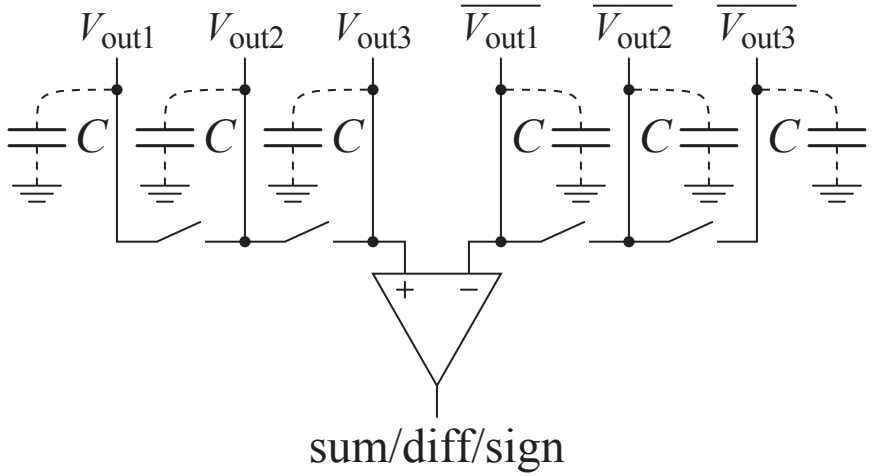


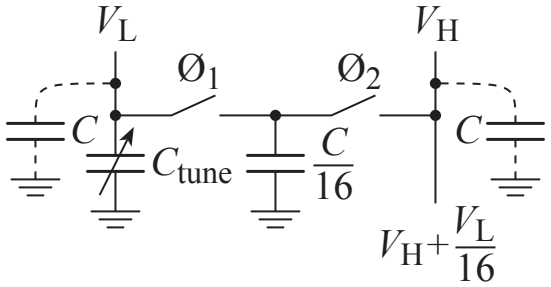
- Programming [5]
  - V_t is programmed



Bitline processing

- Bitlines are connected together with switches [1]
  - Compute difference or scalar product through charge sharing

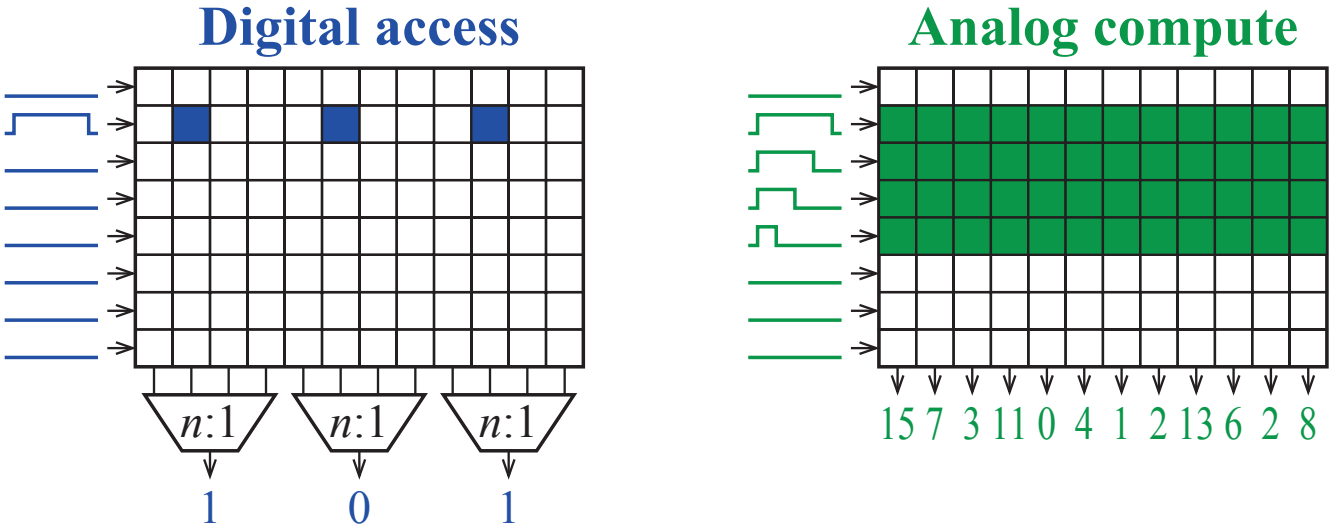




- Sub-ranged read [2]
  - It is difficult to obtain more than 4-bit accuracy per bitline
  - Sub-ranged reads (right) can obtain double, 8-bit accuracy

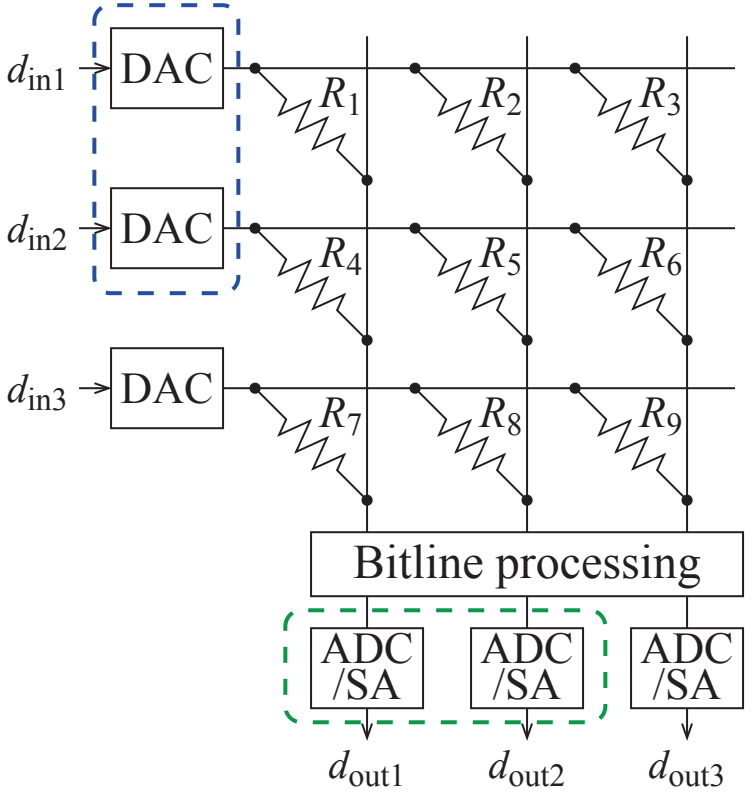
Memory access patterns

- Digital and analog arrays can be designed identically
  - Standard digital array outputs bits for digital MAC
  - Analog output is discharged to an *n*-bit analog voltage



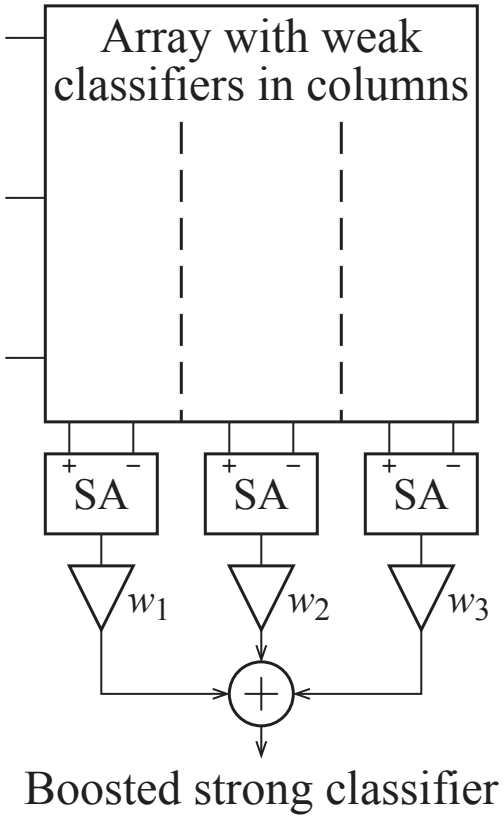
Interface

- Data converters are needed for interfacing
- Input
  - DACs are possibly shared
- Output
  - ADCs are possibly shared
  - SAs are used for decisions



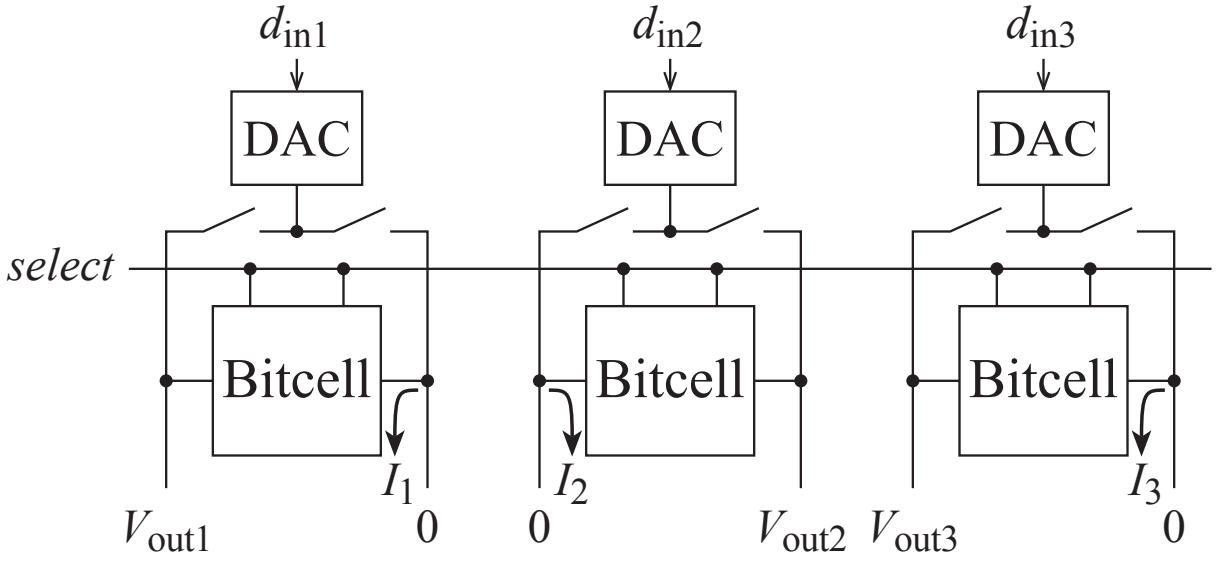
On-chip training for nonidealities

- Circuit variations are large
  - Analog circuits may need correction
- Boosting [6]
  - *Boosting* constructs strong classifier from weak base classifiers
  - *Adaptive boosting* corrects fitting errors iteratively
  - *Error-adaptive classifier boosting* corrects non-ideal classifiers



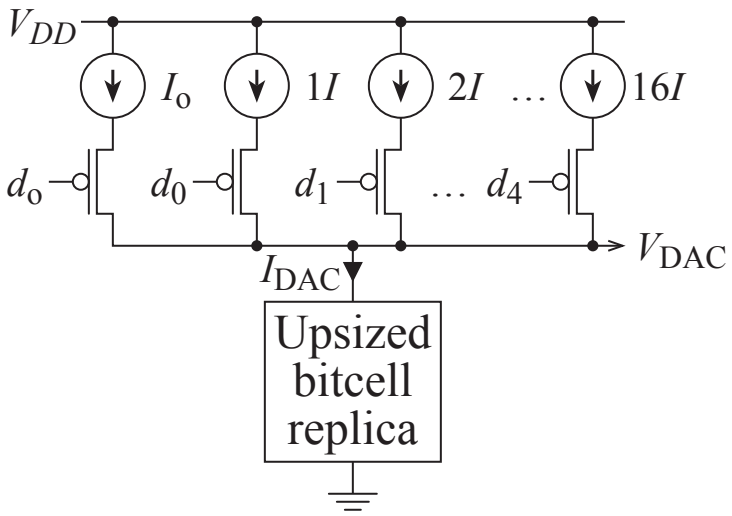
Avoiding on-chip training

- Apply input on bitline instead of wordline [4]



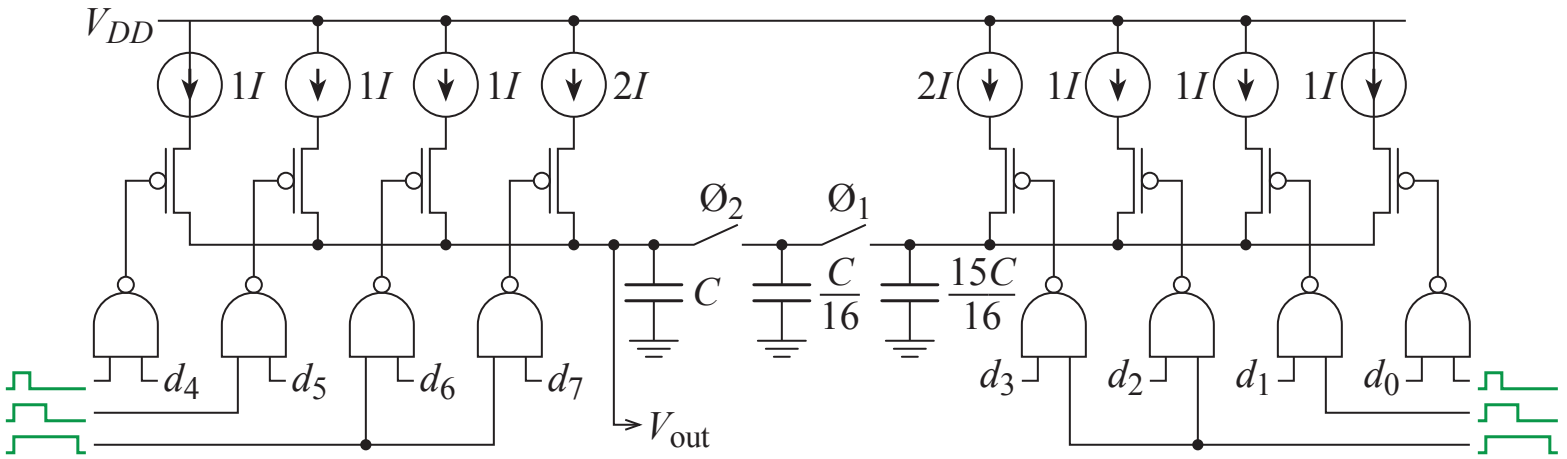
- Use binary weights to discharge bitline fully for linearity

Simple DAC



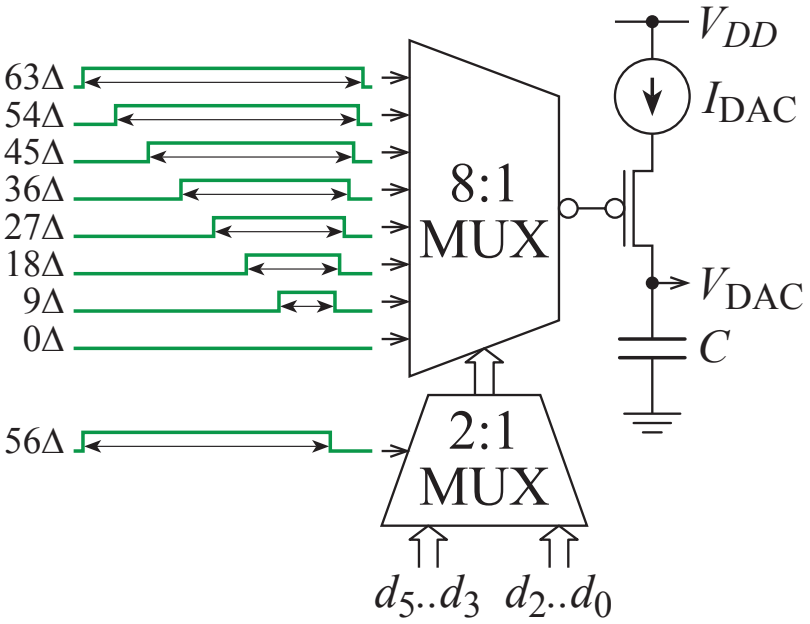
- DAC with binary-weighted current sources [6]
- Bitcell replica generates proper voltage for γI_{DAC} in bitcells
- I_o is an offset that increases linearity

Switched-capacitor DAC



- DAC/multiplier with 4-bit MSB and 4-bit LSB unit [7]
- Exponential function is obtained by charging *C* through *R*

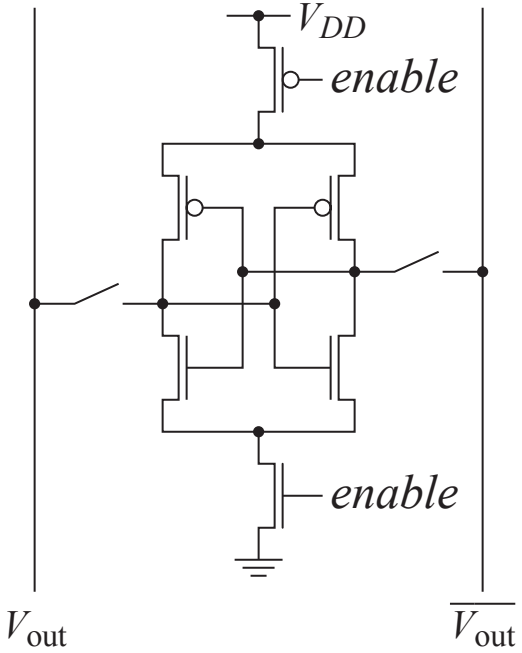
DAC with digital-to-time converter

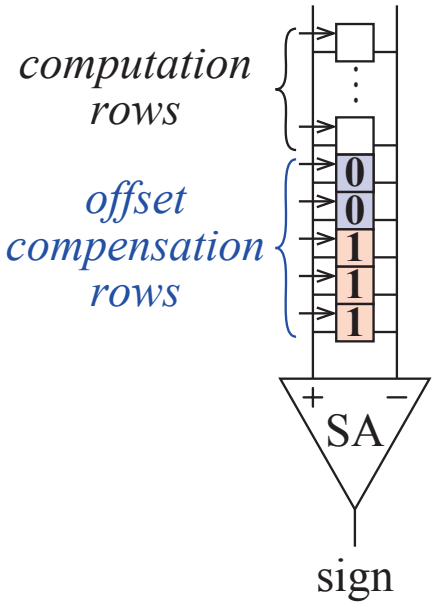


- DAC output is generated by combining clock pulses [4]
- A constant current I_{DAC} charges the output V_{DAC}

Sense amplifier

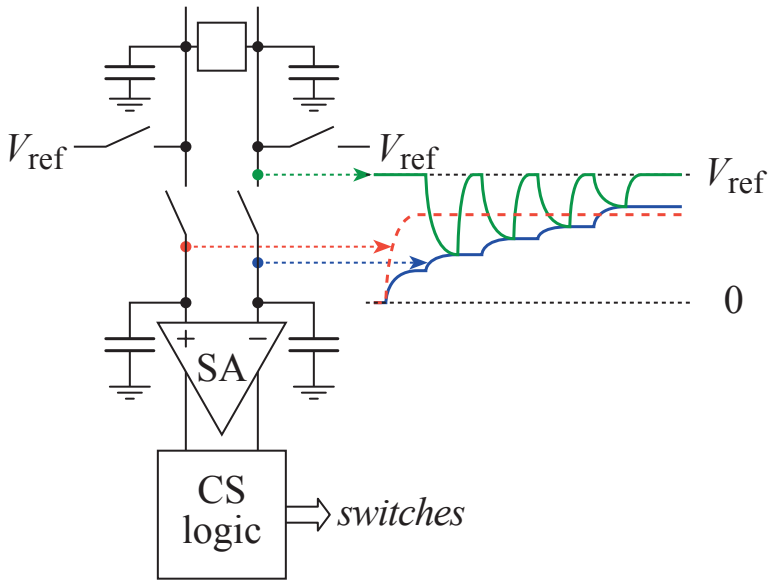
- Sense amplifier (SA) compares or computes sign [6]
  - Design for larger voltage swing than in standard SRAM
  - Can compensate for offset with extra memory rows





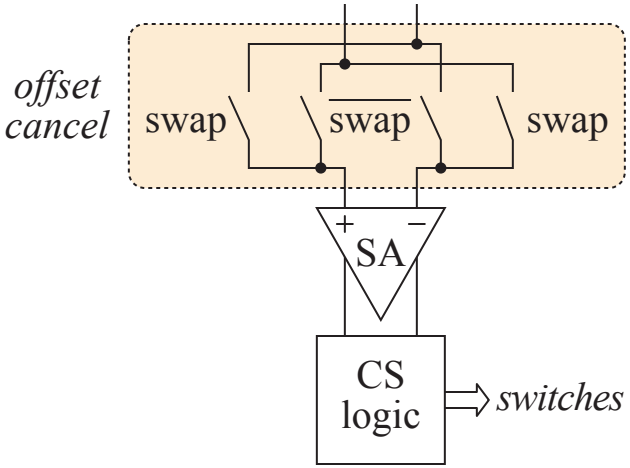
Charge-sharing ADC

- Charge-sharing based integrator with replica bitlines
  - A logic block provides timing signals
  - A counter counts the number of cycles it takes to equalize voltage



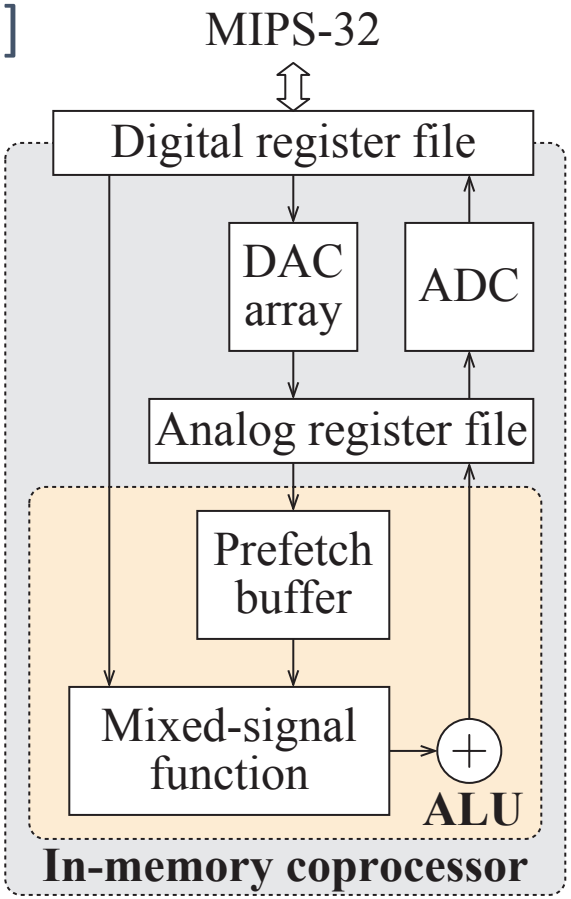
Offset cancellation in ADC

- The charge-sharing ADC is directly affected by SA offset
  - A solution is to make a double conversion
  - Swapping inputs between conversions and average cancels offset



Analog programmable coprocessor

- In-memory computing coprocessor [7]
- Handles analog cache misses, register refresh, delayed write back to registers, multiplication
- Data conversion occurs when analog registers are flushed to digital
- Registers use SC memory cells
- Tuning C:s compensate for PVT



Coprocessor interface

- The analog pipeline aligns with the digital pipeline

Digital processor	fetch	decode	D execute	memory access	write-back
Analog coprocessor		DA conversion/ A register access	A execute (MAC, exp, ReLU)	A register access/ AD conversion	delayed write-back

- GCC cross compiler inserts approximate analog computing instructions in the back-end code generation
- Compiler tracks number of instructions since analog store
- Write-back to digital memory before error accumulates

Custom instructions of coprocessor

- Instructions for approximate analog computation [7]
  - MOVEAn Transfer vector between analog and digital registers
  - LHA Load a half-word (16-bit) from memory
  - SHA Store a half-word (16-bit) to memory
  - LWA Load a word (32-bit) from memory
  - SWA Store a word (32-bit) to memory
  - EACAn Vector $A = A + \sum_{i=1}^{n} exp(A + X_i)$
  - MACAn Vector $A = A + \sum_{i=1}^{n} X_i Y_i$

Results — Inference in standard SRAM

- Four inference tasks mapped on test circuit [1]
  - SVM, MF, *k*-NN, TM
- In-memory architecture vs single-function ASIC
  - achieves ≤ 1% accuracy degradation
  - requires 16x fewer read accesses
- Measured energy savings over conventional architecture
  - MD mode 5x
  - DP mode 10x

Results — Dot-product for CNNs

- SRAM-embedded convolution was investigated [4]
- Measured error rates are close to ideal digital values
- Batch normalization improves error rates with ~ 30%/layer
- Energy efficiency of published works operating on MNIST
  - ~ 30x improvement over standard digital
  - ~ 8x improvement over digital near-memory
  - Similar efficiency as digital solution with 1-bit weights

Results — Classifier in standard SRAM

- Classifier was implemented in a standard SRAM [6]
- Classification of MNIST handwritten numbers
  - Linear classifier 10-bit quantization yields 96% accuracy
  - Linear classifier 1-bit quantization yields 52% accuracy
  - Linear classifier 1-bit optimized quantization yields 91% accuracy
  - Measured data with 18 EACB iterations yields 90% accuracy
- Estimated energy of 10-way classification
  - Conventional digital system: 71 nJ/classification
  - Digital system with 1-bit weights: 7.9 nJ/classification
  - Analog system with 1-bit weights: 0.63 nJ/classification

Results — Programmable coprocessor

- An analog pipelined coprocessor was modeled [7]
- Benchmarks was evaluated (results were similar)
  - FFT, k-NN, SVM, LeNet5, Google MobileNet, Quick sort
- Estimated accuracy and normalized throughput and energy of processor configurations running LeNet-5

Configuration	Accuracy	Throughput	Energy
Digital processor	98%	1	1
+ scalar coproc.		2.5	0.96
+ vector coproc.	97%	15	0.61

Conclusion

- MAC fits embedding into a memory array well
- Accuracy of analog computing matches that of inference
- Energy efficiency can improve with an order of magnitude
- Latency and throughput improve due to the parallelism

References

[1] M. Kang, S. K. Gonugondla, A. Patil and N. R. Shanbhag, "A Multi-Functional In-Memory Inference Processor Using a Standard 6T SRAM Array," in IEEE Journal of Solid-State Circuits, vol. 53, no. 2, pp. 642-655, Feb. 2018.

[2] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017.

[3] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in Proc. ISCA, 2016, pp. 367–379.

[4] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An Energy-Efficient SRAM With In-Memory Dot-Product Computation for Low-Power Convolutional Neural Networks," in IEEE Journal of Solid-State Circuits, vol. 54, no. 1, pp. 217-230, Jan. 2019.

[5] R. Han et al., "A Novel Convolution Computing Paradigm Based on NOR Flash Array With High Computing Speed and Energy Efficiency," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 5, pp. 1692-1703, May 2019.

[6] J. Zhang, Z. Wang and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," in IEEE Journal of Solid-State Circuits, vol. 52, no. 4, pp. 915-924, April 2017.

[7] S. Chung and J. Wang, "Tightly Coupled Machine Learning Coprocessor Architecture With Analog In-Memory Computing for Instruction-Level Acceleration," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 3, pp. 544-561, Sept. 2019.

Questions?

LINKÖPING UNIVERSITY