Accelerator Architectures for Machine Learning

Lecture 11: Analog DNN Accelerators

Friday: 1:20 – 4:20 pm
Classroom: EC-221

Acknowledgements and Disclaimer

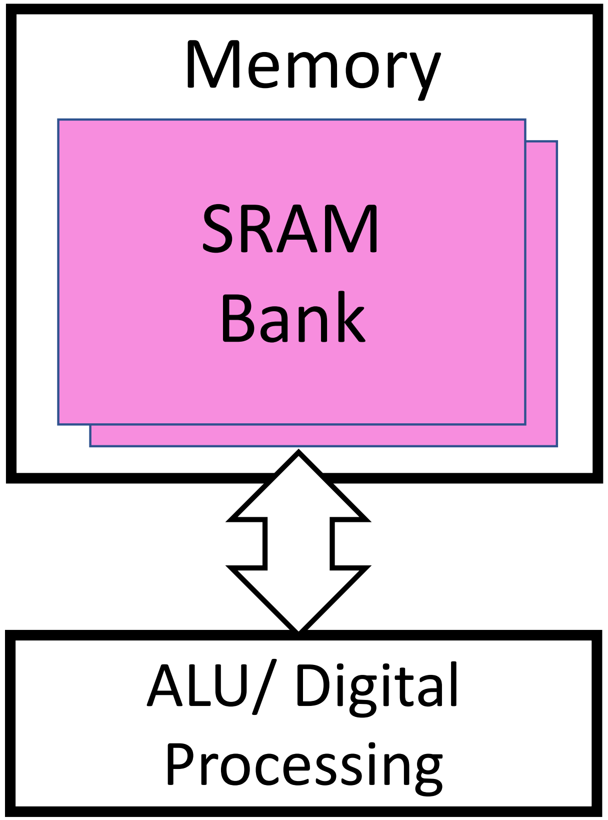
- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020
- CS7960, Neuromorphic Accelerator, University of Utah
  https://www.cs.utah.edu/~rajeev/cs7960

Outline

- Processing-in-memory architecture
  - Memrister
- Spiking neural network
  - Neuromorphic accelerator

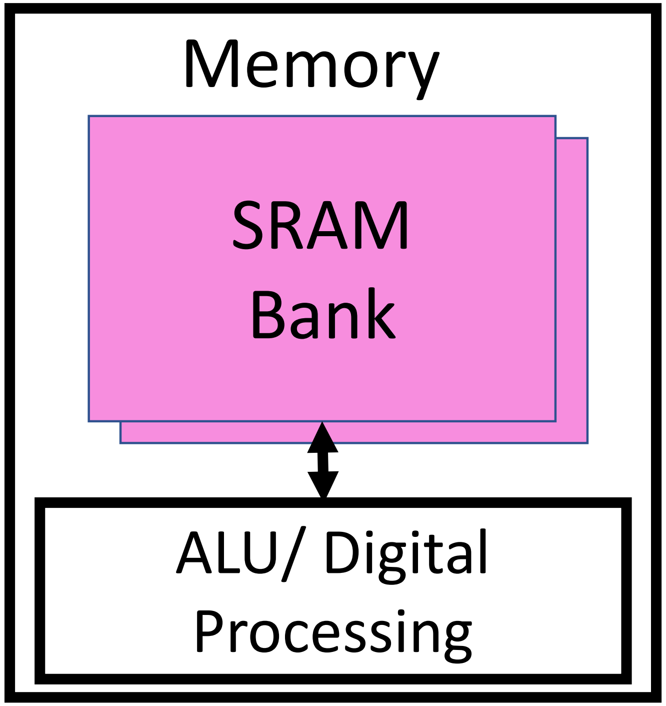
Data Access Overhead in Digital Computer

- **Digital computer**
  - Fabricated MEM and ALU as separated chips (Why?)
  - Limited by # of I/O pads per chip and off-chip interconnect channels
  - High data access energy and latency
  - Reuse and compress data to improve in-efficient data access



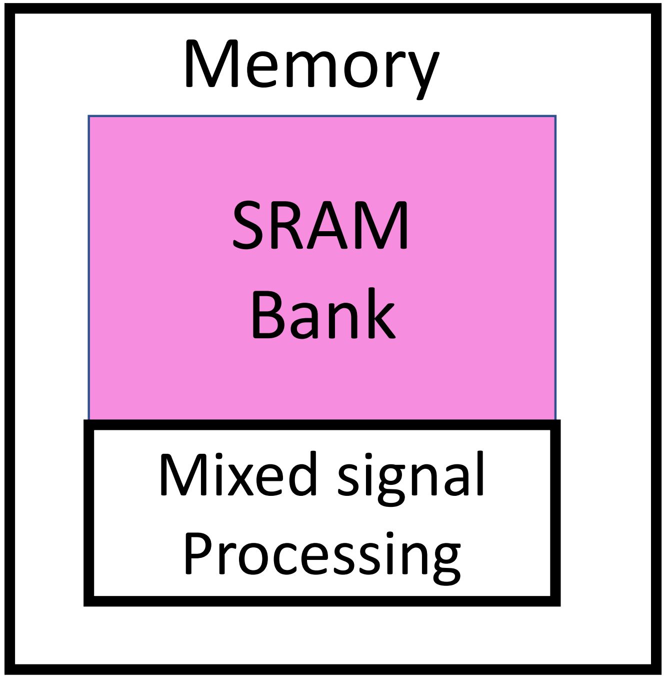
Processing Near Memory

- **Near memory processing**
  - Computation is still in digital manner
  - High bandwidth communication in-between memory and ALU
  - HMC and HBM 3D-stacked memory
  - Eliminate data transfer costs
  - Memory read energy dominates



Processing in Memory

- **Deep in-memory**
  - Combined memory access and computation
  - Mixed digital and analog computation
  - Significant energy and latency reduction
  - DAC/ADC overhead and error-prone operations

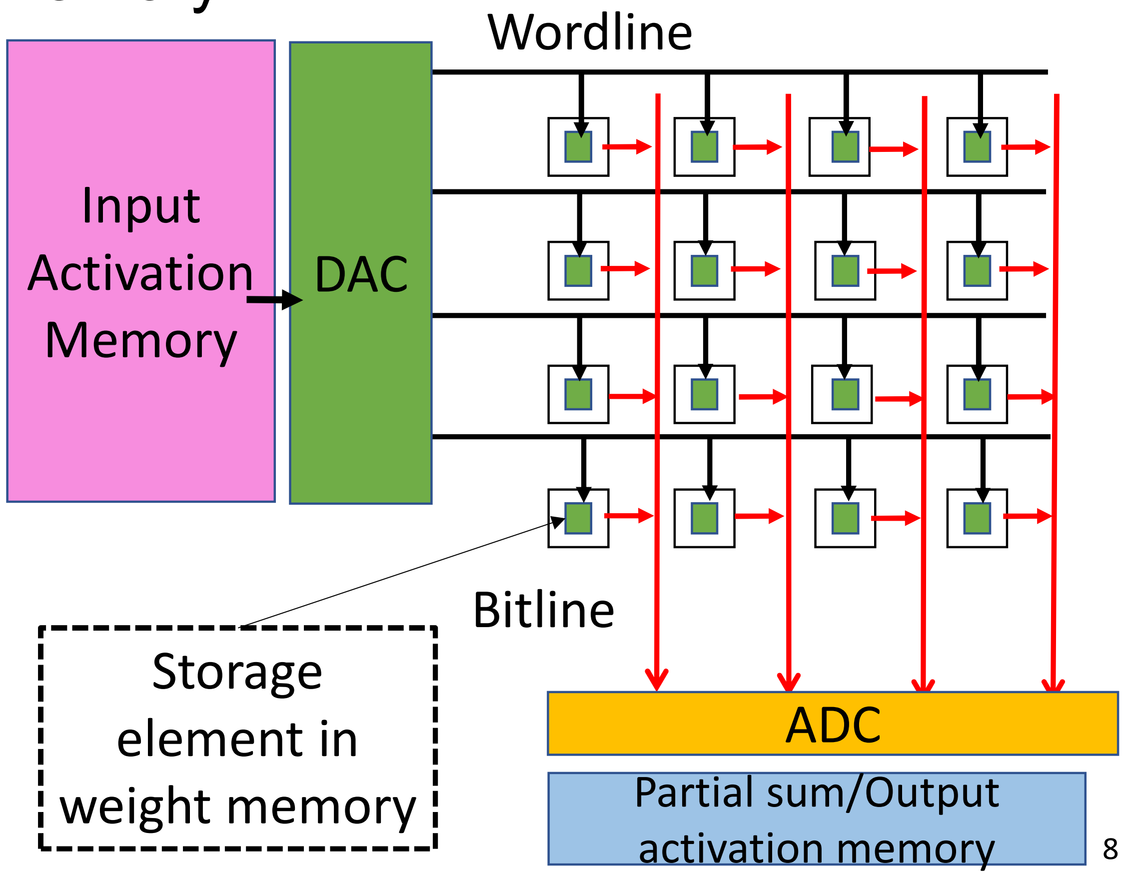


Processing Near Memory

- Bring compute closer to the high density memory
- **Benefits**
  - Increase memory bandwidth
  - Reduce energy per access
- **Approaches**
  - Reduced interconnect length and wider interconnect
  - Embedded DRAM (eDRAM), non-volatile (eNVM)
- **Challenges**
  - Limited access patterns to allocate data to MEM banks and vaults
  - NoC between MEM and PEs

Dataflow in Processing in Memory (PIM)

- Brings the compute **into** the memory
- **Weight-stationary** dataflow
  - WLs deliver inputs to storage elements
  - BLs read computed outputs and partial sums
  - MAC is performed at each storage element
  - In theory, A x B MAC operations per cycle



Challenges of PIM

- **Number of storage elements per weight**
  - Limited precision of each device or bit cell
  - Needs multiple low-precision storage element to represent a higher precision weight

4 × 4

1	2	3	2
0	1	2	0
3	0	1	1
2	3	1	1

Decimal value

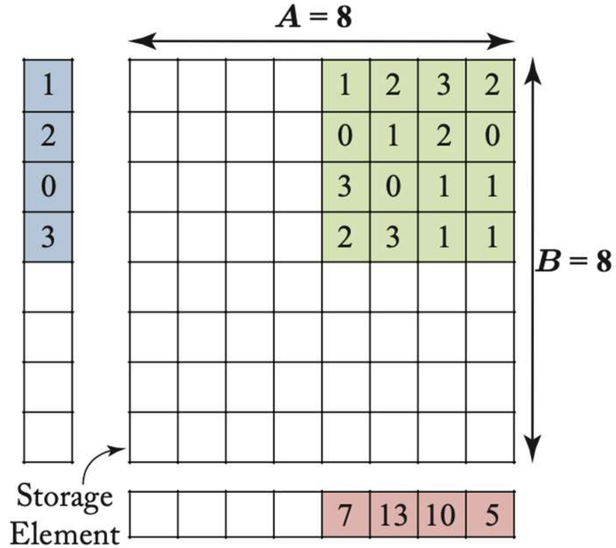
4 × 8

0	1	1	0	1	1	1	0
0	0	0	1	1	0	0	0
1	1	0	0	0	1	0	1
1	0	1	1	0	1	0	1

Binary value

Challenges of PIM

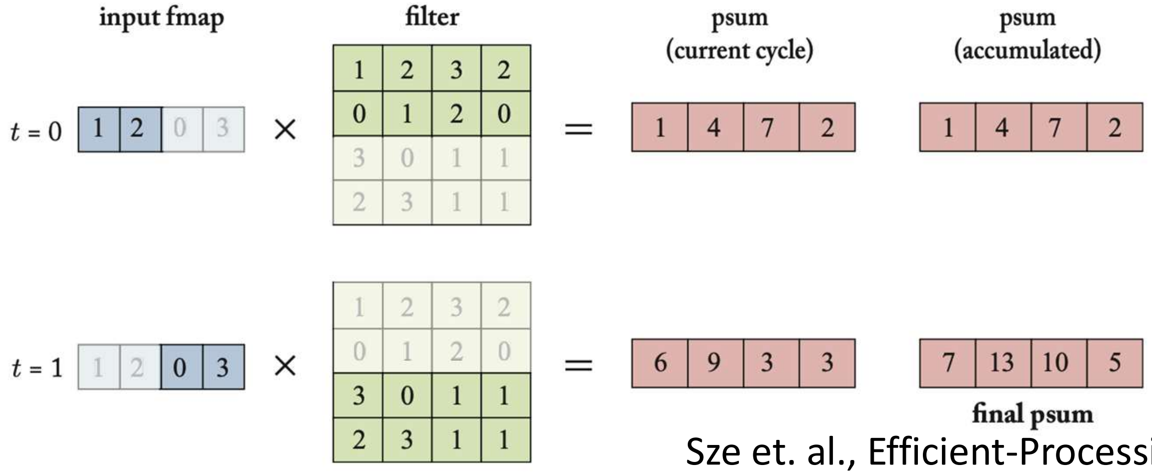
- **The size of memory array**
  - Large memory array increases the cost of peripheral circuit
  - ADC and DAC can account for over 50% energy in NVM memory
  - A large bitline capacitance is difficult to sense charge stored in the bit cell
  - Impact the utilization of memory array



Sze et. al., Efficient-Processing-of-Deep-Neural-Networks, 2020

Challenges of PIM

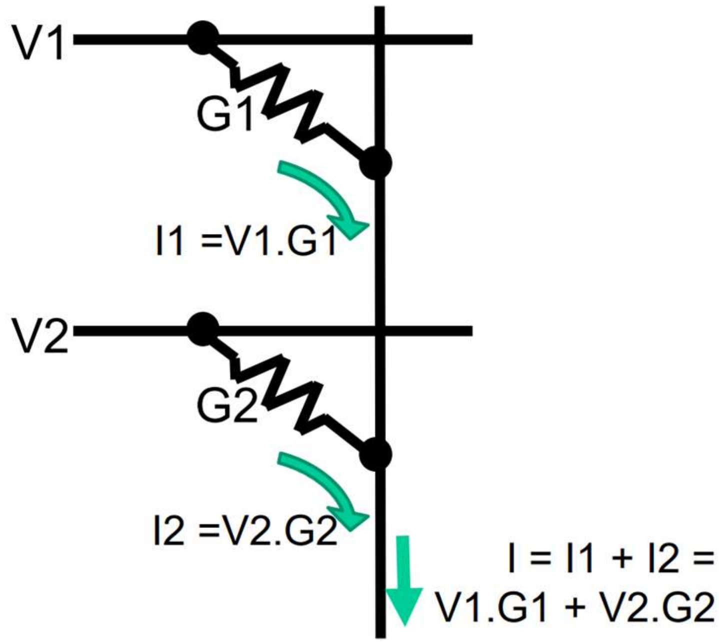
- **Number of rows activated in parallel**
  - ADC is hard to resolve many bits
  - Hard to control bit line operations in advanced process technologies
  - If ADC is only 3-bits, only two rows can be used at a time. -> more cycles to complete the computation



Sze et. al., Efficient-Processing-of-Deep-Neural-Networks, 2020

Analog Acceleration

- Many electronic phenomena correspond to multiplication and addition
- **The example**
  - Compute the dot-product with the 3 wires and 2 resistors
  - Each resistor injects a current that is the product of V1 and G1
  - By merging two wires and performing addition of currents
  - The dot-product is being performed in the analog domain

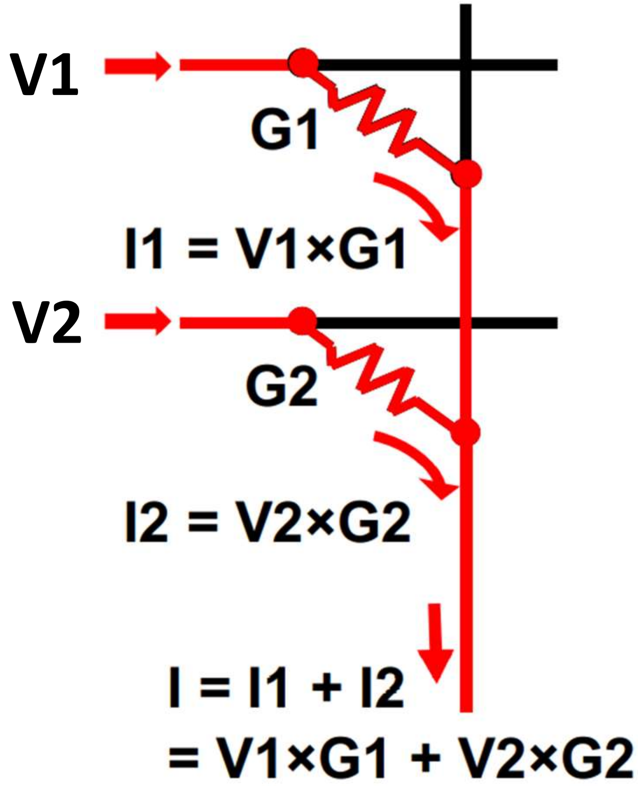


Memristor Analog Computation

- Current-based non-volatile memory
- **Conductance = weight**
- **Voltage amplitude = Input**
- **Current = Voltage x Conductance**
- **Sum currents for addition**

  Output = ∑ Weight x Input
- Input = V1, V2
- Filter Weight = G1, G2, …

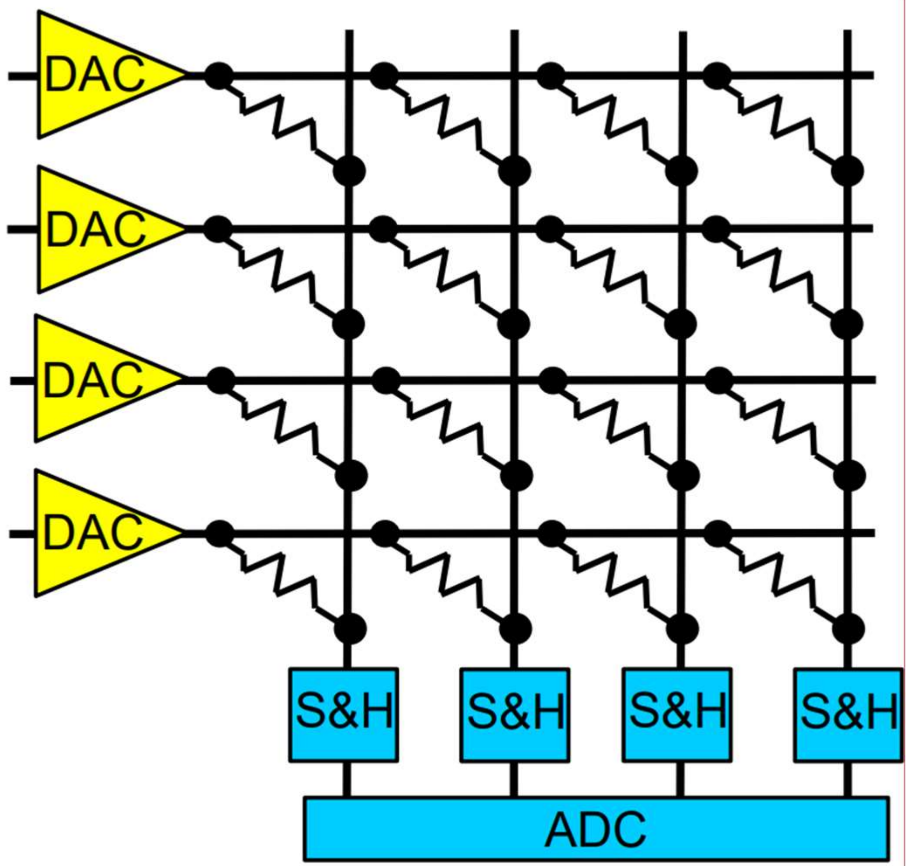
Weight Stationary Dataflow



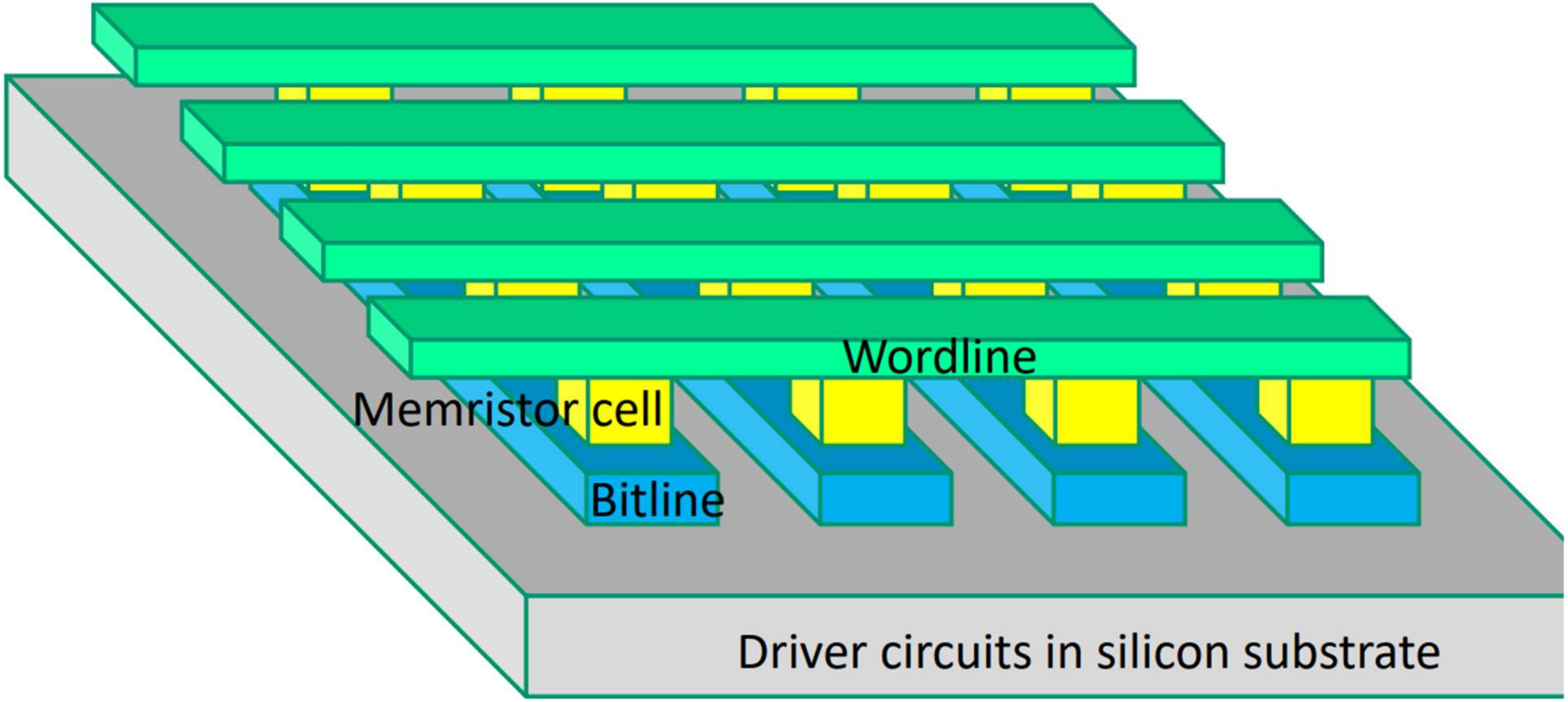
ISAAC, ISCA 2016

Crossbar for vector-matrix multiplication

- **A grid of resistances and wires**
  - The input voltages are provided on the wordlines (horizontal)
  - These voltages are seen by all the columns (bitlines)
  - Each column represents a different neuron
  - Each column sees the same set of inputs, but computes a different dot-product
  - Why ADC/DAC/S&H ?

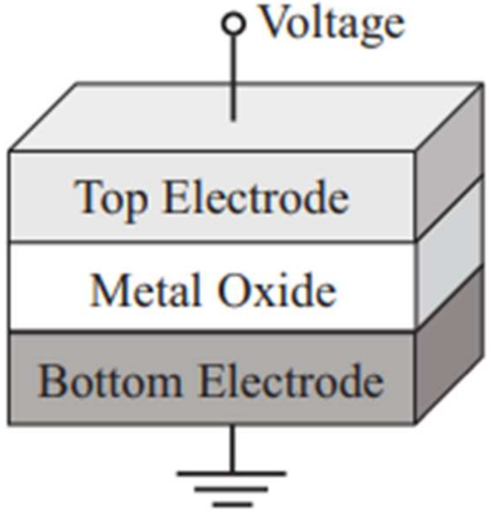


Physical view of a memristor crossbar array



Memristor

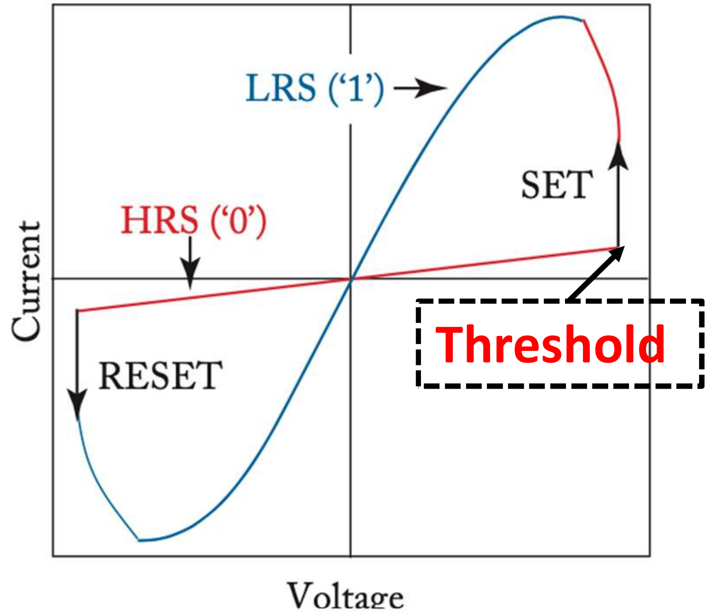
- **Metal-insulator-metal (MIM)** structure
  - The structure of a ReRAM cell
- ReRAM cell Switching states
  - High resistance state (HRS)
  - Low resistance state (LRS)
  - Represent the logic “0” and “1”

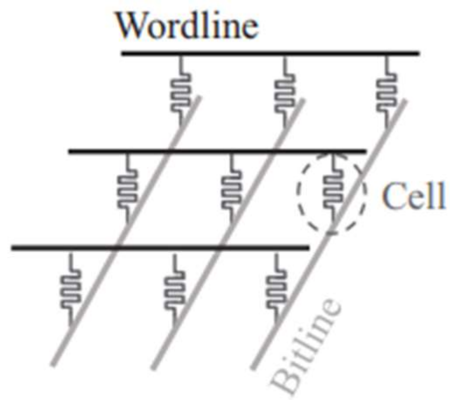


Chi et al., ISCA 2016

Memristor

- **Current-voltage (I-V) bipolar switching**
  - SET operation: switching a cell from HRS (“0”) to LRS (“1”)
  - To **SET** the cell needs a **positive voltage** to generate sufficient write current
  - To **RESET** the cell, a **negative voltage** with proper magnitude is necessary
  - The endurance of ReRAM is 10^{12} > PCM 10^{6}-10^{8}
- **ReRAM Crossbar structure**
  - Multi-layer crossbar structure
  - Multi-level cell (MLC) -> 7-bit MLC ReRAM with various levels of resistance





Chi et al., ISCA 2016

Memristor Computation

- Use memristors as programmable weights (resistance)
- **Advantages**
  - High Density (< 10 nm x 10 nm size)
  - ~30 X smaller than SRAM
  - 1.5 X smaller than DRAM
- **Non-volatile**
- **Operates at low voltage**
- **Computation within memory (in situ)**
  - Reduce data movement

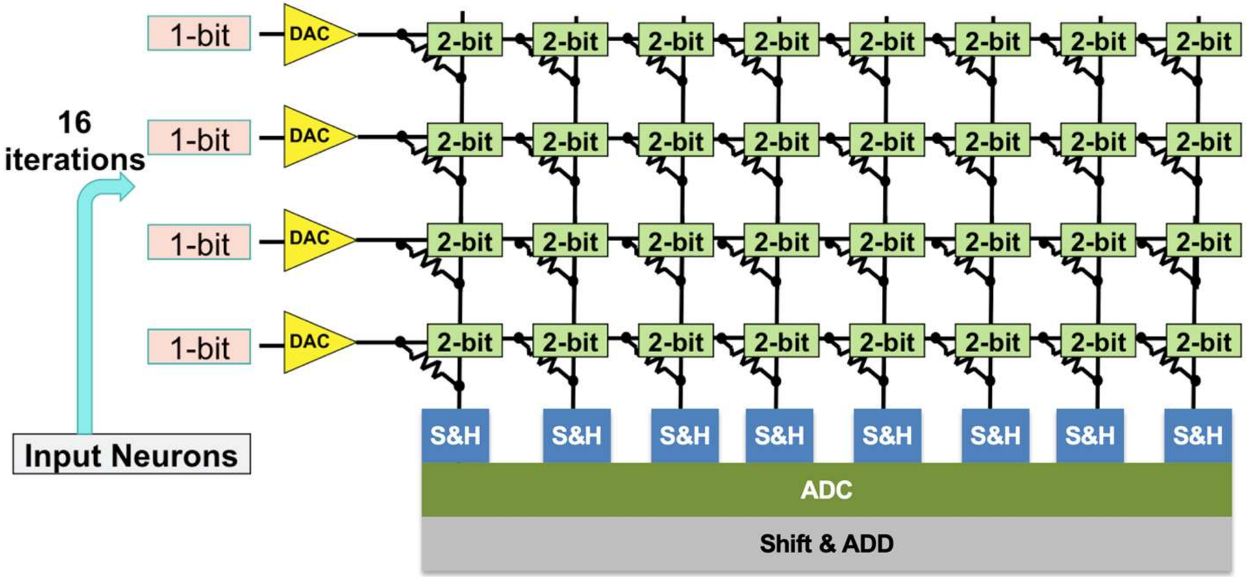
Govoreanu et al., IEDM 2011

Challenge

- **High ADC/DAC area/energy**
  - Expensive analog buffering
  - Significant noise that accumulates across network layers
  - Some ADC overheads increase exponentially with resolution
  - Resolution increases with computational density
- **The number of bits coming out of a bitline is**
  - A function of the bits of info in the voltage (v)
  - The bits of info in the weight (w)
  - The number of rows (R) being added
  - To increase the parallelism and storage density, we want high v, w, and R – expensive high-res ADC

ISAAC (Memristor)

- **16-bit dot-product operation**
  - 8 x 2-bit per memristors
  - 1-bit per cycle computation
  - Trade off area and cycles to address low precision



Shafiee et al., ISCA 2016

Input one bit at a time

- **To bring v down to 1**
  - The grid of resistances is implemented with memristor cells that are sandwiched between the horizontal and vertical wires
  - Provide 16 1-bit inputs over 16 cycles instead of producing a 16-bit input with a very precise voltage
  - Every input is a single 0/1 value
  - The multiplication and addition for each input bit is being performed with the crossbar
  - Results are aggregated with shift-and-adds

Spread the weights

- **To reduce the value of w**
  - Not encode the entire 16-bit value as a precise conductance in a cell
  - Spread the weight across 8 memristor cells in one row
  - Each cell is only responsible for 2 bits
  - Help to bring w down to 2
  - The outputs of 8 columns have to be shifted and added
  - Low bits per cell is good for precision and for ADC efficiency

Few rows per crossbar

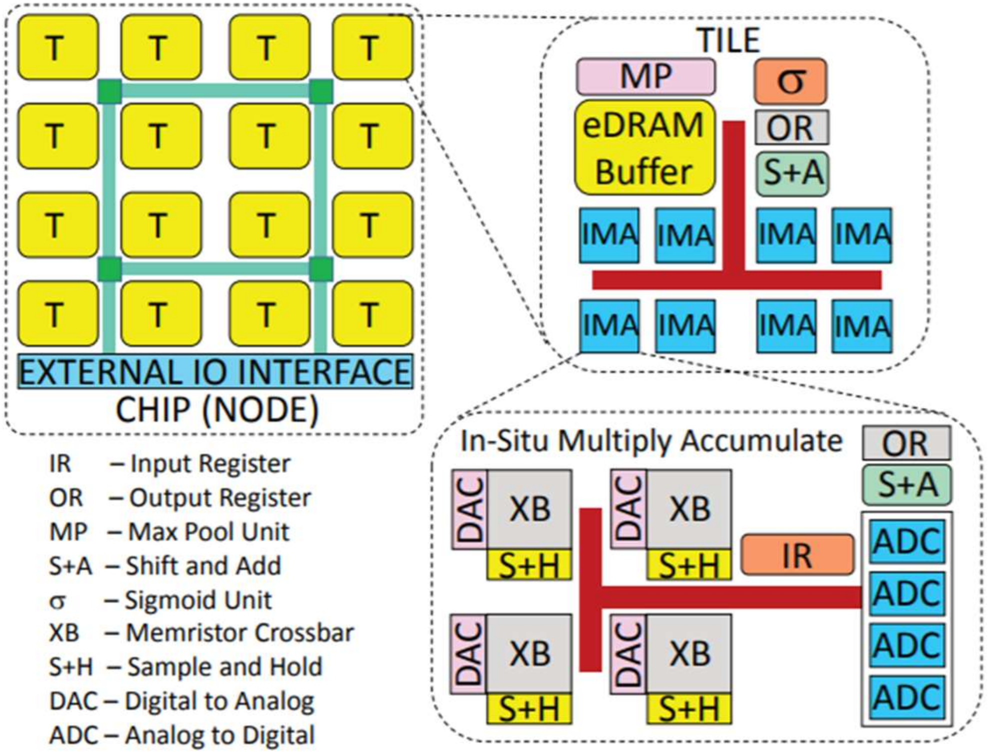
- **To keep the value of R in check**
  - Small crossbars of size 128 x 128
  - Requires to use many small crossbars
  - If a neuron needs more than 128 inputs, it has to spread across multiple crossbars
  - Need to aggregate the partial sums from multiple crossbars to get the final neuron value

Weight encoding

- **To keep A in check**
  - 1-bit inputs at a time, 2-bit cells, and 128 rows – the maximum dot-product value is 384 that requires a 9-bit ADC
  - Most cases, the dot-product value is less than 256 – an 8-bit ADC
  - If the sum of all the weights is greater than 256 -> store the bits in the flipped form
    - Store a 0 for a 3, a 1 for a 2, and a 2 for a 1, and a 3 for a 0

ISAAC (Memristor)

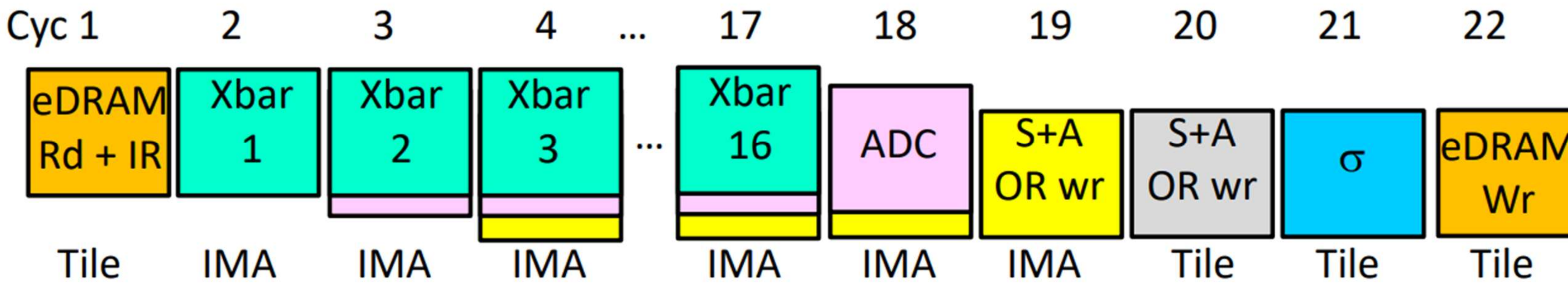
- Eight 128 x 128 arrays per In-situ Multiply Accumulate (IMA)
- 12 IMAs per Tile
- 14 x 12 tiles in ISAAC



Shafiee et al., ISCA 2016

ISACC pipeline

- Example of one operation in a layer I flowing through its pipeline
  - Spatial pipelines – parts of the chip are hard-coded to execute specific layers
  - Latency impact is small (no batching required)



Solutions for analog computation challenges?

- **High ADC/DAC area/energy**
  - 1-bit input at a time (small v)
  - 2-bit cells (small w)
  - Few rows per array (small R)
  - Encoding tricks to produce small numbers
  - Spread the computation across a single xBar, across multiple xBars, and across time to reduce ADC size

Takeaway Questions

- What are challenges of analog computation?
  - (A) ADC/DAC overhead
  - (B) The inference of electronic noise
  - (C) Low storage density on ReRAM
- What are results to bring voltage down to 1 in ReRAM?
  - Improve the performance of the computation
  - Reduce the overhead of ADC/DAC
  - Results are aggregated with shift-and-adds

Spiking Neural Network

- **Spiking neural network (SNN)**
  - SNNs are modeled after operations in the brain
  - Spiking neurons have state and more features than the basic ANN
  - Sending spikes is more energy-efficient than sending 8b or 16b values around
- **Biological neuron**
  - Input = dendrite; output = axon; connection = synapse
  - The neuron fires when its potential reaches a threshold
  - A single neuron may connect to > 10K other neurons; ~100 billion neurons in human brains; ~500 trillion synapses

The Spiking Approach

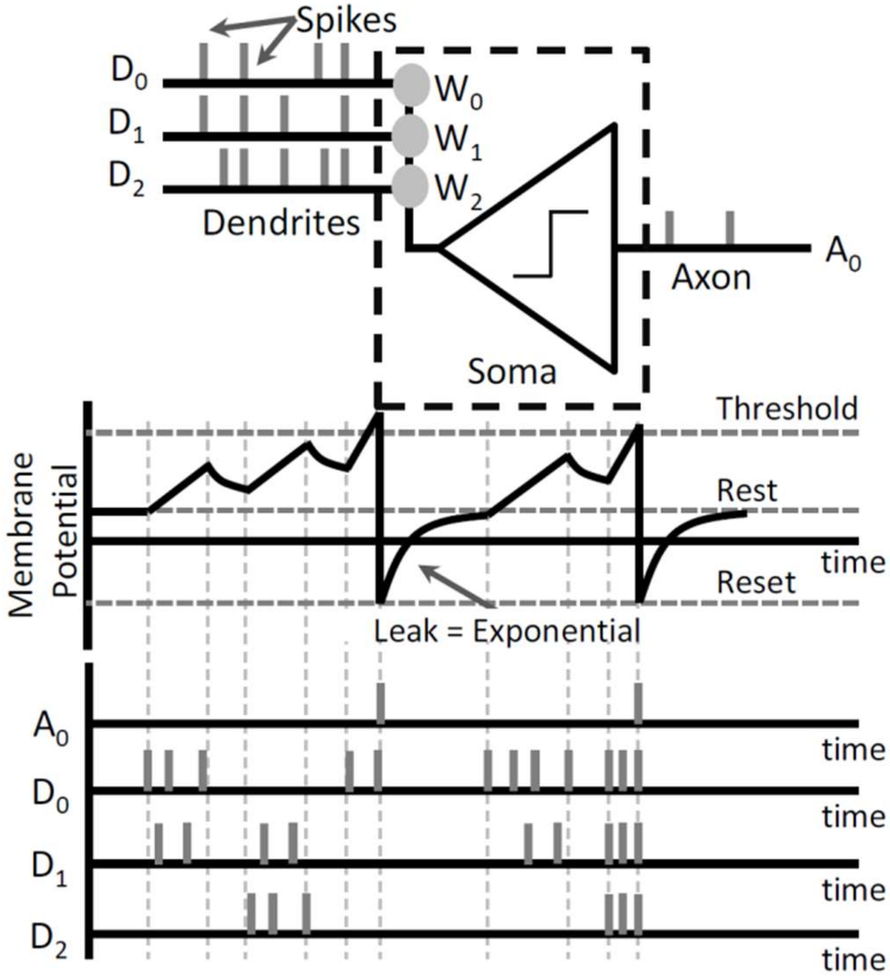
- **Low energy for computation**
  - Only adds, no multiplies
- **Low energy for communication**
  - Depends on spikes per signal
- **Neurons have state**
  - Inputs arrive asynchronously, info in relative timing of spikes
  - The spike trains potentially carry more information
  - Have the potential for higher accuracy

Neuron Models

- **Hodgkin and Huxley**
  - Took differential equations to quantify how ion flow impacts neuron potential from many measurements
  - There are different kinds of neurons that respond differently to the same stimulus
- **Izhikevich**
  - Summarize 20 different neuronal behaviors
  - Biological architectures strive to efficiently emulate these 20 neuron types

LIF Model

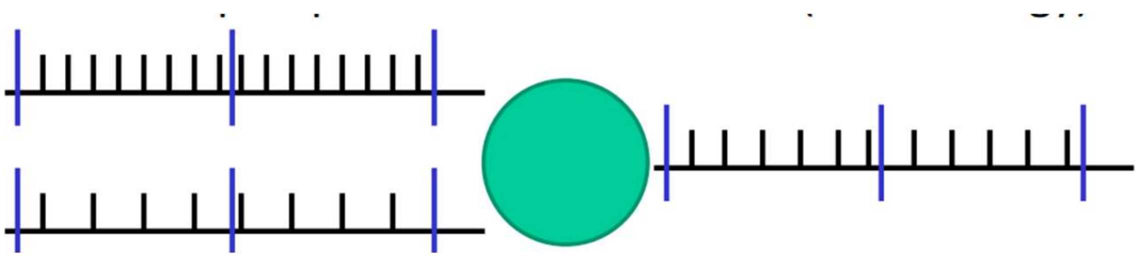
- **Leaky-integrate-fire (LIF) model**
  - **When input spikes show up**
    - The potential is incremented/ decremented based on the synaptic weight for that input
  - **Linear LIF (LLIF) neuron model**
    - The increments/decrements are step functions
    - Unlike the smooth curves
    - There's a threshold potential, reset potential, and a resting potential (typically 0)



Rate vs. temporal codes

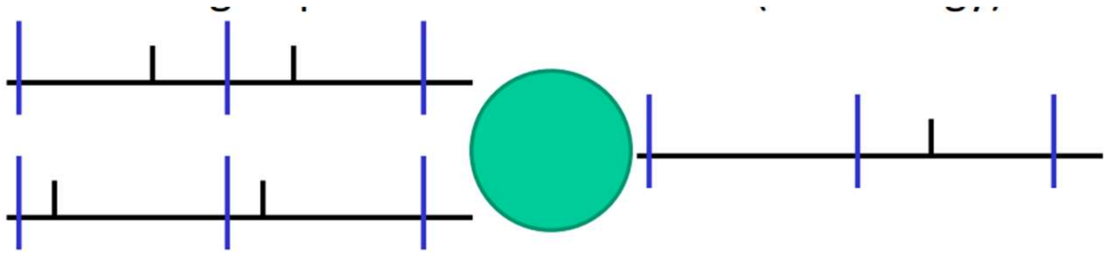
- **Rate code**
  - Information is carried in terms of the frequency of spike
  - Relative timing of spikes on two dendrites is irrelevant
- **Temporal code**
  - Information is carried in terms of the exact time of a spike
  - Time to first spike or a phase code
- **Observation**
  - The same code can apply throughout a multi-layer network
  - A new input is presented after a phase expires

Rate codes



- **Rate code**
  - The first input to the neuron is carrying the value “red” with about 8 spikes per input window
  - The input window is within consecutive blue lines
  - The second input is carrying the value “blue” with 4 spikes per window
  - The rate of the input spikes dictates
    - The potential rises
    - How quickly it reaches the threshold
    - The output spike rate
  - The output frequency = w1 * input_freq1 + w2 * input_freq2

Temporal codes



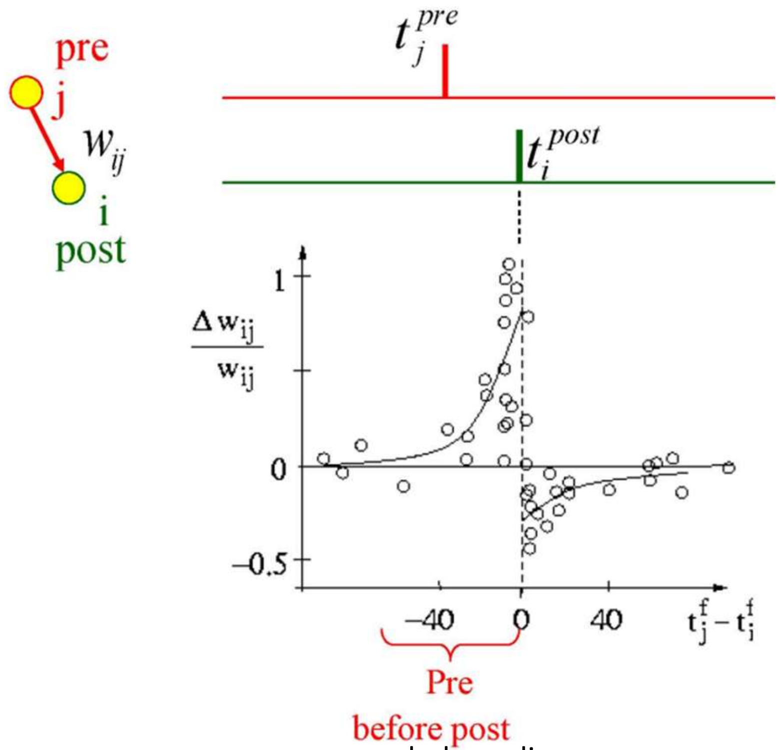
- **Temporal (Spike) code**
  - Reduce the spike rate (low computation and communication energy)
  - An output spike represents the tail end of a weighted cluster of input spikes
  - It’s not the good old ANN equation we understand
  - New learning techniques will have to be developed
  - At the moment, not much exists in this area

SNN Training

- **SNN training**
  - SNNs can be trained with back-prop and rate coding
- **Spike timing dependent plasticity (STDP)**
  - The increment/decrement values depend on when the input spikes arrived
  - If an input spike led to an output spike, that input’s weight is increased
  - If an input spike arrives soon after an output spike, that input’s weight is decreased

STDP

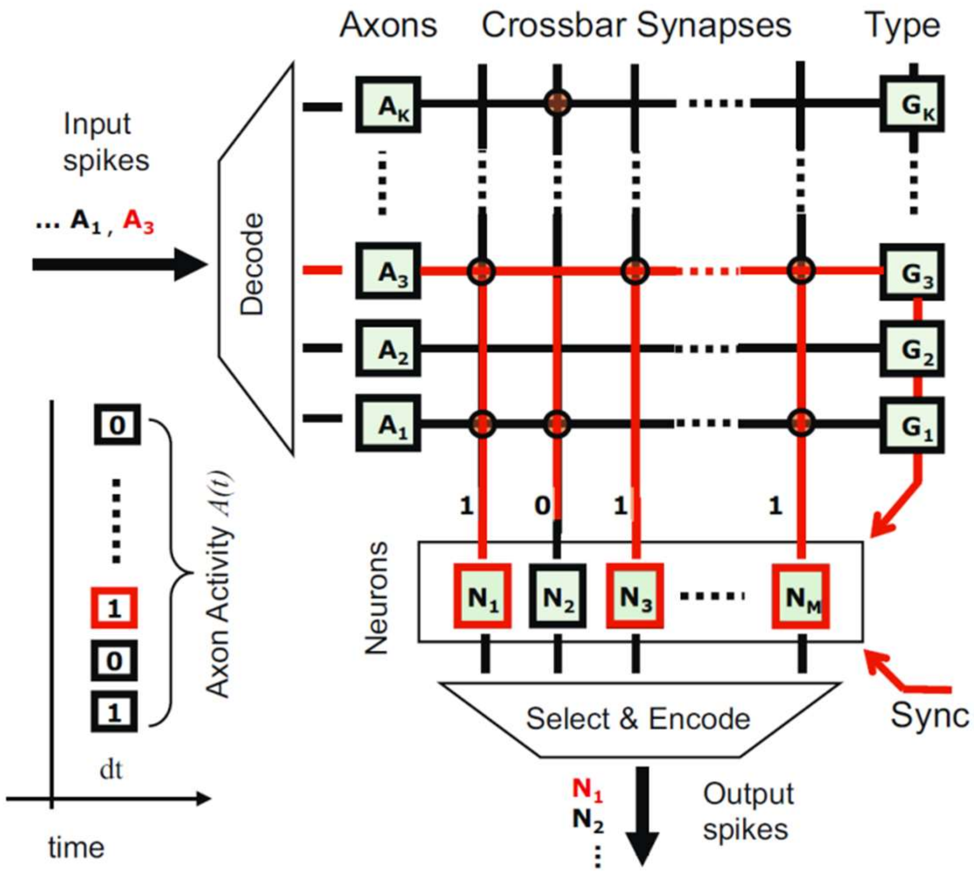
- **STDP is a form of unsupervised learning**
  - The weight adjustments occur independently within each neuron
  - Do not require labeled inputs
  - Over time, some output neuron gets trained to recognize a certain pattern
  - A post processing step
    - Label that neuron as recognizing a cert type of output



www.scholarpedia.org

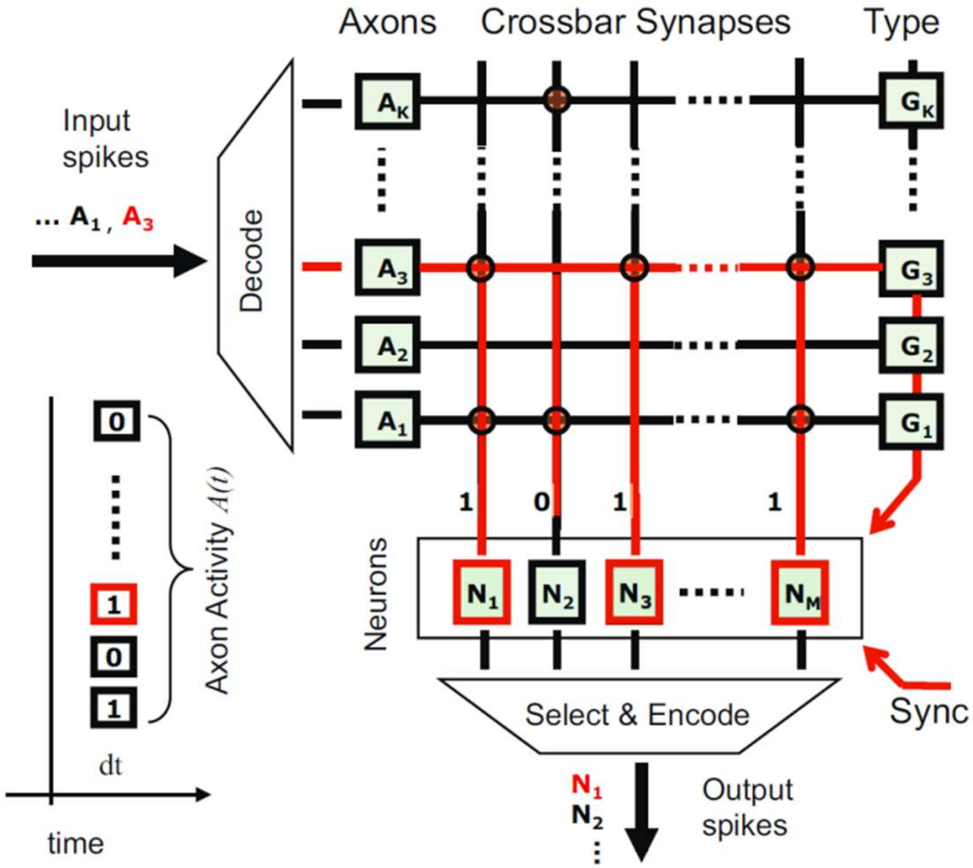
Neuromorphic Accelerator

- **IBM TrueNorth**
  - 5.4 billion transistors
  - Based on **LLIF neuron model**
  - The spikes are processed in a tick and arrive on the left
  - The spike on A3 should be seen by all the neurons that are connected to that axon
  - The grid is storing bits to indicate if that input axon is connected to that neuron



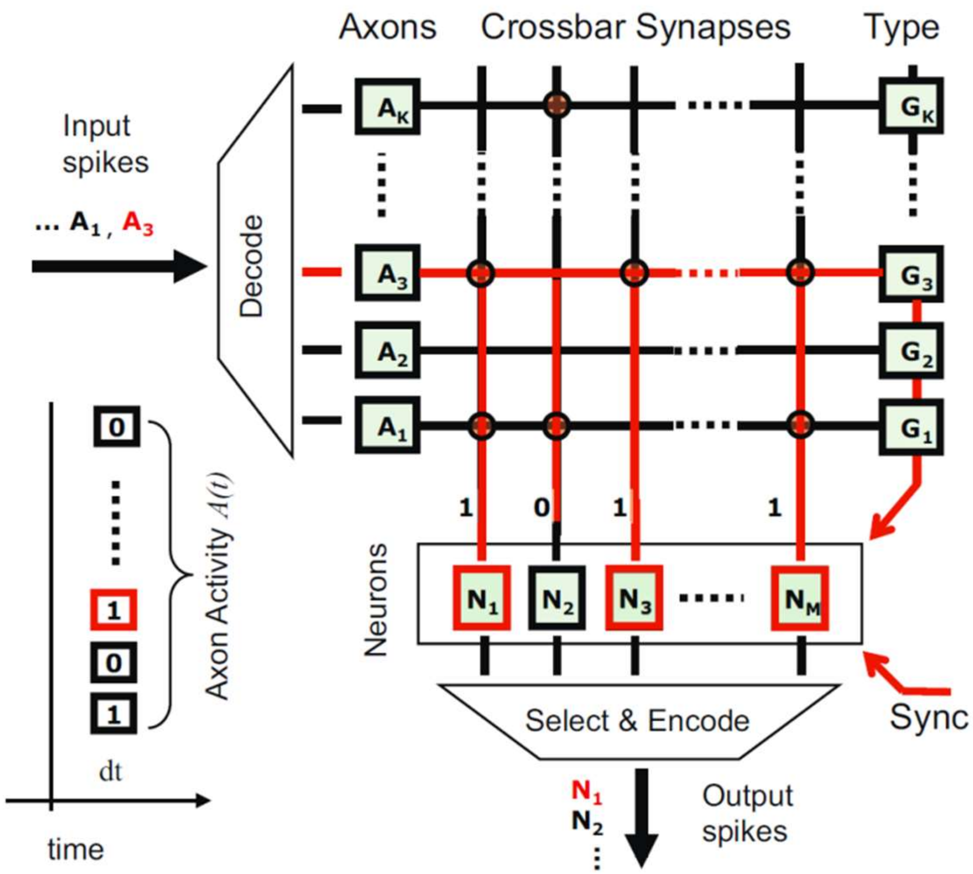
Neuromorphic Accelerator

- **IBM TrueNorth**
  - Each point on the grid also store the synaptic weight for that connection
  - Each neuron only stores 4 weights to save data storage
  - **Classified weights**
    - Strongly Excitatory (a weight between 128 and 255)
    - Weakly Excitatory (0 - 128)
    - Weakly Inhibitory (-128 - 0)
    - Strongly Inhibitory (-255 – -128)

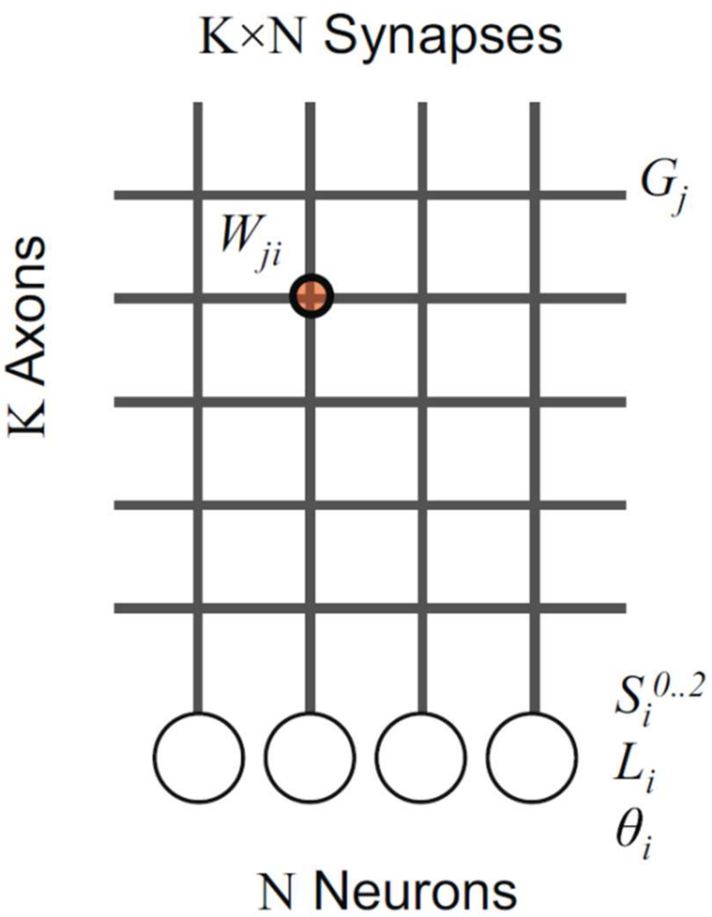


Neuromorphic Accelerator

- **IBM TrueNorth**
  - Store a 2-bit value to indicate which of the 4 weights they should use
  - To reduce the storage, all the connections in a row share the same 2-bit value
  - An input axon will be strong excitatory to all the neurons



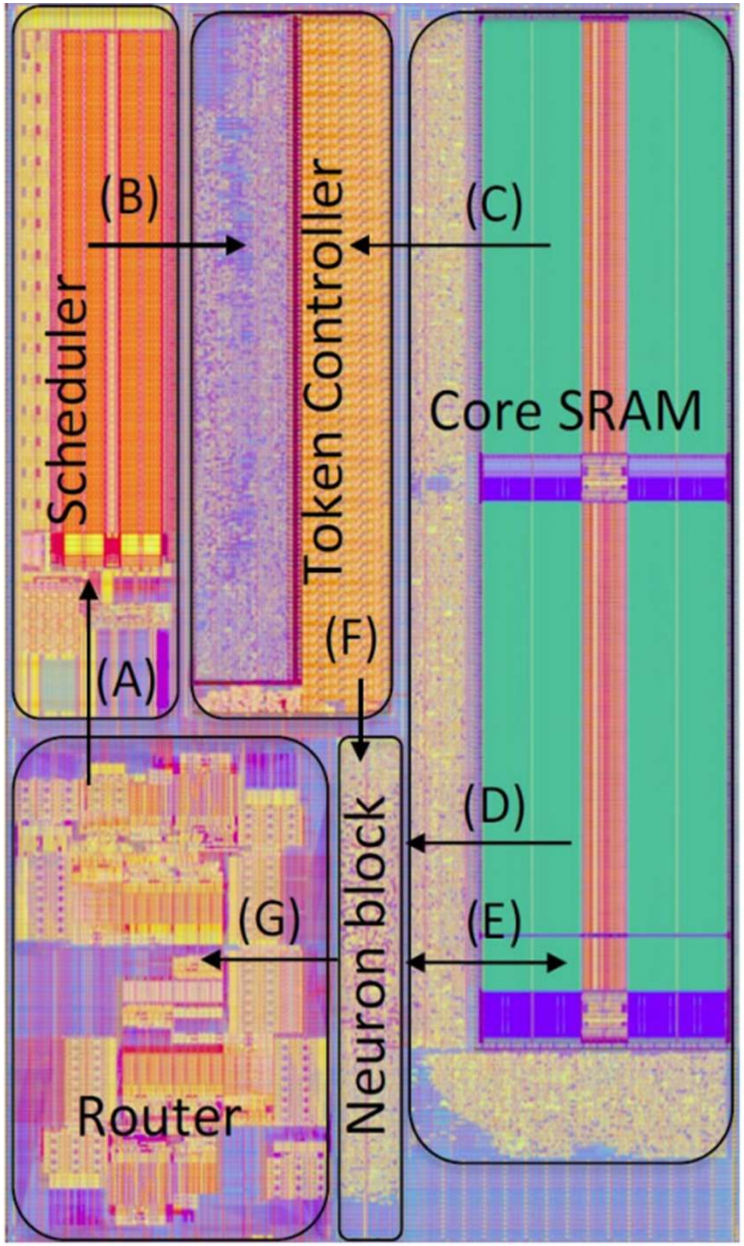
TrueNorth Core (Axonal Approach)



Name	Description	Range
W_{ji}	Connection between axon j and neuron i	0,1
G_j	Axon type	0,1, 2
$S_i^{0..2}$	Synapse values	-256 to 255
L_i	Leak	-256 to 255
θ_i	Threshold	1 to 256

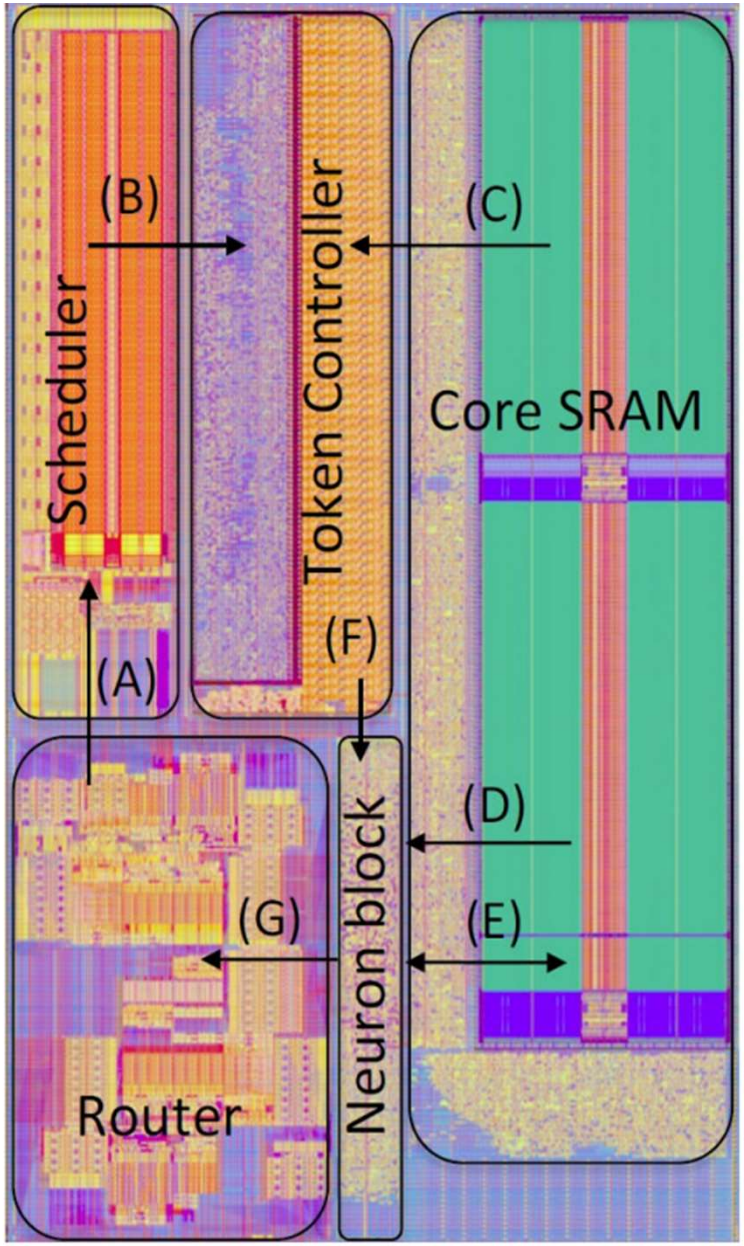
TrueNorth Core (Dendritic)

- **TrueNorth Core (Dendritic approach)**
  - Very low power per spike
  - Compress weights with quantization
  - Axon type sharing
  - Use a mix of async. and sync. circuit
  - **Asynchronous circuit**
    - Rely on handshakes to wake up and perform work when there is an input
    - Sit idle and not burn power when there is no work (like clock gating in sync. circuit)



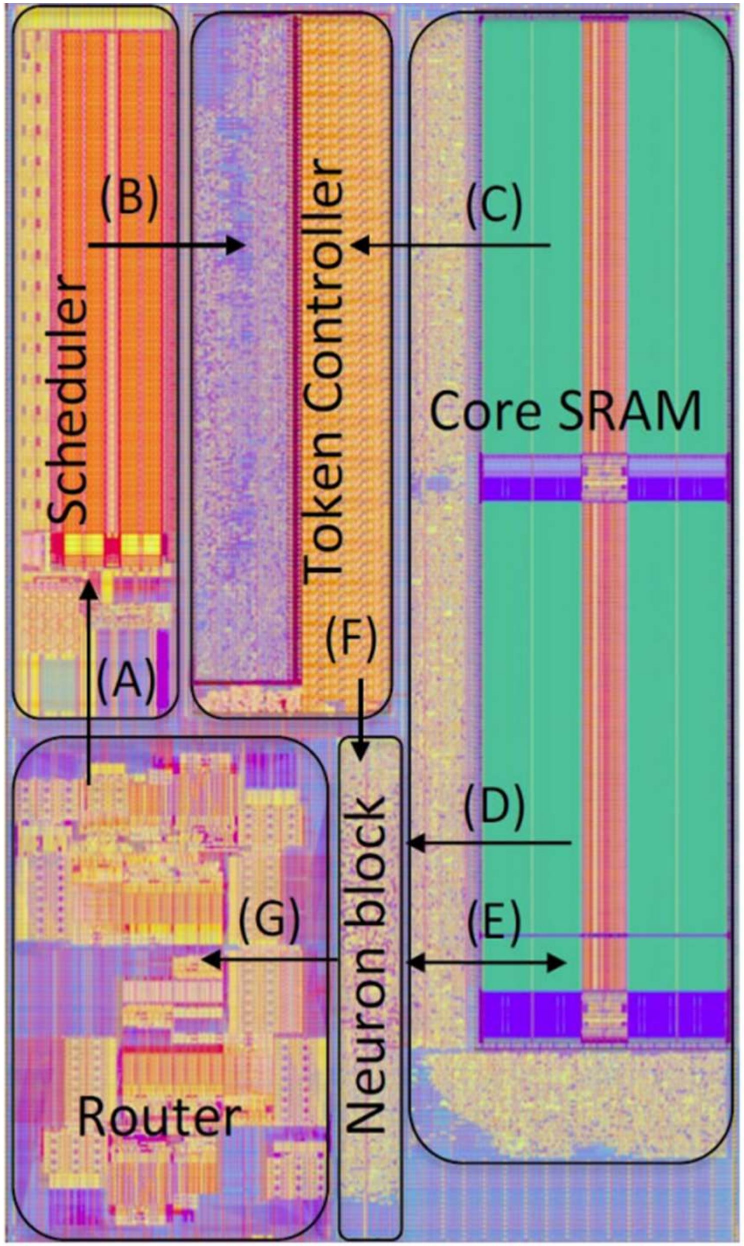
TrueNorth Core (Dendritic)

- **TrueNorth Core (Dendritic approach)**
  - Use ultra high voltage (VT) transistor to reduce leakage (might hurts cycle time)
  - **Asynchronous Circuit**
    - Router
    - Scheduler
    - Token Controller
  - **Synchronous Circuit**
    - Neuron block
    - Dissipate power only when it receives instructions from the token controller



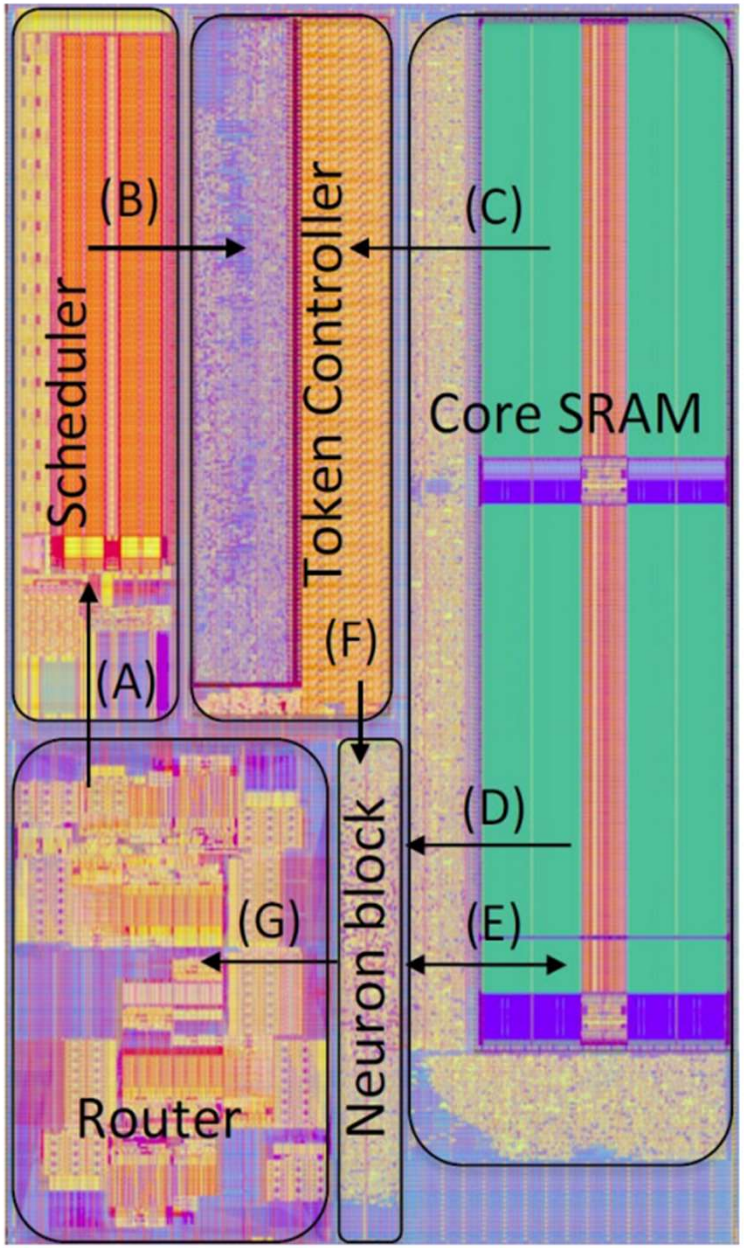
TrueNorth Core (Dendritic)

- **TrueNorth Core (Dendritic approach)**
  - Spikes are placed in a queue in the scheduler when the scheduler receive spikes
  - An incoming spike may be processed in the next tick (a tick is typically 1 ms) or in one of the next 16 ticks
  - The queue is broken into 16 pieces
  - Each piece has a bit for one of the 256 axons (rows) in that core
  - The incoming spike queue is a 256 x 16 bit vector



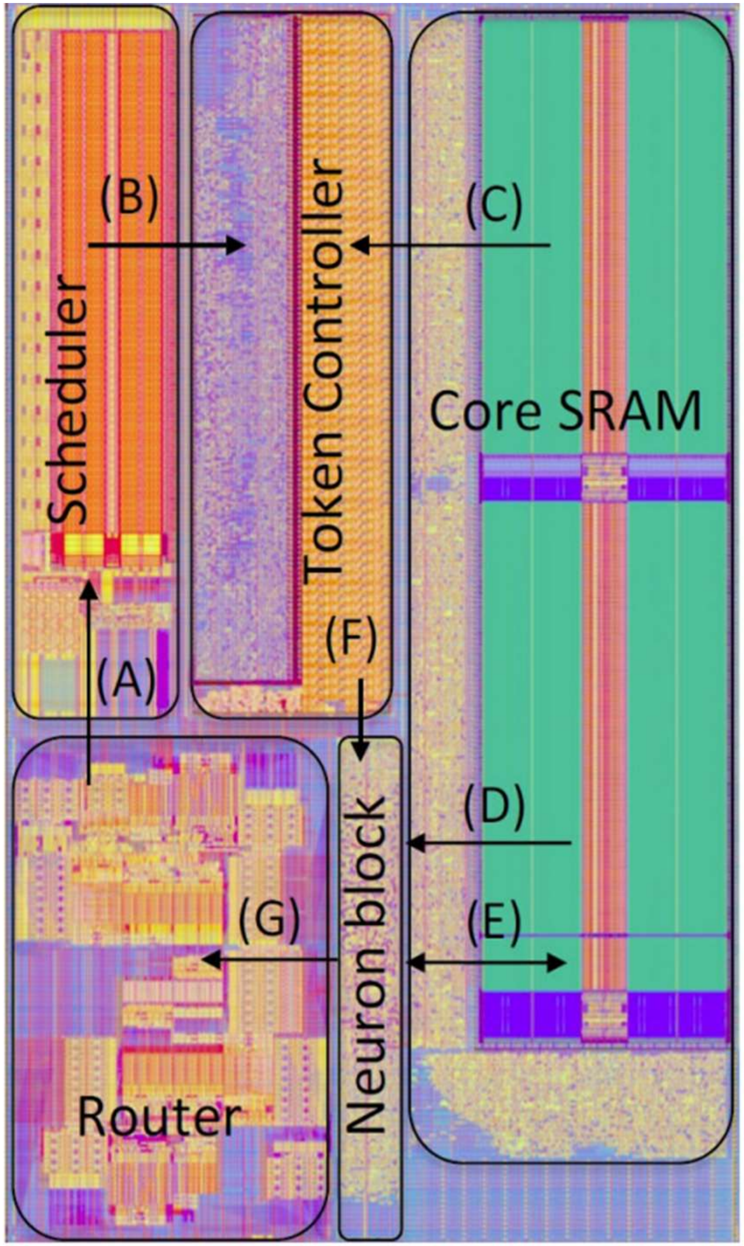
TrueNorth Core (Dendritic)

- **TrueNorth Core (Dendritic approach)**
  - Every tick, the scheduler sends the list of incoming spikes to the token controller (along with axon types)
  - Dendritic approach
    - Sequentially walk through every neuron and process all the dendrites (inputs) for that neuron
  - The token controller first reads all the info for Neuron-1 from the core SRAM
  - Core SRAM is a 410-bit field that stores the 256 1-bit connection



TrueNorth Core (Dendritic)

- **TrueNorth Core (Dendritic approach)**
  - The 256 a-bit connections are ANDed with the incoming spike vector to see if this neuron has any input spikes
  - The token controller picks out the correct weight and packs off an instruction to the neuron block
  - Once the token controller has dealt with all input spikes for that neuron, it sends off a final leak instruction
  - The leak is added to the potential
  - The generated spike is sent to the router, the final potential is written back to the core SRAM



Summary of TrueNorth

- **Design Principle**
  - Focus on low power (speed is secondary)
  - Low power with asynchronous circuits
  - Low power with high-Vt circuits
  - A dendritic approach leads to fewer SRAM reads/updates and is more deterministic

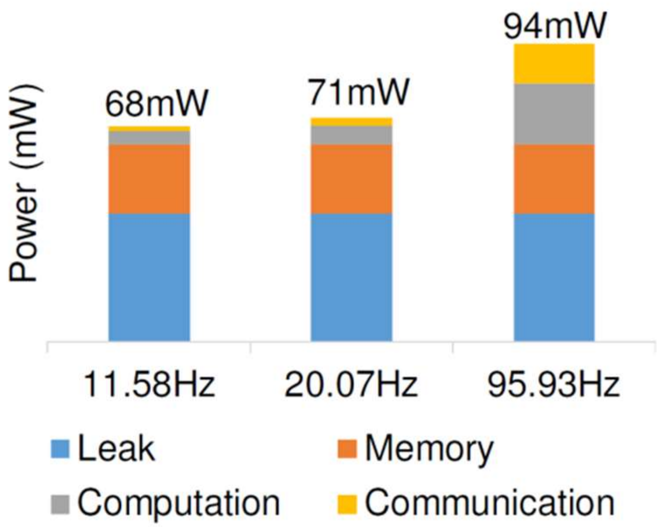
Receiving Spikes

- **Spikes arrive at the Scheduler**
  - Spikes are stored in a 256 x 16 SRAM grid to indicate axon and time of the spike
- **The token controller receives spikes in a tick**
  - It sequentially walks through 256 neurons
  - The dendritic approach makes the latency and SRAM accesses more deterministic
  - It reads a 410-bit word from SRAM for that neuron
  - Based on connectivity and input spikes, instructions are sent to Neuron block

Neuron block

- **Neuron block**
  - Neuron computations are performed here
  - A leak is introduced every tick for every neuron
  - After thresholding, a spike may be triggered
  - Synchronous circuit
    - It is only active when it receives instructions from the token controller

TrueNorth Power Breakdown



- **The chip is dissipating**
  - Roughly 65 mW in leakage and memory all the time
  - Computation/communication energy rises linearly
- **The throughput of super-large TrueNorth**
  - A max throughput of ~60 giga-synaptic ops per second
  - 100 times lower than DaDianNao that uses maybe 1/10 of transistors over the TrueNorth
  - No SIMD-ness, limited wiring, sequentially walks every neurons

TrueNorth Core (Dendritic)

- **Dendritic approach**
  - More deterministic and lead to fewer SRAM accesses than Axonic approach
- **Axonic approach**
  - Walk through each input axon and increment the potentials of the neurons connected to that axon
  - Read neuron’s data structures multiple times

ML vs. Neuroscience

- In MICRO’15 paper by Du et al.:
- **The accuracy**
  - Neuroscience-inspired approach (SNN + STDP) vs. machine learning inspired approach (MLP + BP (Back-prop))
- **The cost of hardware**
  - SNN vs. MLP
- **When should a designer use hardware SNN or MLP ?**
- **Workload**
  - MNIST: 28 x 28 images

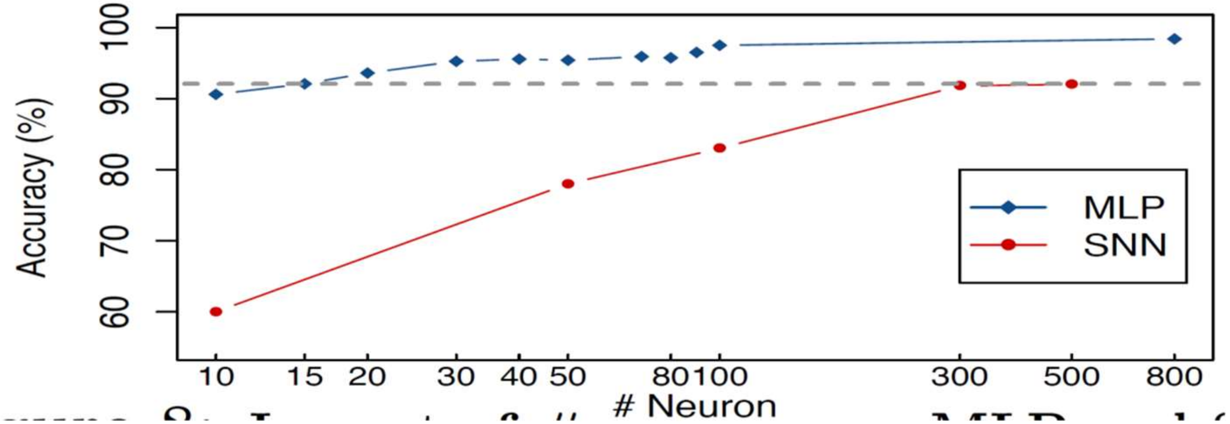
SNN models

- One layer with 300 neurons
- Each neuron receives inputs from all 784 pixels
- Weights are either 8b or 12b
- Use rate coding, but convert the 8-bit input value into just 0-10 spikes
- Spike intervals are drawn from a Gaussian distribution
- Trained with STDP

MLP model

- 8b fixed-point weights, inputs, operators
- 784 input neurons, 100 neuron hidden layer, 10 neurons in output layer
- Sigmoid activation function
- Training with back-propagation
- Use 8-bit precision for all computations

Accuracy on MNIST



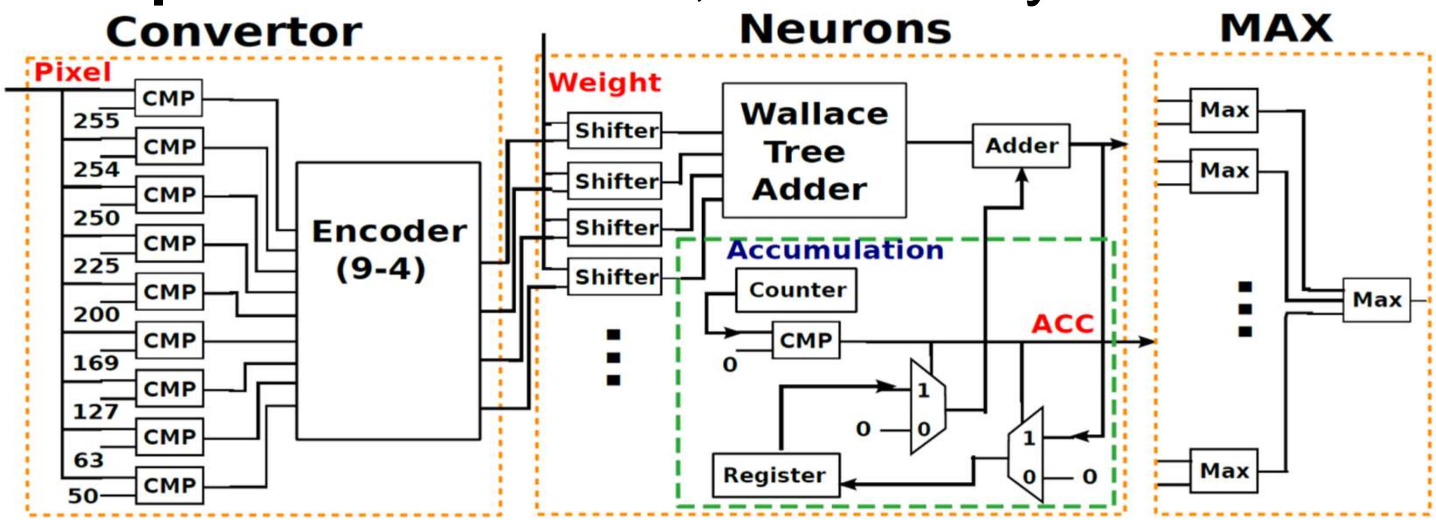
- **Accuracy of MLP with 100 hidden neurons**
  - 97.65%
- **Accuracy of SNN + STDP**
  - 91.82% with 300 neurons
  - Starting with SNN + STDP, but computing error function and applying gradient descent -> 95.4%
  - Most of the 6% gap can be bridged by using back-prop instead of STDP
  - The rest can be attributed to be ANN’s better activation function and use high precision math

Hardware complexity (MLP)

- **MLP implementation (Expanded design)**
  - Every neuron has its own dedicated hardware
  - Only work for small networks
  - There is a multiple and register for every synapse, a multi-input adder, and a look-up table for activation

Hardware complexity (SNN)

- **SNNwot**
  - An **encoder** to convert the 8-bit pixel value into a number between 0-10 spikes
  - A simple 4bx12b **multiplier** at the neuron, followed by an **adder**
  - A **max circuit** to figure out the winning neuron (max of potential or spike)



Expanded Design vs. SNNwot

- The 4x12 multiplier (SNNwot) is 8x cheaper than 8x8 multiplier (MLP)
- The SNN needs 3x more ALUs and storage than MLP because SNN has more neurons
- SNN takes ~2x less area than MLP

Table 5: Hardware Characteristics of SNN (4x4-20) and MLP (4x4-10-10).

Type	Area (mm^2)	Delay (ns)	Power (W)	Energy (nJ)
SNN	0.08	1.18	0.52	0.63
MLP	0.21	1.96	0.64	1.28

Conclusion

- MLP achieves higher accuracy over SNN
- The gap of the accuracy can be bridged with
  - Back-prop, sigmoid, better input encoding, etc.
- SNN has an advantage in on-line learning and for spatially expanded designs

Takeaway Questions

- What does rate code carry spikes?
  - (A) In terms of the exact time of a spike
  - (B) In terms of the exact location of a spike
    (C) In terms of the frequency of a spike
- How does Spike timing dependent plasticity (STDP) work?
  - (A) If an input spike led to an output spike, that input's weight is decreased
  - (B) The increment/decrement values depend on when the input spikes arrived
  - (C) If an input spike arrives soon after an output spike, that input's weight is increased

Takeaway Questions

- How to improve the accuracy of SNN + STDP?
  - (A) Applying gradient descent
  - (B) Using back-prop.
  - (C) Using better activation