Binary Neural Networks & TinyML HW

Intelligent Architectures: 5LIL0

April 1, 2025

Dr. Federico Corradi, Assistant Professor, Neuromorphic Edge Computing Systems Lab

Department of Electrical Engineering, Electronic Systems Group

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Outline

Last few lectures

GeMM & CNN processors enhancements in ASIC, CPU, GPU

- Sparsity
  - Sparse CNN engine
  - Efficient Inference Engine
  - NVIDIA Ampere GPU
- Quantization
  - Supporting mixed-precision quantization
  - Envision (KU) Leuven

Last few lectures

GeMM & CNN processors enhancements in ASIC, CPU, GPU

- Sparsity
  - Sparse CNN engine
  - Efficient Inference Engine
  - NVIDIA Ampere GPU
- Quantization
  - Supporting mixed-precision quantization
  - Envision (KU) Leuven

Now

Canvas quizzes:

- Lecture 11: Parallelization & Stationarity in DNN HW
- Lecture 12: Sparsity & Quantization in DNN HW

Outline

Lesson Plan

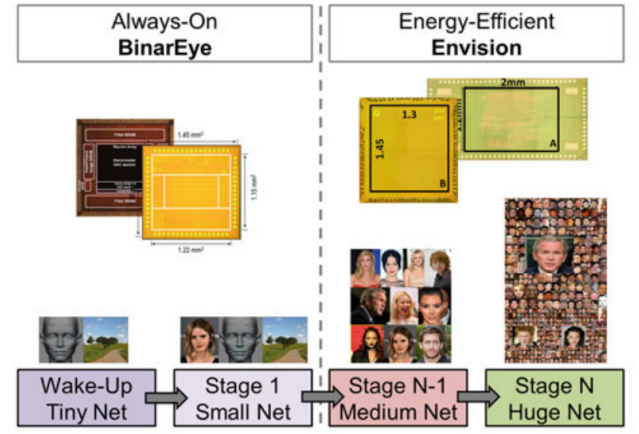
Extreme of quantization

- Binary neural networks
- Binareye:
  1. digital KU Leuven
  2. mixed-signal (KU Leuven & Stanford)

Learning objectives

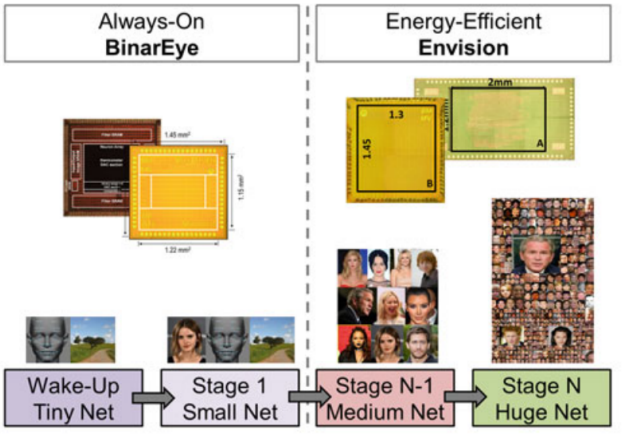
- You will be able to execute BNN inference.
- You will be able to analyzed BNN neural network accelerators and describe their architectures.

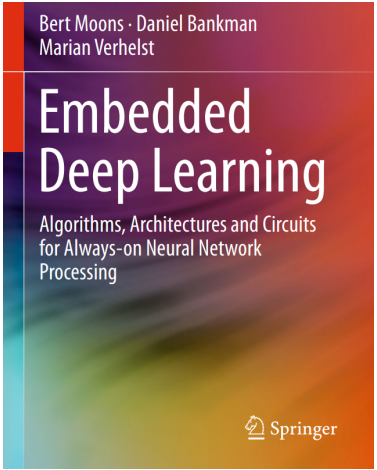
The big picture



[[Moons et al, 2018]]

The big picture

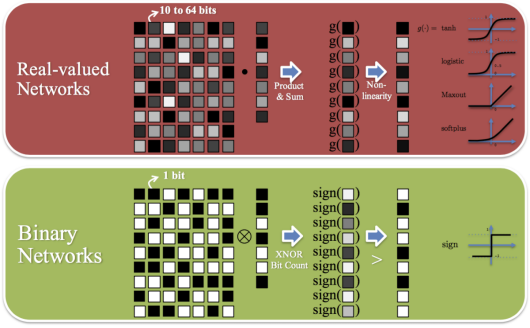




[[Moons et al, 2018]]

Outline

Binary Neural Networks

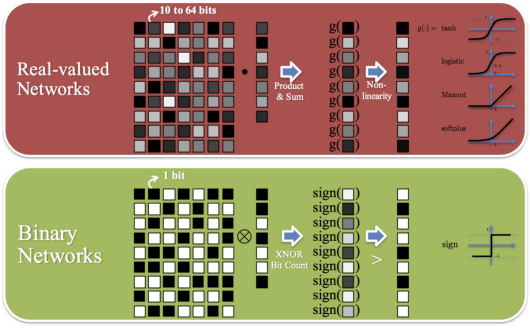


[Courbariaux, 2016]

BNN

- push precision to the extreme

Binary Neural Networks

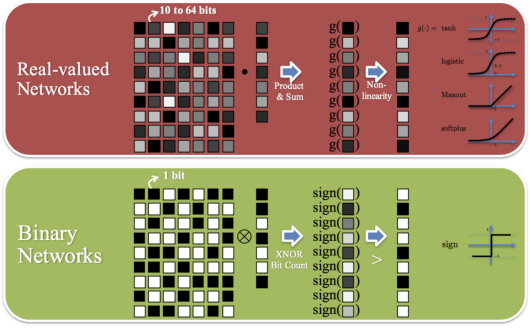


[Courbariaux, 2016]

BNN

- push precision to the extreme
- 1-bit weights

Binary Neural Networks

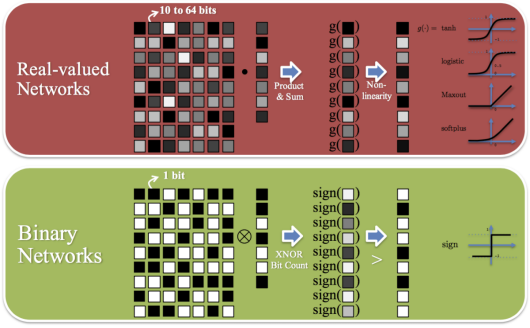


[Courbariaux, 2016]

BNN

- push precision to the extreme
- 1-bit weights
- 1-bit activation

Binary Neural Networks

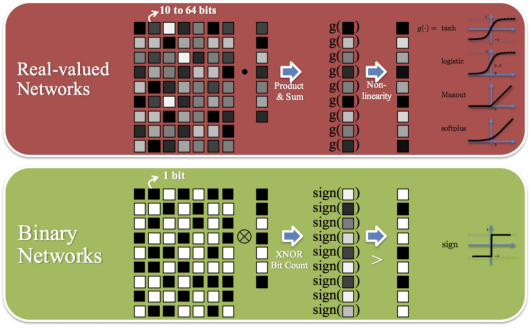


[Courbariaux, 2016]

BNN

- push precision to the extreme
- 1-bit weights
- 1-bit activation
- threshold operation (step function)

Binary Neural Networks



[Courbariaux, 2016]

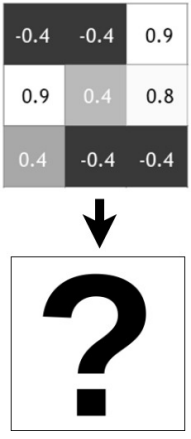
BNN

- push precision to the extreme
- 1-bit weights
- 1-bit activation
- threshold operation (step function)
- multiplication $\rightarrow$ XNOR operation

A	B	A $\odot$ B (XNOR)
0	0	1
0	1	0
1	0	0
1	1	1

Table: Truth table for the XNOR operation

Binary neural networks: extreme quantization

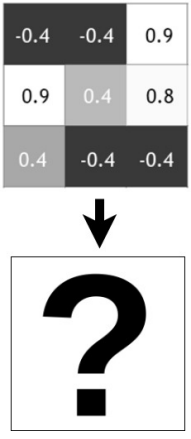


The popular definition of the binarization function is given as follows:

$$Q_w(w) = \alpha b_w, Q_a(a) = \beta b_a \tag{1}$$

where:

Binary neural networks: extreme quantization



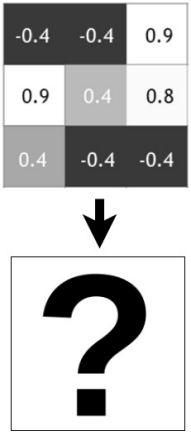
The popular definition of the binarization function is given as follows:

$$Q_w(w) = \alpha b_w, Q_a(a) = \beta b_a \tag{1}$$

where:

- b_w are the binary weights (i.e. $sign(w)$)

Binary neural networks: extreme quantization



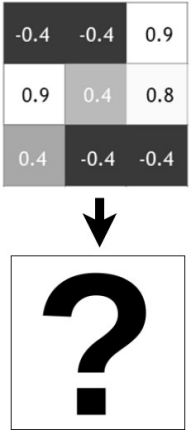
The popular definition of the binarization function is given as follows:

$$Q_w(w) = \alpha b_w, Q_a(a) = \beta b_a \quad (1)$$

where:

- b_w are the binary weights (i.e. $sign(w)$)
- b_a are the binary activations (i.e., $sign(a)$)

Binary neural networks: extreme quantization



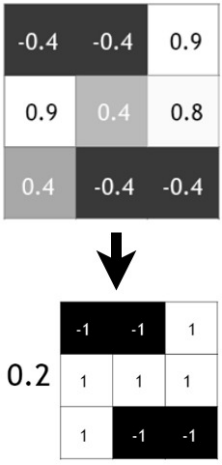
The popular definition of the binarization function is given as follows:

$$Q_w(w) = \alpha b_w, Q_a(a) = \beta b_a \quad (1)$$

where:

- b_w are the binary weights (i.e. $sign(w)$)
- b_a are the binary activations (i.e., $sign(a)$)
- $\alpha = \frac{1}{n}||w||_{L1}$, $\beta = \frac{1}{n}||a||_{L1}$

Binary neural networks: extreme quantization



The popular definition of the binarization function is given as follows:

$$Q_w(w) = \alpha b_w, Q_a(a) = \beta b_a \tag{1}$$

where:

- b_w are the binary weights (i.e. $sign(w)$)
- b_a are the binary activations (i.e., $sign(a)$)
- $\alpha = \frac{1}{n}||w||_{L1}$, $\beta = \frac{1}{n}||a||_{L1}$

The sign function is used for binarization

$$sign(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases}$$

Binary neural networks: forward path

In full precision, a convolution operation is expressed as:

$$z = \sigma(w \otimes a) \tag{2}$$

Binary neural networks: forward path

In full precision, a convolution operation is expressed as:

$$z = \sigma(w \otimes a) \tag{2}$$

With the binarized weights and activations, the vector multiplication in forward propagation can be reformulated a:

$$z = \sigma(Qw(w) \otimes Qa(a)) = \sigma(\alpha\beta(b_w \odot b_a)) \tag{3}$$

where $\odot$ denotes the inner product for vectors with bitwise operation XNOR-Bitcount.

XNOR Operation in Binary Neural Networks

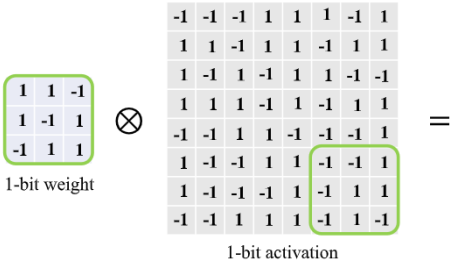
Binary Neural Networks (BNNs) use binary values instead of full-precision weights. Multiplication between bipolar values (± 1) can be replaced with a bitwise XNOR operation:

A (bipolar)	B (bipolar)	A (binary)	B (binary)	A XNOR B	Result (bipolar)
-1	-1	0	0	1	$+1$
-1	$+1$	0	1	0	-1
$+1$	-1	1	0	0	-1
$+1$	$+1$	1	1	1	$+1$

Table: Mapping of bipolar multiplication to binary XNOR logic

Key idea: XNOR yields 1 when both binary values match $\Rightarrow$ equivalent to multiplying equal-sign bipolar values.

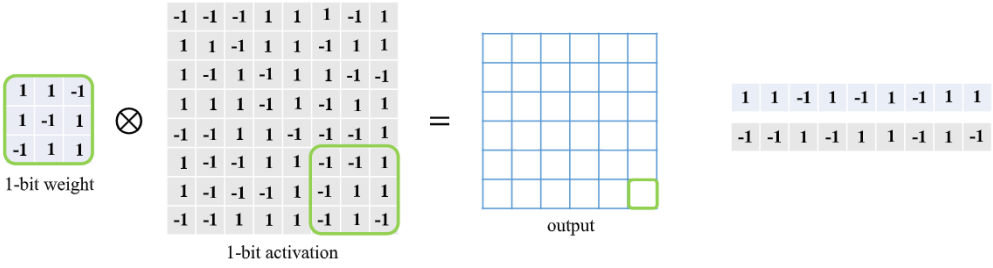
Binary neural networks: concept



[Qin et al 2020]

stride = 1, padding = 0

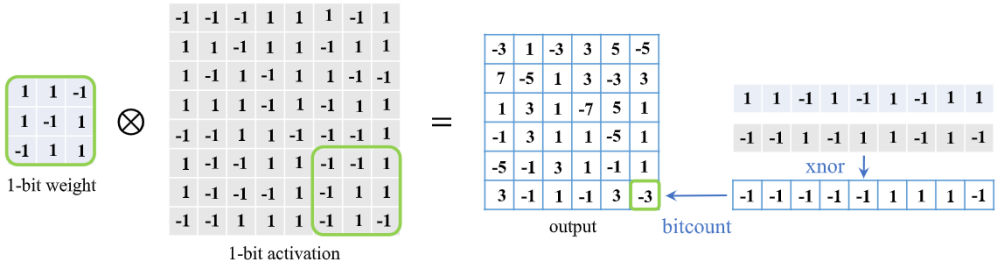
Binary neural networks: concept



[Qin et al 2020]

stride = 1, padding = 0

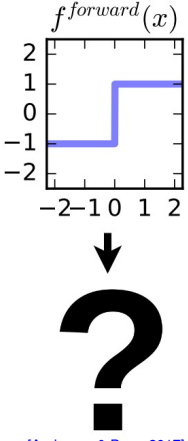
Binary neural networks: concept



[Qin et al 2020]

stride = 1, padding = 0

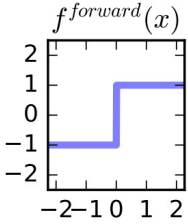
Binary neural networks: backward path

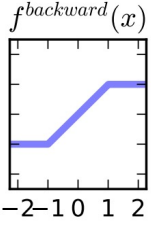


[Anderson & Berg, 2017]

- step function is not differentiable

Binary neural networks: backward path

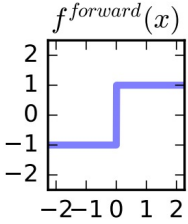




[Anderson & Berg, 2017]

- step function is not differentiable
- straight-through estimator (STE)

Binary neural networks: backward path



[Anderson & Berg, 2017]

- step function is not differentiable
- straight-through estimator (STE)
- this preserves the gradient's information and cancels large gradients

Binary neural networks: training algorithm

Algorithm 1 Training an L-layers CNN with binary weights:

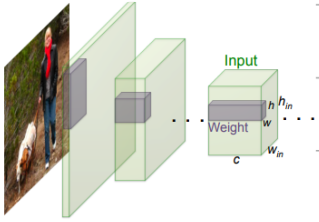
Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

Binarizing weight filters:
for $l = 1$ to L **do**
  for k^{th} filter in l^{th} layer **do**
    $\mathcal{A}_{lk} = \frac{1}{n}\|\mathcal{W}^t_{lk}\|_{\ell 1}$
    $\mathcal{B}_{lk} = \text{sign}(\mathcal{W}^t_{lk})$
    $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$
$\hat{\mathbf{Y}} =$ **BinaryForward**$(\mathbf{I}, \mathcal{B}, \mathcal{A})$ //
$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} =$ **BinaryBackward**$(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$
$\mathcal{W}^{t+1} =$ **UpdateParameters**$(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)$ // Any update rules (*e.g.*,SGD or ADAM)
$\eta^{t+1} =$ **UpdateLearningrate**(η^t, t) // Any learning rate scheduling function

[Rastegari et al, 2016]

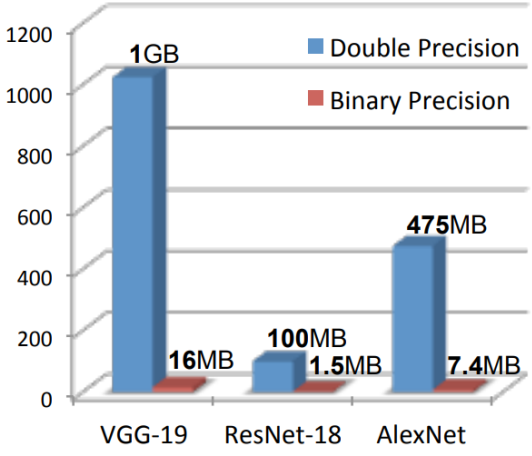
Binary neural networks: Does it work?



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

[Rastegari et al, 2016]

Binary neural networks: Does it work?



[Rastegari et al, 2016]

Binary neural networks: Binareye

Layer	Type	W,H	K	C	Stride
0	IO	32	2	256	1
1	CONV	32	2	256	1
2	CONV	31	2	256	1
3	CONV	30	2	256	1
4	CONV	29	2	256	1
4p	MP	28	-	-	2
5	CONV	14	2	256	1
6	CONV	13	2	256	1
6p	MP	12	-	-	2
7	CONV	6	2	256	1
8	CONV	5	2	256	1
9	FC	4096 – to - 10			

MNIST	CIFAR -10	Face Det. [d]	Owner Det. [e]	3 / 7 Angles
98.85	86	98.1*/95.7+	98.2*/83.3+	99.1/ 98.9

Can I work with 1 bit and have useful NN?

- yes for simple tasks, with slight accuracy drop, and not for all tasks! [Moons et al, 2018]

Thermometer Coding

Definition: Thermometer coding is a unary encoding scheme where the number of ones represents the value. It is often used for low-precision quantization in neural networks due to its robustness to bit-flips.

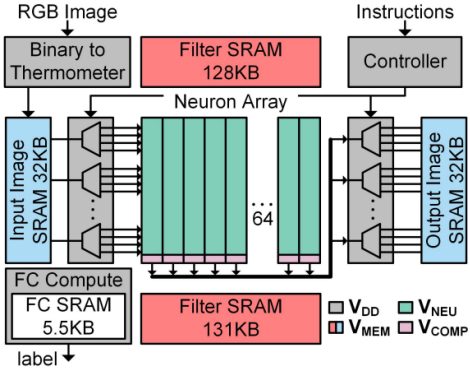
Example (3-bit resolution):

Value	Thermometer Code
0	000
1	100
2	110
3	111

Properties:

- Only one transition from 0 to 1
- Robust to single-bit errors
- Useful for quantized activations and analog-to-digital conversions

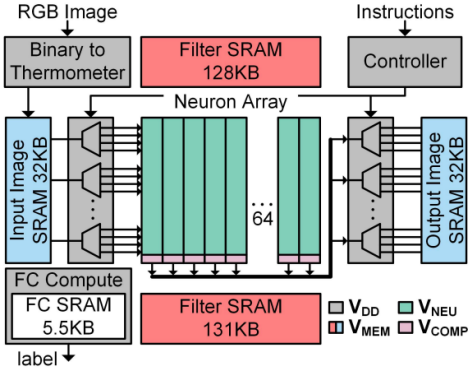
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems

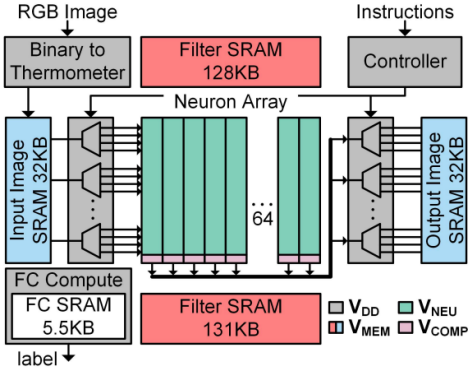
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems
- a small input image (32x32)

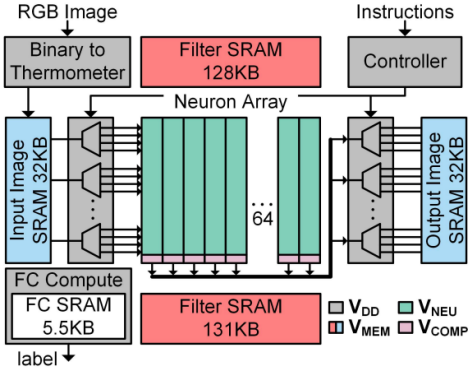
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems
- a small input image (32x32)
- it has a data path to execute a binary convolutional neural network

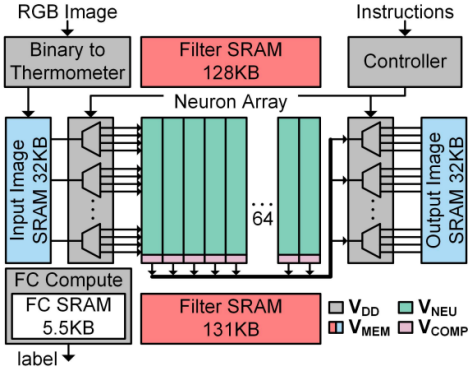
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems
- a small input image (32x32)
- it has a data path to execute a binary convolutional neural network
- weight in SRAM blocks (top/bottom), + inputs and output SRAM

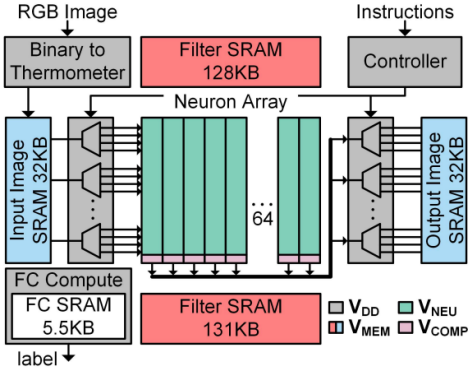
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems
- a small input image (32x32)
- it has a data path to execute a binary convolutional neural network
- weight in SRAM blocks (top/bottom), + inputs and output SRAM
- the controller can only do 3 instructions

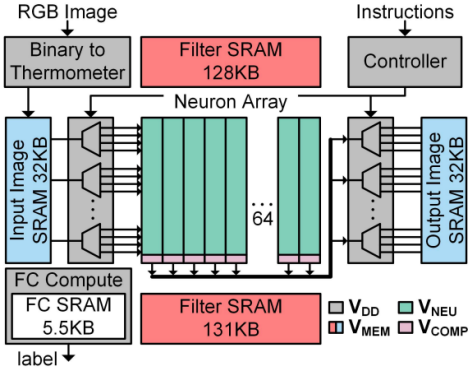
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems
- a small input image (32x32)
- it has a data path to execute a binary convolutional neural network
- weight in SRAM blocks (top/bottom), + inputs and output SRAM
- the controller can only do 3 instructions
  1. take input

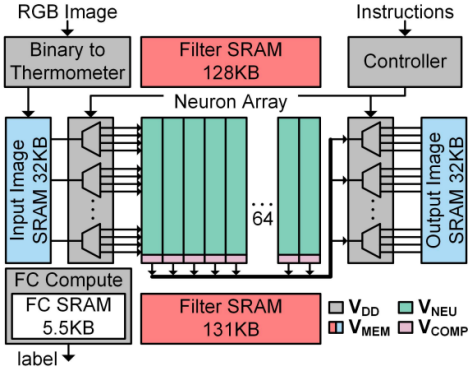
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems
- a small input image (32x32)
- it has a data path to execute a binary convolutional neural network
- weight in SRAM blocks (top/bottom), + inputs and output SRAM
- the controller can only do 3 instructions
  1. take input
  2. CONV

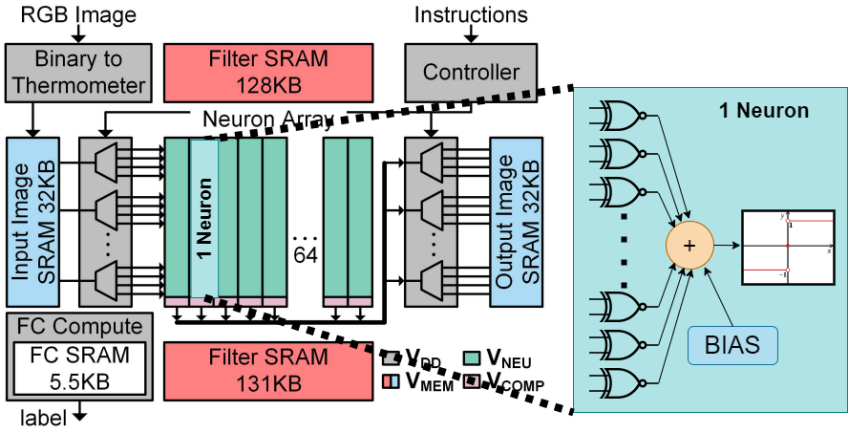
Binareye: architecture



[Moons et al, 2018]

- always-on network for wake-up systems
- a small input image (32x32)
- it has a data path to execute a binary convolutional neural network
- weight in SRAM blocks (top/bottom), + inputs and output SRAM
- the controller can only do 3 instructions
  1. take input
  2. CONV
  3. FC (in a separate data path)

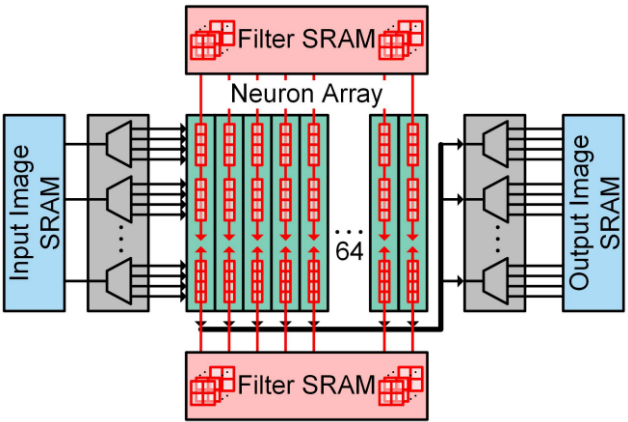
Binareye: architecture



[Moons et al, 2018]

- Weights stationary (in RF)
- Inputs and output parallel
- 1024 inputs each clock cycles

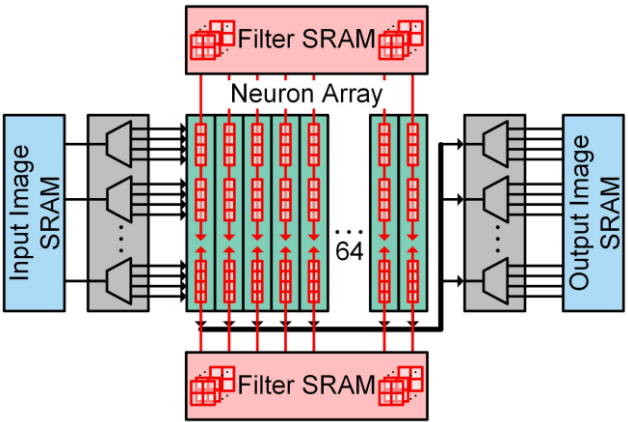
Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Load each column C*FX*FY in local register, i.e., weight stationary

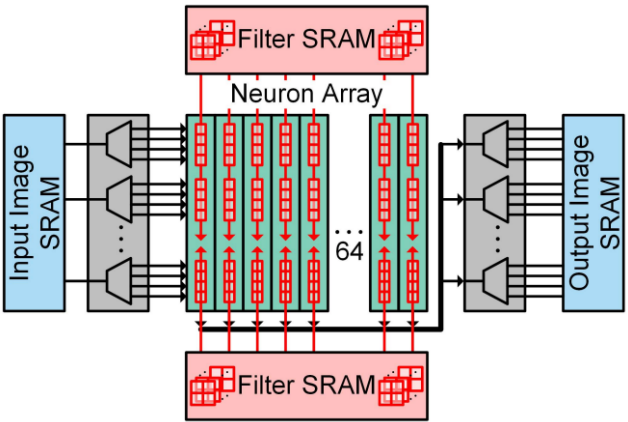
Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Load each column C*FX*FY in local register, i.e., weight stationary
- Load inputs, and inputs are shared across 64 columns (every neuron reuse the same input), i.e., input reuse

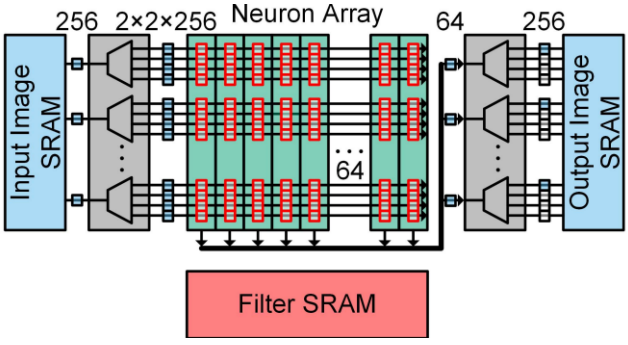
Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Load each column C*FX*FY in local register, i.e., weight stationary
- Load inputs, and inputs are shared across 64 columns (every neuron reuse the same input), i.e., input reuse
- Across one column, results are accumulated (they are computing products for the same output), i.e., output reuse

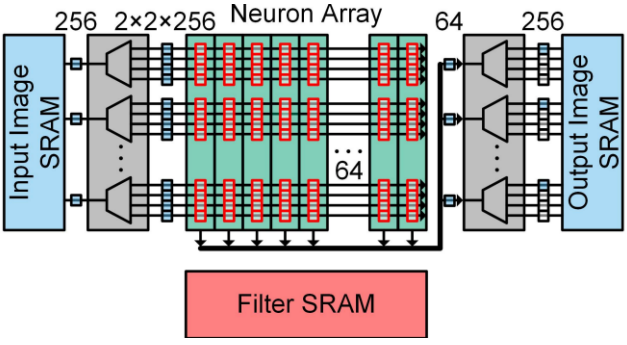
Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Very few memory fetches! (weight stationary, input reuse, output reuse)

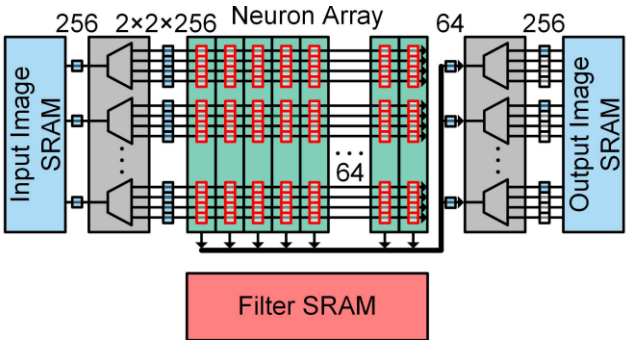
Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Very few memory fetches! (weight stationary, input reuse, output reuse)
- Every cc:

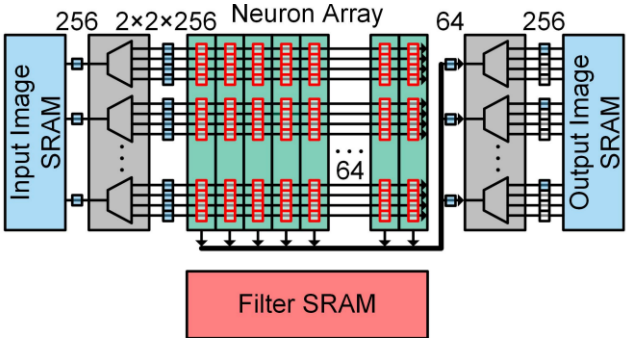
Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Very few memory fetches! (weight stationary, input reuse, output reuse)
- Every cc:
  - fetch 2x2x256 (1024) bits input

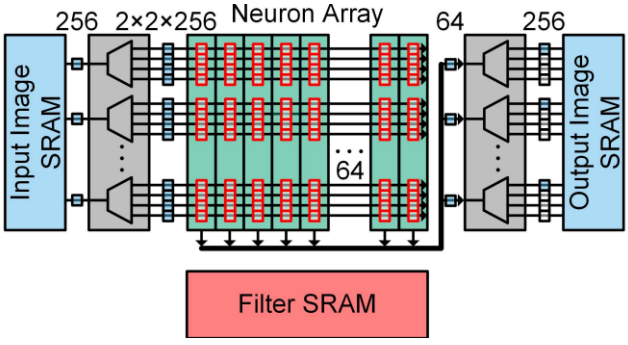
Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Very few memory fetches! (weight stationary, input reuse, output reuse)
- Every cc:
  - fetch 2x2x256 (1024) bits input
  - store 64 bits output

Binareye: what kind of stationarity/parallelism?



[Moons et al, 2018]

- Very few memory fetches! (weight stationary, input reuse, output reuse)
- Every cc:
  - fetch 2x2x256 (1024) bits input
  - store 64 bits output
  - 65536 MUL (1-bit)

Binareye: what kind of stationarity/parallelism?

```
for(k2=0 to K/64-1); //for each output channel
 for(x=0 to X-1); //for each in/output column
  for(y=0 to Y-1); //for each in/output row
   parfor(c=0 to 255); //for each input channel
    parfor(fx=0 to 2); //for each kernel row
     parfor(fy=0 to 2); //for each kernel column
      parfor(k1=0 to 63); //for each output channel
       o[b][k1+64.k2][x][y] += i[b][c][x+fx][y+fy] *
           w[k1+64.k2][c][fx][fy]
```

why weight stationary?

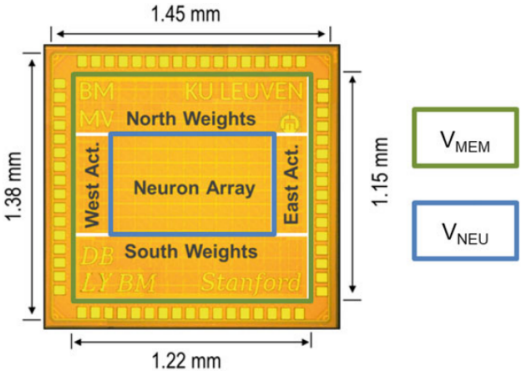
Binareye: what kind of stationarity/parallelism?

```
for(k2=0 to K/64-1); //for each output channel
 for(x=0 to X-1); //for each in/output column
  for(y=0 to Y-1); //for each in/output row
   parfor(c=0 to 255); //for each input channel
    parfor(fx=0 to 2); //for each kernel row
     parfor(fy=0 to 2); //for each kernel column
      parfor(k1=0 to 63); //for each output channel
       o[b][k1+64.k2][x][y] += i[b][c][x+fx][y+fy] *
           w[k1+64.k2][c][fx][fy]
```

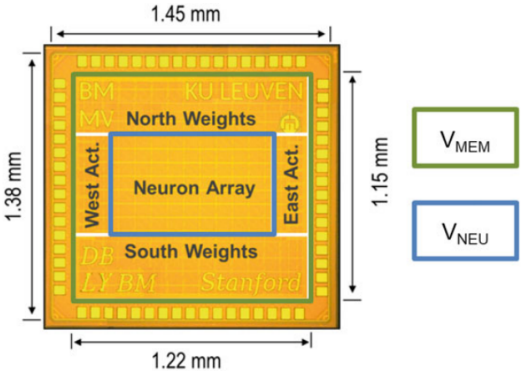
why weight stationary?

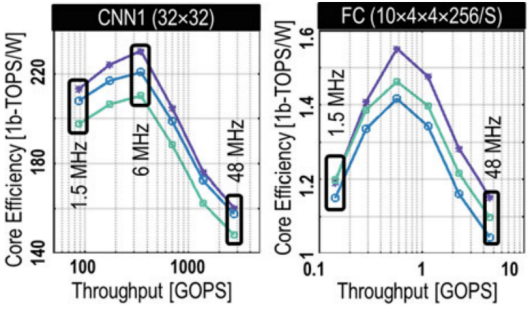
- because my x,y loop is just before the parfor (x,y reuse the same weight)

Binareye: KU Leuven



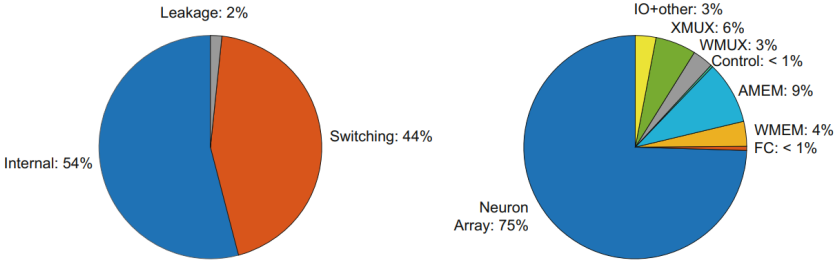
Binareye: KU Leuven





- 1 op = BINARY operation!

Binareye: KU Leuven

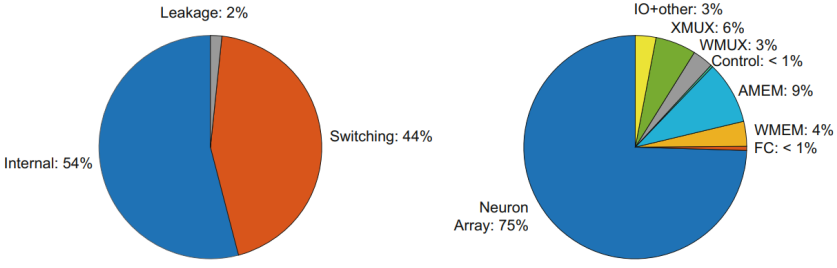


(a) Breakdown by type

(b) Functional breakdown

- The benefits of this is massive parallelization $\rightarrow$ massive data reuse

Binareye: KU Leuven

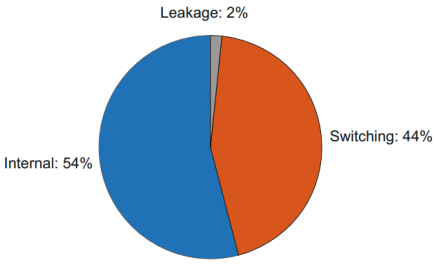


(a) Breakdown by type

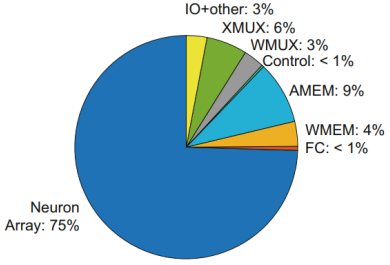
(b) Functional breakdown

- The benefits of this is massive parallelization $\rightarrow$ massive data reuse
- Thus, energy goes into effective compute and less into memory fetches! (75%!)

Binareye: KU Leuven



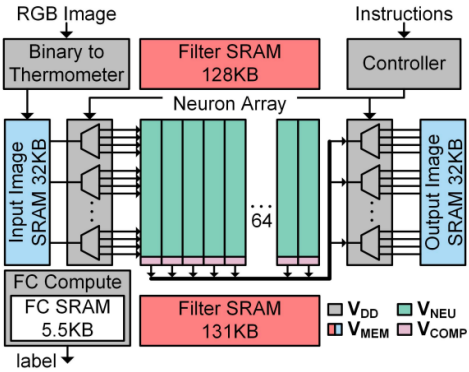
(a) Breakdown by type

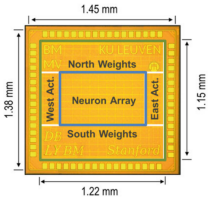


(b) Functional breakdown

- The benefits of this is massive parallelization $\rightarrow$ massive data reuse
- Thus, energy goes into effective compute and less into memory fetches! (75%!)
- It is possible to do better?

Binareye: KU Leuven & Stanford

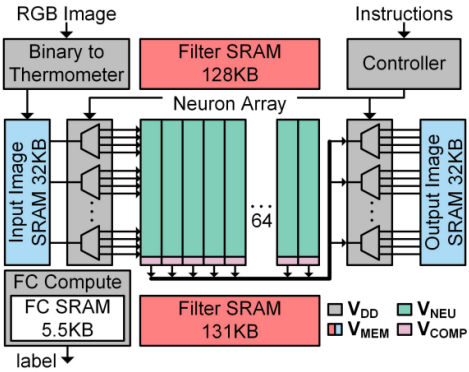


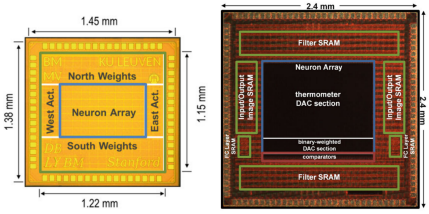


Stanford Mixed-Signal Design

Design the same functionality but using analog and mixed-signal design principles [Bankman et al, 2019]

Binareye: KU Leuven & Stanford

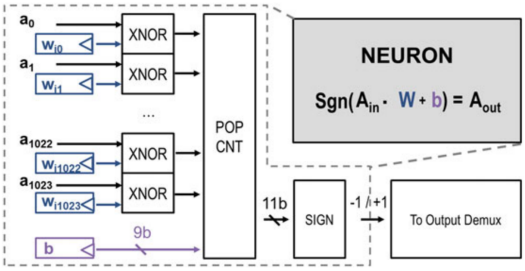




Stanford Mixed-Signal Design

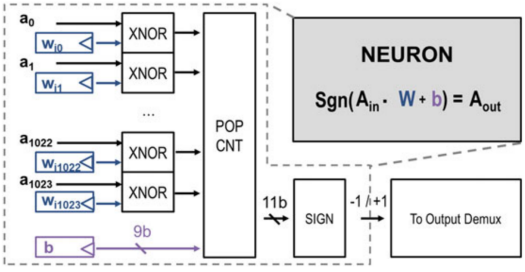
Design the same functionality but using analog and mixed-signal design principles [Bankman et al, 2019]

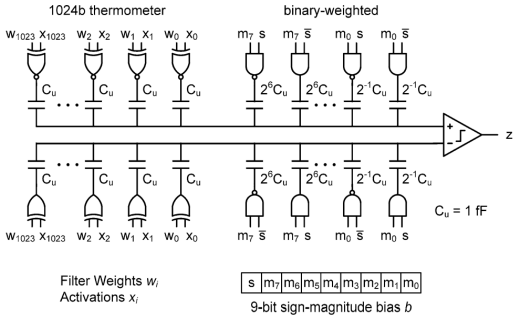
Binareye: KU Leuven & Stanford



- Wide vector summation in the neuron array is the main energy bottleneck in the digital implementation

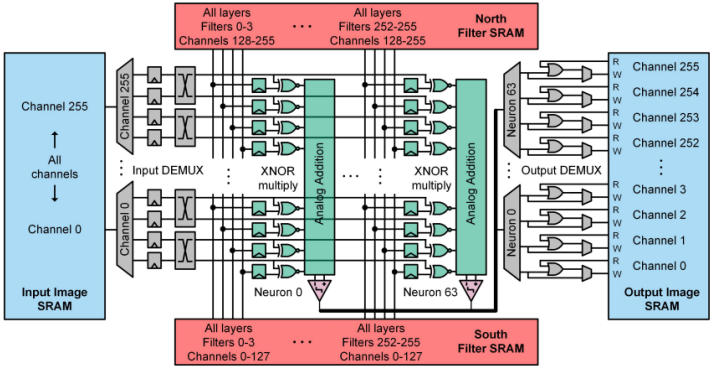
Binareye: KU Leuven & Stanford





- Wide vector summation in the neuron array is the main energy bottleneck in the digital implementation
- Addition in the analog domain with a switched-capacitor (SC) neuron

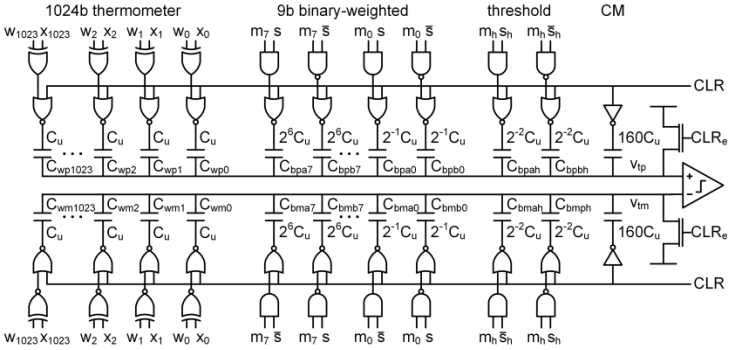
Binareye: KU Leuven & Stanford



Mixed-signal design

Keep XOR digital, and add up in the analog domain $\rightarrow$ charge distribution, over smaller cap, compared to digital.

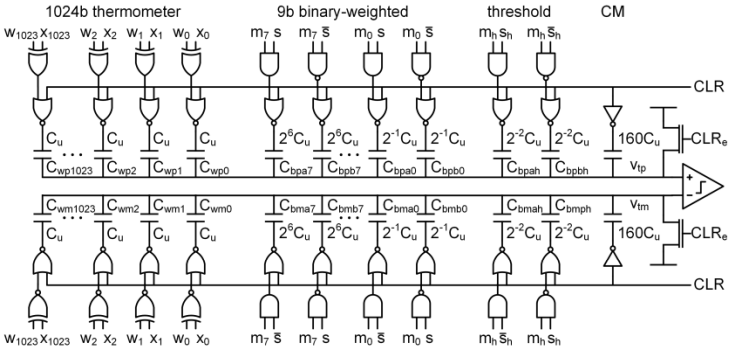
Binareye: KU Leuven & Stanford



Mixed-signal design

Keep XOR digital, and add up in the analog domain $\rightarrow$ charge distribution, over smaller cap, compared to digital.

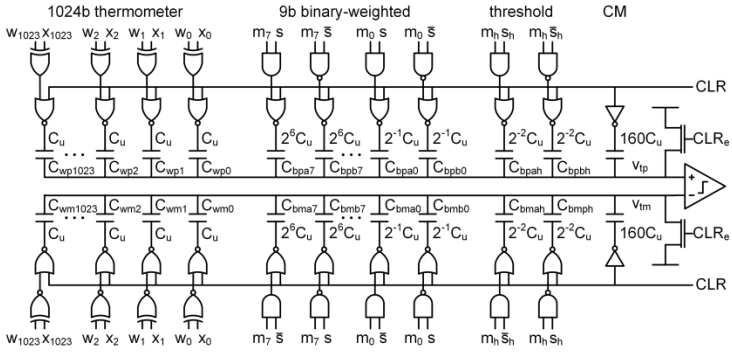
Binareye: KU Leuven & Stanford



Downsides?

- Area (from $1.3mm^2$ to $2.4mm^2$)

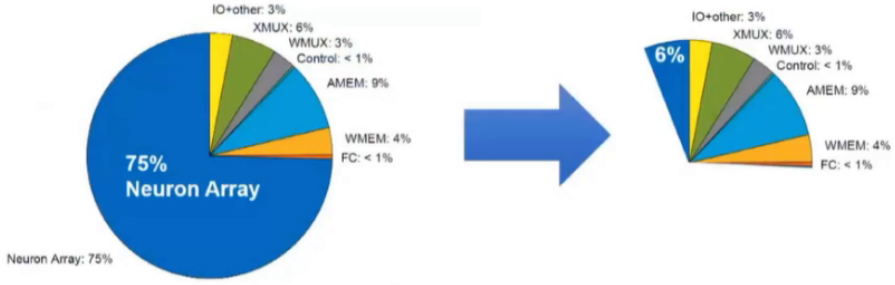
Binareye: KU Leuven & Stanford



Downsides?

- Area (from $1.3mm^2$ to $2.4mm^2$)
- Calibration (bias-voltage, PVT variations)

Binareye: KU Leuven & Stanford

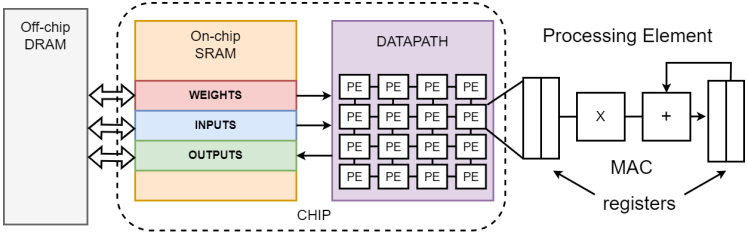


Results

- **Left:** all digital binareye, **Right:** mixed-signal binareye.
- Neuron array bottleneck reduced by 13x, system level by 4x!
- CNN, Cifar-10, accuracy=86.05%, $E_{frame} = 3.79\mu J$, power=0.899mW, fps=237, $532 TOPS/W (1 bit)$

Outline

Architectural choices



- Number of memory levels
- Size of memories
- BW of memories
- Number of PEs
- Spatial unrolling
- Precision

Many, many, many degrees of freedom

So many approaches and degrees of freedom $\rightarrow$ we need analytical costs models.

Optimizing DNN embedded processor design

Analytical costs models

- Energy, latency, area
- Technology characteristics
- Hardware architecture & constraints
- Neural network workload
- Software compilers/mapping tools

Optimizing DNN embedded processor design

Analytical costs models

- Energy, latency, area
- Technology characteristics
- Hardware architecture & constraints
- Neural network workload
- Software compilers/mapping tools

Models/Tools

- Timeloop (MIT) paper, code.
- Interstellar (Stanford) paper, code.
- ZigZag (KU Leuven) paper,code.
- MAESTRO (Nvidia) paper

Summary

Extreme of quantization

- Binary neural networks
- Binareye:
  1. digital KU Leuven
  2. mixed-signal KU Leuven & Stanford

Extreme edge-AI processors

- extreme quantization can be useful for always-on systems

Summary

Extreme of quantization

- Binary neural networks
- Binareye:
  1. digital KU Leuven
  2. mixed-signal KU Leuven & Stanford

Extreme edge-AI processors

- extreme quantization can be useful for always-on systems
- it is possible to reach very high A_I at the cost of accuracy

Summary

Extreme of quantization

- Binary neural networks
- Binareye:
  1. digital KU Leuven
  2. mixed-signal KU Leuven & Stanford

Extreme edge-AI processors

- extreme quantization can be useful for always-on systems
- it is possible to reach very high A_I at the cost of accuracy
- might be acceptable for some applications, but not for all

Summary

Extreme of quantization

- Binary neural networks
- Binareye:
  1. digital KU Leuven
  2. mixed-signal KU Leuven & Stanford

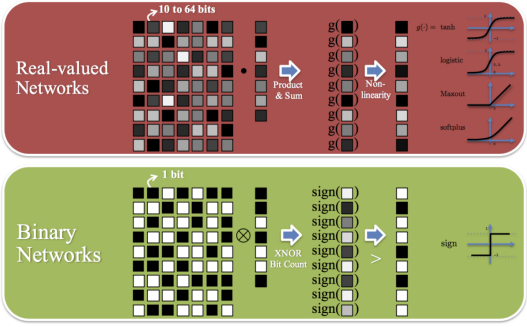
Extreme edge-AI processors

- extreme quantization can be useful for always-on systems
- it is possible to reach very high A_I at the cost of accuracy
- might be acceptable for some applications, but not for all
- mixed signal approach enable to save further energy in compute at the cost of area and tedious calibration

Reduced Precision in Research

Much more than just binary...

- Reduce number of bits
  - Binary Nets [Courbariaux, 2015]
- Reduce number of unique weights and/or activations
  - Ternary Weight Nets [Li, 2022]
  - XNOR-Net [Rastegari, 2016]
- Non-Linear Quantization
  - LogNet [Lee, 2017]



[Courbariaux, 2016]

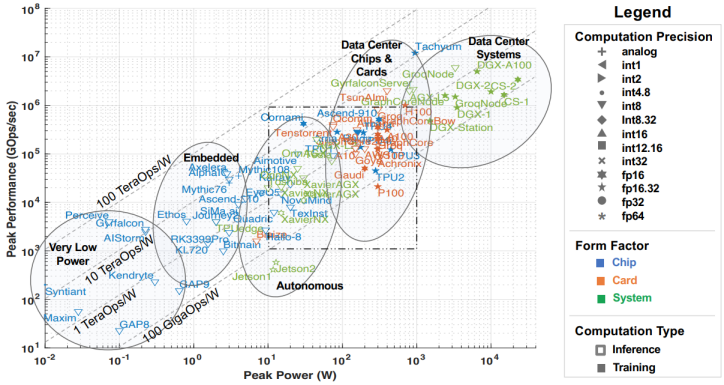
Outline

TinyML

State-of-the-art tinyML

- motivations for ML on microcontroller (MCU)
- tinyML platforms and applications
- anatomy of a keyword spotting application
- energy/cost analysis

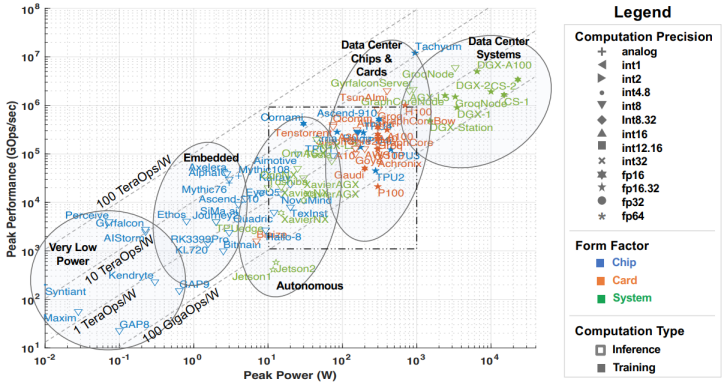
DNN processors overview



- there is a big range in power and efficiency!

[Reuther et al, 2022]

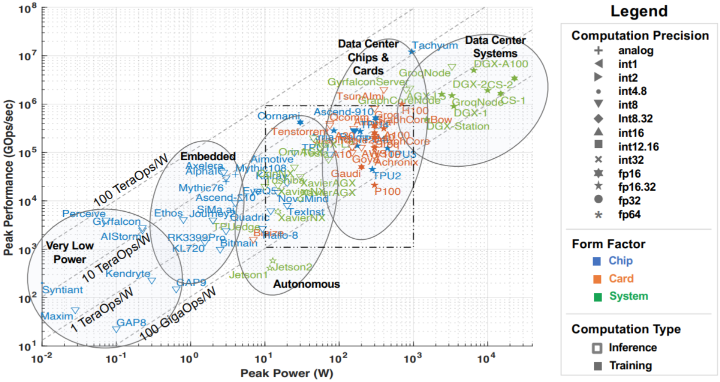
DNN processors overview



- there is a big range in power and efficiency!
- high peak performance → very specialized solutions

[Reuther et al, 2022]

DNN processors overview

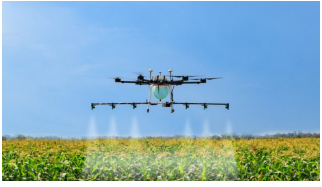


- there is a big range in power and efficiency!
- high peak performance → very specialized solutions
- how about programmability & very-low-power?

[Reuther et al, 2022]

TU/e

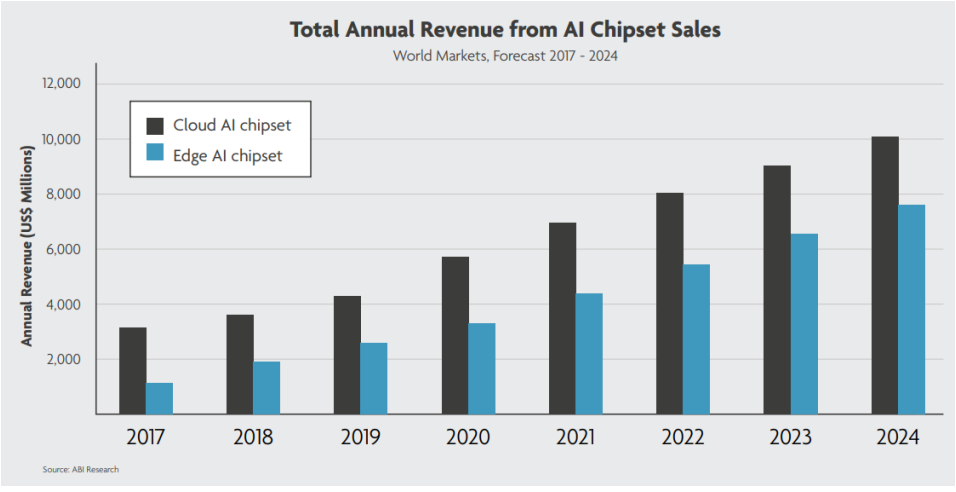
TinyML: why?



Some promising social applications of TinyML

- wildlife conservation
- agriculture
- early detection of respiratory diseases

TinyML: why?



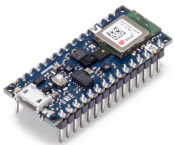
[Rabaey et al, 2019]

TinyML: challenges

TinyML

- provide machine learning capabilities in a microcontroller-like HW (i.e., few mW in average - operational vs stand-by)
- battery lifetime
- cheap (small area, simple system)
- versatile (audio, vision, radar, vibration, ..)

HiMax WE-I

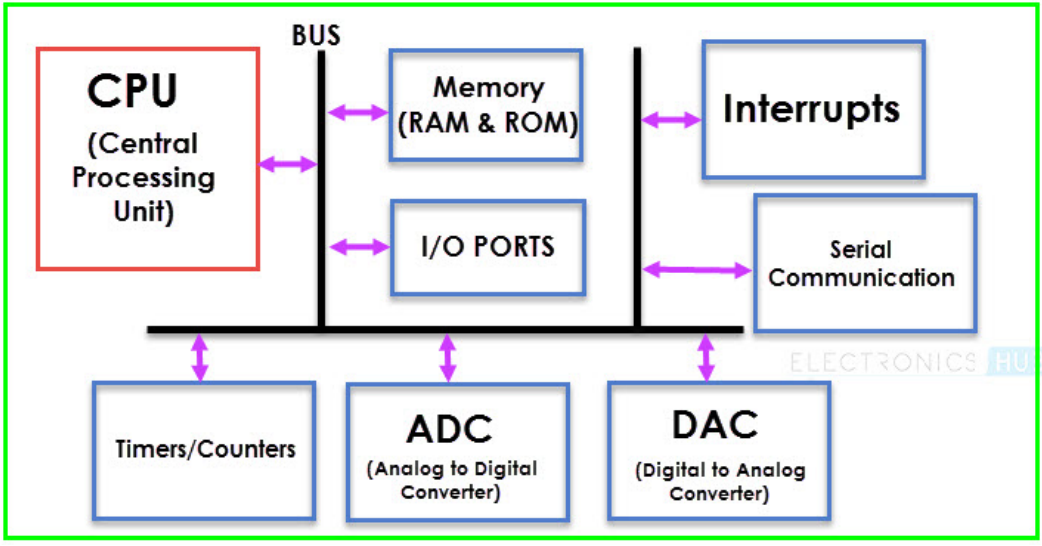


Arduino Nano BLE Sense [M. Banzi, 2021]

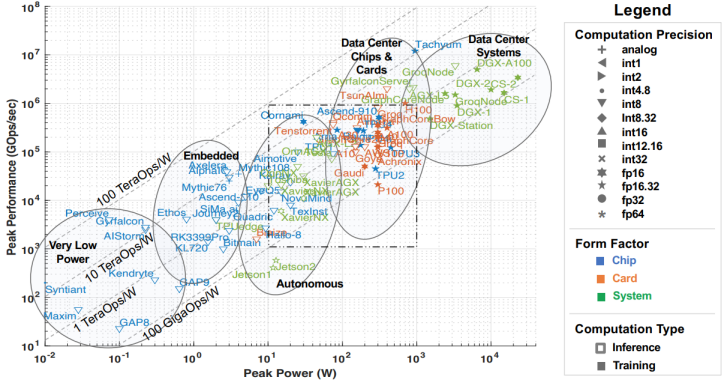
SparkFun Edge 2

ESP-EYE

Example of Micro Controller Architecture



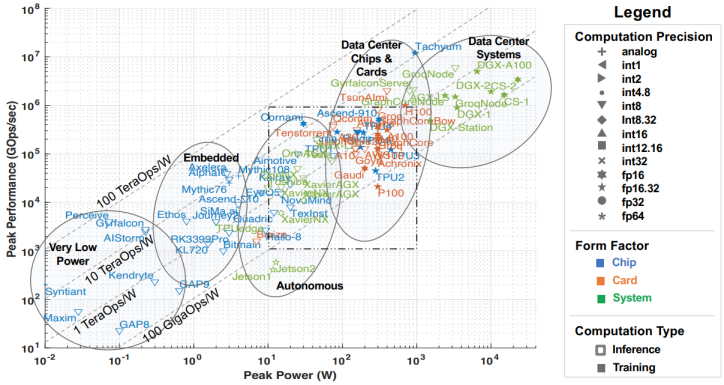
DNN processors overview



- there is a big range in power and efficiency!

[Reuther et al, 2022]

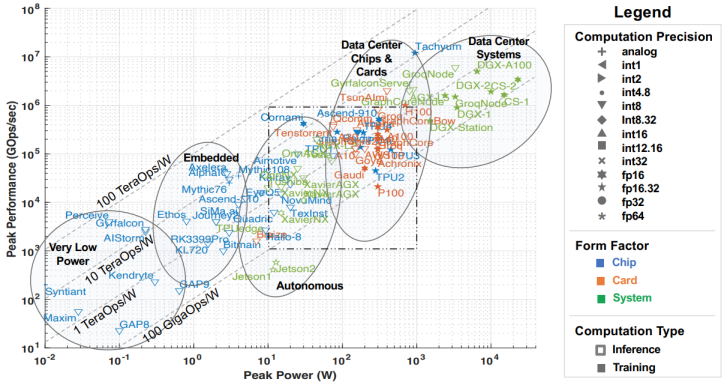
DNN processors overview



- there is a big range in power and efficiency!
- high peak performance → very specialized solutions

[Reuther et al, 2022]

DNN processors overview



- there is a big range in power and efficiency!
- high peak performance → very specialized solutions
- MCUs are in the very-low-power region

[Reuther et al, 2022]

Can we use MCUs for ML tasks?

Example: keyword spotting on ARM MCU

- Always-on systems (e.g., "Hello Google", "Alexa")

Can we use MCUs for ML tasks?

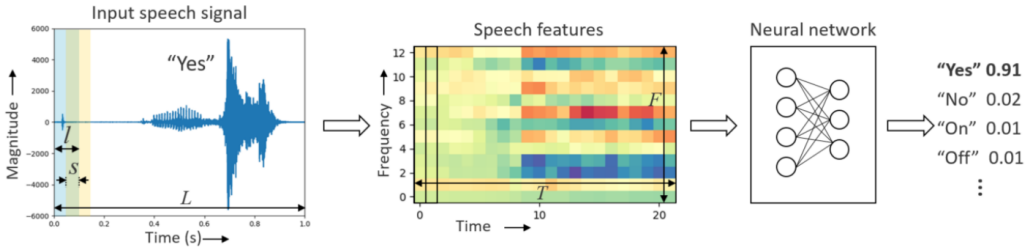
Example: keyword spotting on ARM MCU

- Always-on systems (e.g., "Hello Google", "Alexa")
- How do they work?

Can we use MCUs for ML tasks?

Example: keyword spotting on ARM MCU

- Always-on systems (e.g., "Hello Google", "Alexa")
- How do they work?



Can we use MCUs for ML tasks?

Microcontroller systems

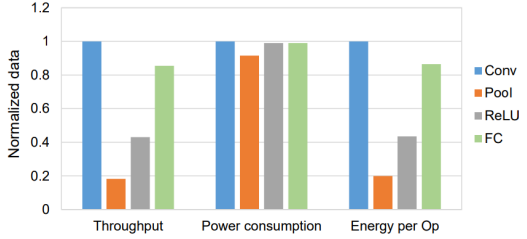
- Memory is limited
- Operation per second is limited
- Low-throughput for compute-intensive workloads
- Some MCUs (e.g., Cortex-M4/M7) have DPS instructions (SIMD, MAC)
- Require low-precision DNNs

Arm Mbed™ platform	Processor	Frequency	SRAM	Flash
Mbed LPC11U24	Cortex-M0	48 MHz	8 KB	32 KB
Nordic nRF51-DK	Cortex-M0	16 MHz	32 KB	256 KB
Mbed LPC1768	Cortex-M3	96 MHz	32 KB	512 KB
Nucleo F103RB	Cortex-M3	72 MHz	20 KB	128 KB
Nucleo L476RG	Cortex-M4	80 MHz	128 KB	1 MB
Nucleo F411RE	Cortex-M4	100 MHz	128 KB	512 KB
FRDM-K64F	Cortex-M4	120 MHz	256 KB	1 MB
Nucleo F746ZG	Cortex M7	216 MHz	320 KB	1 MB

Neural Networks Architectures for Keyword Spotting (KWS)

For example, in an ARM Cortex M7:

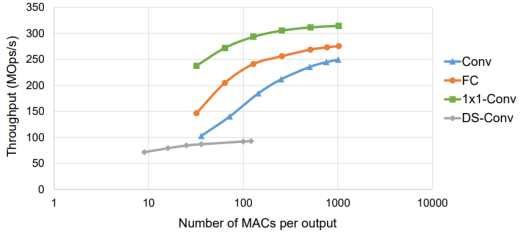
- What layers requires more throughput/power/energy per op?



Neural Networks Architectures for Keyword Spotting (KWS)

For example, in an ARM Cortex M7:

- What layers requires more throughput/power/energy per op?
- What types of layers have the highest throughput and are less MAC intensive?

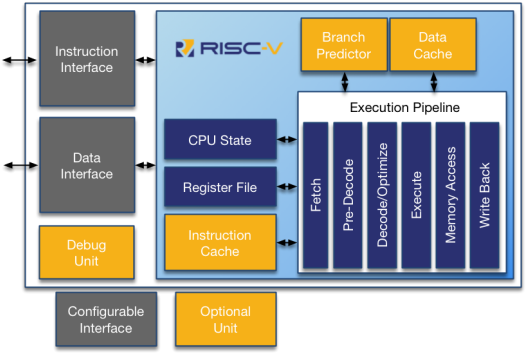


TinyML: microcontrollers start to support ML workloads

How to achieve tinyML in MCU?

- add specialized instruction (e.g., MAC)
- add efficient data motion instruction (e.g., auto-increment, vectorized data fetch)
- HW support for multiple low-precision instructions in a single clock cycle (single instruction multiple data SIMD)
- this will inevitably increase HW costs $\rightarrow$ we need a large gain in energy efficiency

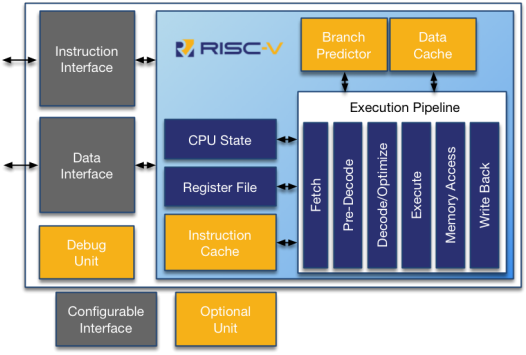
RISC-V architecture and instruction set



What is an instruction set?

Is a group of commands for a central processing unit (CPU) in machine language. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations.

RISC-V architecture and instruction set



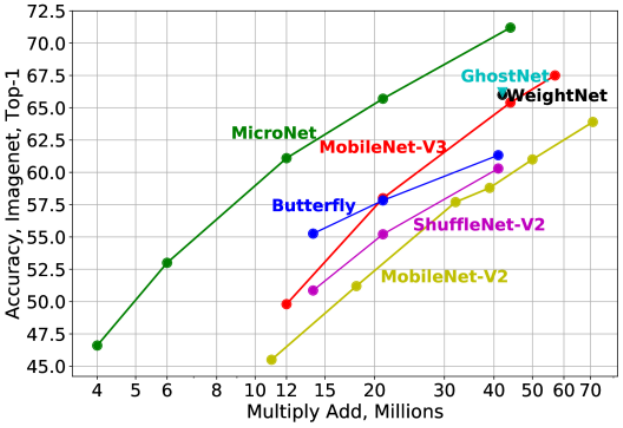
What is an instruction set?

Is a group of commands for a central processing unit (CPU) in machine language. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations.

RISC-V

- RISC-V is an open instruction set (free and open)
- started in UC Berkeley in 2010
- instruction set can be extended by anybody!

TinyML: workloads

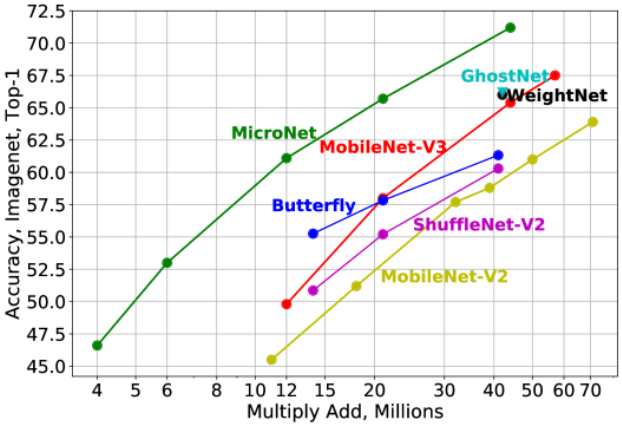


[Li et al, 2021]

Model zoo → programmable hardware!

- tiny neural networks (many network topologies)
- still useful! (not at the level of larger models)
- costs (memory & complexity)
- computational capabilities (elementary operation needed to run one inference)

TinyML: workloads

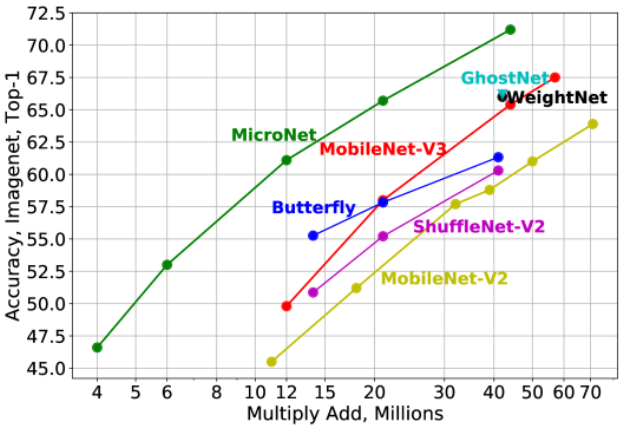


[Li et al, 2021]

Model zoo → some good news!

- lots of operations per data fetch

TinyML: workloads

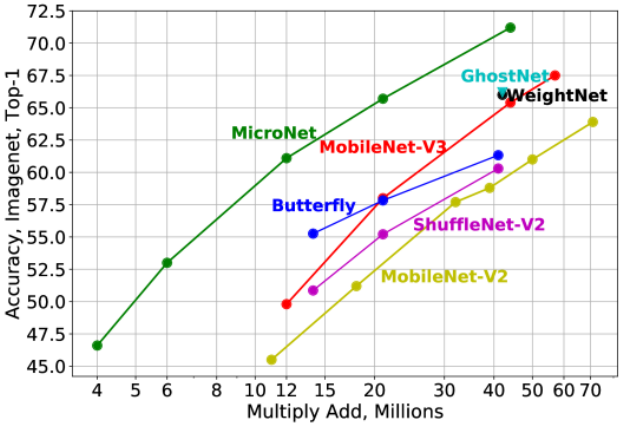


[Li et al, 2021]

Model zoo → some good news!

- lots of operations per data fetch
- massive data parallelism

TinyML: workloads

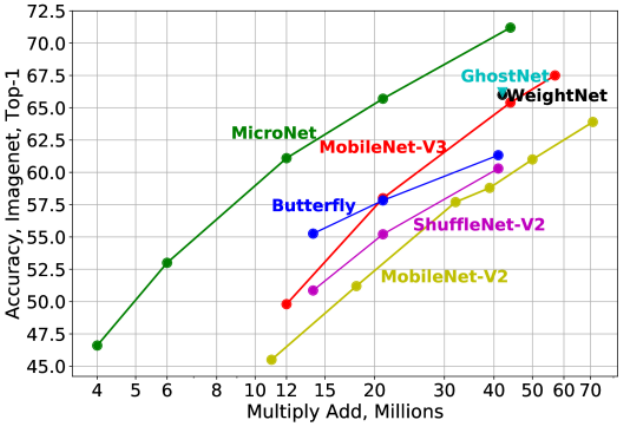


[Li et al, 2021]

Model zoo → some good news!

- lots of operations per data fetch
- massive data parallelism
- MAC-dominated

TinyML: workloads



[Li et al, 2021]

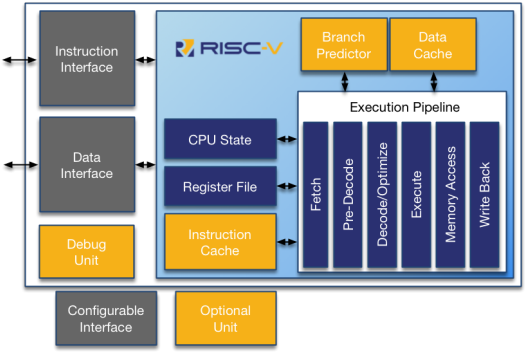
Model zoo → some good news!

- lots of operations per data fetch
- massive data parallelism
- MAC-dominated
- they support low-precision (8b,4b,2b)

TinyML: RISC-V architecture and instruction set

RISC-V

- there is a small core of instruction set (common in all RISC-V family processors)

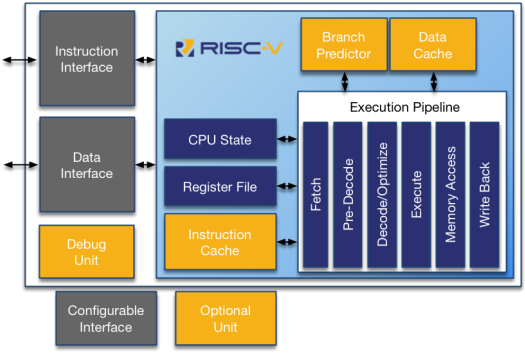


[RISC-V Reference Card]

TinyML: RISC-V architecture and instruction set

RISC-V

- there is a small core of instruction set (common in all RISC-V family processors)
- there are a set of extra instruction set that can be added (depending on the functionality)



[RISC-V Reference Card]

RISC-V architecture and instruction set

Free & Open RISC-V Reference Card ①

Base Integer Instructions: RV32I, RV64I, and RV128I

Category	Name	Fmt	RV32I Base	+RV{64,128}
Loads	Load Byte	I	LB rd,rs1,imm	
	Load Halfword	I	LH rd,rs1,imm	
	Load Word	I	LW rd,rs1,imm	L{D|Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm	
	Load Half Unsigned	I	LHU rd,rs1,imm	L{W|D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm	
	Store Halfword	S	SH rs1,rs2,imm	
	Store Word	S	SW rs1,rs2,imm	S{D|Q} rs1,rs2,imm
Shifts	Shift Left	R	SLL rd,rs1,rs2	SLL{W|D} rd,rs1,rs2
	Shift Left Immediate	I	SLLI rd,rs1,shamt	SLLI{W|D} rd,rs1,shamt
	Shift Right	R	SRL rd,rs1,rs2	SRL{W|D} rd,rs1,rs2
	Shift Right Immediate	I	SRLI rd,rs1,shamt	SRLI{W|D} rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRA{W|D} rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI rd,rs1,shamt	SRAI{W|D} rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADD{W|D} rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDI{W|D} rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUB{W|D} rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm	
	Add Upper Imm to PC	U	AUIPC rd,imm	
Logical	XOR	R	XOR rd,rs1,rs2	
	XOR Immediate	I	XORI rd,rs1,imm	
	OR	R	OR rd,rs1,rs2	
	OR Immediate	I	ORI rd,rs1,imm	
	AND	R	AND rd,rs1,rs2	
	AND Immediate	I	ANDI rd,rs1,imm	
Compare	Set <	R	SLT rd,rs1,rs2	
	Set < Immediate	I	SLTI rd,rs1,imm	
	Set < Unsigned	R	SLTU rd,rs1,rs2	
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	
Branches	Branch =	SB	BEQ rs1,rs2,imm	
	Branch ≠	SB	BNE rs1,rs2,imm	
	Branch <	SB	BLT rs1,rs2,imm	
	Branch ≥	SB	BGE rs1,rs2,imm	
	Branch < Unsigned	SB	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm	
Jump & Link	J&L	UJ	JAL rd,imm	
	Jump & Link Register	UJ	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
System	System CALL	I	SCALL	
	System BREAK	I	SBREAK	
Counters	ReaD CYCLE	I	RDCYCLE rd	
	ReaD CYCLE upper Half	I	RDCYCLEH rd	
	ReaD TIME	I	RDTIME rd	
	ReaD TIME upper Half	I	RDTIMEH rd	
	ReaD INSTR RETired	I	RDINSTRET rd	
	ReaD INSTR upper Half	I	RDINSTRETH rd	

RV Privileged Instructions

Category	Name	RV mnemonic
CSR Access	Atomic R/W	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	CSRRC rd,csr,rs1
	Atomic R/W Imm	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	CSRRCI rd,csr,imm
Change Level	Env. Call	ECALL
	Environment Breakpoint	EBREAK
	Environment Return	ERET
Trap Redirect	to Supervisor	MRTS
	Redirect Trap to Hypervisor	MRTH
	Hypervisor Trap to Supervisor	HRTS
Interrupt	Wait for Interrupt	WFI
MMU	Supervisor FENCE	SFENCE.VM rs1

Optional Compressed (16-bit) Instruction Extension: RVC

Category	Name	Fmt	RVC	RVI equivalent
Loads	Load Word	CL	C.LW rd′,rs1′,imm	LW rd′,rs1′,imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Load Double	CL	C.LD rd′,rs1′,imm	LD rd′,rs1′,imm*8
	Load Double SP	CI	C.LDSP rd,imm	LD rd,sp,imm*8
	Load Quad	CL	C.LQ rd′,rs1′,imm	LQ rd′,rs1′,imm*16
	Load Quad SP	CI	C.LQSP rd,imm	LQ rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1′,rs2′,imm	SW rs1′,rs2′,imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Store Double	CS	C.SD rs1′,rs2′,imm	SD rs1′,rs2′,imm*8
	Store Double SP	CSS	C.SDSP rs2,imm	SD rs2,sp,imm*8
	Store Quad	CS	C.SQ rs1′,rs2′,imm	SQ rs1′,rs2′,imm*16
	Store Quad SP	CSS	C.SQSP rs2,imm	SQ rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,rs1
	ADD Word	CR	C.ADDW rd,rs1	ADDW rd,rd,imm
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm	ADDIW rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd',imm	ADDI rd',sp,imm*4
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm
	Load Upper Imm	CI	C.LUI rd,imm	LUI rd,imm
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
	SUB	CR	C.SUB rd,rs1	SUB rd,rd,rs1
Shifts	Shift Left Imm	CI	C.SLLI rd,imm	SLLI rd,rd,imm
Branches	Branch=0	CB	C.BEQZ rs1′,imm	BEQ rs1',x0,imm
	Branch≠0	CB	C.BNEZ rs1′,imm	BNE rs1',x0,imm
Jump	Jump	CJ	C.J imm	JAL x0,imm
	Jump Register	CR	C.JR rd,rs1	JALR x0,rs1,0
Jump & Link	J&L	CJ	C.JAL imm	JAL ra,imm
	Jump & Link Register	CR	C.JALR rs1	JALR ra,rs1,0
System	Env. BREAK	CI	C.EBREAK	EBREAK

32-bit Instruction Formats

Format	31 … 25	24 … 20	19 … 15	14 … 12	11 … 7	6 … 0
R	funct7	rs2	rs1	funct3	rd	opcode
I	imm[11:0]		rs1	funct3	rd	opcode
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1] imm[11]	opcode
U	imm[31:12]				rd	opcode
UJ	imm[20] imm[10:1]	imm[11]	imm[19:12]		rd	opcode

16-bit (RVC) Instruction Formats

Format	15 14 13	12	11 10 9 8 7	6 5 4 3 2	1 0
CR	funct4		rd/rs1	rs2	op
CI	funct3	imm	rd/rs1	imm	op
CSS	funct3	imm		rs2	op
CIW	funct3	imm		rd′	op
CL	funct3	imm	rs1′	imm rd′	op
CS	funct3	imm	rs1′	imm rs2′	op
CB	funct3	offset	rs1′	offset	op
CJ	funct3	jump target			op

Free & Open RISC-V Reference Card (riscv.org) ②

Optional Multiply-Divide Instruction Extension: RVM

Category	Name	Fmt	RV32M (Multiply-Divide)	+RV{64,128}
Multiply	MULtiply	R	MUL rd,rs1,rs2	MUL{W|D} rd,rs1,rs2
	MULtiply upper Half	R	MULH rd,rs1,rs2	
	MULtiply Half Sign/Uns	R	MULHSU rd,rs1,rs2	
	MULtiply upper Half Uns	R	MULHU rd,rs1,rs2	
Divide	DIVide	R	DIV rd,rs1,rs2	DIV{W|D} rd,rs1,rs2
	DIVide Unsigned	R	DIVU rd,rs1,rs2	
Remainder	REMainder	R	REM rd,rs1,rs2	REM{W|D} rd,rs1,rs2
	REMainder Unsigned	R	REMU rd,rs1,rs2	REMU{W|D} rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV32A (Atomic)	+RV{64,128}
Load	Load Reserved	R	LR.W rd,rs1	LR.{D|Q} rd,rs1
Store	Store Conditional	R	SC.W rd,rs1,rs2	SC.{D|Q} rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W rd,rs1,rs2	AMOSWAP.{D|Q} rd,rs1,rs2
Add	ADD	R	AMOADD.W rd,rs1,rs2	AMOADD.{D|Q} rd,rs1,rs2
Logical	XOR	R	AMOXOR.W rd,rs1,rs2	AMOXOR.{D|Q} rd,rs1,rs2
	AND	R	AMOAND.W rd,rs1,rs2	AMOAND.{D|Q} rd,rs1,rs2
	OR	R	AMOOR.W rd,rs1,rs2	AMOOR.{D|Q} rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.W rd,rs1,rs2	AMOMIN.{D|Q} rd,rs1,rs2
	MAXimum	R	AMOMAX.W rd,rs1,rs2	AMOMAX.{D|Q} rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W rd,rs1,rs2	AMOMINU.{D|Q} rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W rd,rs1,rs2	AMOMAXU.{D|Q} rd,rs1,rs2

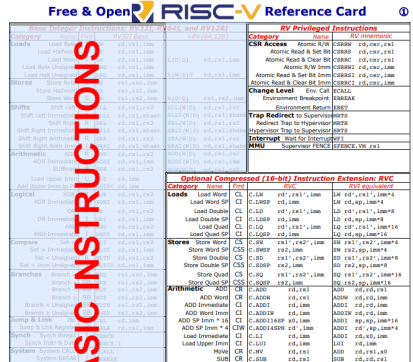
Three Optional Floating-Point Instruction Extensions: RVF, RVD, & RVQ

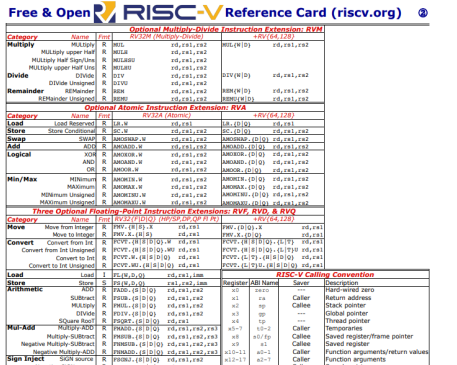
Category	Name	Fmt	RV32{F|D|Q} (HP/SP,DP,QP Fl Pt)	+RV{64,128}
Move	Move from Integer	R	FMV.{H|S}.X rd,rs1	FMV.{D|Q}.X rd,rs1
	Move to Integer	R	FMV.X.{H|S} rd,rs1	FMV.X.{D|Q} rd,rs1
Convert	Convert from Int	R	FCVT.{H|S|D|Q}.W rd,rs1	FCVT.{H|S|D|Q}.{L|T} rd,rs1
	Convert from Int Unsigned	R	FCVT.{H|S|D|Q}.WU rd,rs1	FCVT.{H|S|D|Q}.{L|T}U rd,rs1
	Convert to Int	R	FCVT.W.{H|S|D|Q} rd,rs1	FCVT.{L|T}.{H|S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{H|S|D|Q} rd,rs1	FCVT.{L|T}U.{H|S|D|Q} rd,rs1
Load	Load	I	FL{W,D,Q} rd,rs1,imm	
Store	Store	S	FS{W,D,Q} rs1,rs2,imm	
Arithmetic	ADD	R	FADD.{S|D|Q} rd,rs1,rs2	
	SUBtract	R	FSUB.{S|D|Q} rd,rs1,rs2	
	MULtiply	R	FMUL.{S|D|Q} rd,rs1,rs2	
	DIVide	R	FDIV.{S|D|Q} rd,rs1,rs2	
	SQuare RooT	R	FSQRT.{S|D|Q} rd,rs1	
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q} rd,rs1,rs2,rs3	
	Multiply-SUBtract	R	FMSUB.{S|D|Q} rd,rs1,rs2,rs3	
	Negative Multiply-SUBtract	R	FNMSUB.{S|D|Q} rd,rs1,rs2,rs3	
	Negative Multiply-ADD	R	FNMADD.{S|D|Q} rd,rs1,rs2,rs3	
Sign Inject	SiGN source	R	FSGNJ.{S|D|Q} rd,rs1,rs2	
	Negative SiGN source	R	FSGNJN.{S|D|Q} rd,rs1,rs2	
	Xor SiGN source	R	FSGNJX.{S|D|Q} rd,rs1,rs2	
Min/Max	MINimum	R	FMIN.{S|D|Q} rd,rs1,rs2	
	MAXimum	R	FMAX.{S|D|Q} rd,rs1,rs2	
Compare	Compare Float =	R	FEQ.{S|D|Q} rd,rs1,rs2	
	Compare Float <	R	FLT.{S|D|Q} rd,rs1,rs2	
	Compare Float ≤	R	FLE.{S|D|Q} rd,rs1,rs2	
Categorization	Classify Type	R	FCLASS.{S|D|Q} rd,rs1	
Configuration	Read Status	R	FRCSR rd	
	Read Rounding Mode	R	FRRM rd	
	Read Flags	R	FRFLAGS rd	
	Swap Status Reg	R	FSCSR rd,rs1	
	Swap Rounding Mode	R	FSRM rd,rs1	
	Swap Flags	R	FSFLAGS rd,rs1	
	Swap Rounding Mode Imm	I	FSRMI rd,imm	
	Swap Flags Imm	I	FSFLAGSI rd,imm	

RISC-V Calling Convention

Register	ABI Name	Saver	Description
x0	zero	---	Hard-wired zero
x1	ra	Caller	Return address
x2	sp	Callee	Stack pointer
x3	gp	---	Global pointer
x4	tp	---	Thread pointer
x5-7	t0-2	Caller	Temporaries
x8	s0/fp	Callee	Saved register/frame pointer
x9	s1	Callee	Saved register
x10-11	a0-1	Caller	Function arguments/return values
x12-17	a2-7	Caller	Function arguments
x18-27	s2-11	Callee	Saved registers
x28-31	t3-t6	Caller	Temporaries
f0-7	ft0-7	Caller	FP temporaries
f8-9	fs0-1	Callee	FP saved registers
f10-11	fa0-1	Caller	FP arguments/return values
f12-17	fa2-7	Caller	FP arguments
f18-27	fs2-11	Callee	FP saved registers
f28-31	ft8-11	Caller	FP temporaries

RISC-V architecture and instruction set

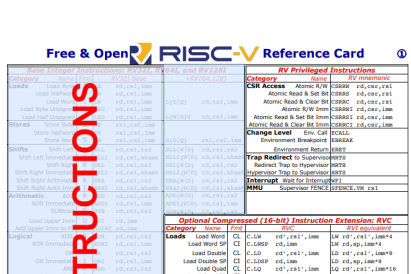


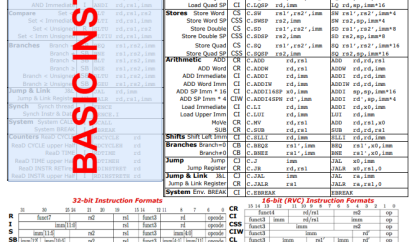


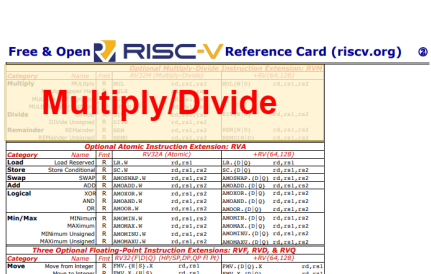
5LIL0 – Personal student use only.

TU/e

RISC-V architecture and instruction set

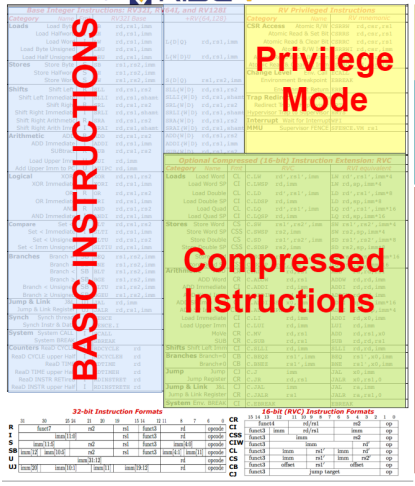


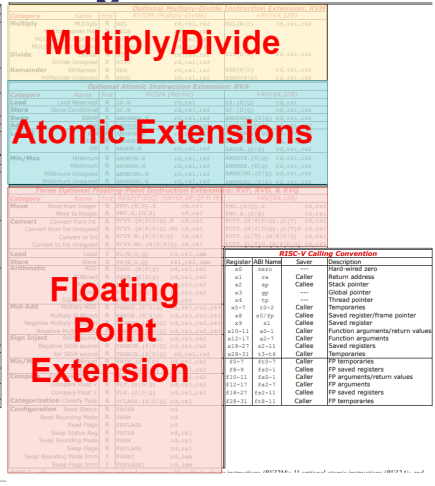




TU/e

RISC-V architecture and instruction set





What are the performance metrics?

Performance metrics

- Area (num. of simple logic gates) or mm^2
- Frequency (depends on num. of gates on longest path)
- Power
  - Strongly depends on area & frequency
  - Leakage: dissipated even when not working (Area)
  - Dynamic Power: dissipated on logic transitions (Frequency & Area)

CPU design

- IPC (instruction per cycle)
  - IPC implicitly measured in commonly used benchmarks (Coremark, Dhrystone, SpecInt)
- Energy Efficiency: OPs/Joule

Hardware designer

- Tries to find a good balance
- Considers application (HPC, tinyML, etc..)
- One size unfortunately does not fit all!

Several RISC-V flavours

Zero-riscy / Ibex

- Low Cost Core
- 32-bit CPU
- fully open source
- GitHub

DSP

- RI5CY
- 32-bit CPU SIMD
- fully open source
- GitHub

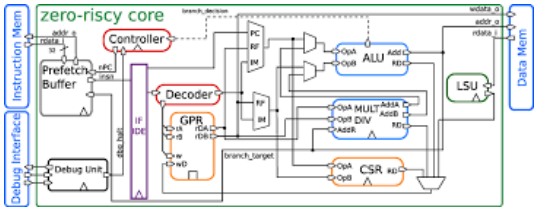
Ariane

- Linux capable core
- 64-bit CPU
- fully open source
- GitHub

Zero-riscy / Ibex, small core for control applications

Ibex

- 2-stage pipeline
- Optimized for area
  - Area:
    - 16kGE (Zero-riscy)
    - 12kGE (Micro-riscy)
  - Critical path:
    - $\sim$ 30 logic levels



Two main configurations:

- Zero-riscy: RV32IMC (2,44 Coremark/Mhz)
  - 32 registers, hardware multiplier
- Micro-riscy: RV32EC (0,91 Coremark/Mhz)
  - 16 registers (E), software emulated multiplier

Zero-riscy / Ibex

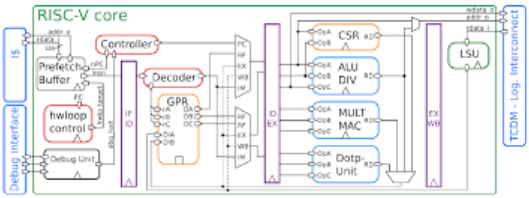
Applications

- Zero-riscy/ibex is suiteable for simple applications (control, book-keeping)
- For number crunching, we need more capable cores
- Tuned for cost and simplicity
- Does not make use of custom extentions

RI5CY 32-bit core

RI5CY

- 4-stage pipeline
  - Area: 41kGE
- Includes extensions
  - SIMD
  - Fixed point
  - Bit manipulations
  - HW loops



Different Options:

- FPU: IEEE 754 single precision
  - including hardware support for FDIV, FSQRT, FMAC, FMUL
- Privilege support
  - Supports privilege mode (M) and (U)

RISC-V has space for custom instruction

Instruction Set X

- There is a reserved decoding space for custom instructions
  - allows everyone to add new instructions to the core
  - the address decoding space is reserved, it will not be used by further extensions
  - implementations supporting custom instructions will be compatible with standard ISA
    - code compiled for standard RISC-V will run without issues
  - The user has to provide support to take advantage of the additional instructions
    - compiler that generates code for the custom instructions

RISC-V ISA extension example

Example of C code

```
for(int i=0; i<128; i++){
        c[i] += a[i]+b[i];
}
```

Baseline 11 cycles/output

```
mv x5,0
mv x4, 100
Lstart:
lb x2,0(x10)
lb x3,0(x12)
addi x10,x10,1
addi x11,x11,1
add x2, x3, x2
sb x2, 0(x12)
addi x4,x4,-1
addi x12,x12,1
bne x4,x5, Lstart
```

Auto-incr, 8 cycles/output

```
mv x5,0
mv x4, 100
Lstart:
lb x2,0(x10!)
lb x3,0(x12!)
addi x4,x4,-1
add x2, x3, x2
sb x2, 0(x12!)
bne x4,x5, Lstart
```

HW loops, 5 cycles/output

```
lp.setupi 100, Lend
lb x2,0(x10!)
lb x3,0(x12!)
add x2, x3, x2
Lend sb x2, 0(x12!)
```

SIMD, 1.25 cycles/output

```
lp.setupi 25, Lend
lw x2,0(x10!)
lw x3,0(x12!)
pv.add.b x2, x3, x2
Lend sw x2, 0(x12!)
```

Basic instruction set is not efficient for DNNs

With the basic instruction set, we require 4 cycles to do a single MAC operation!

```
addi a0,a0,1
addi t1,t1,1
addi t3,t3,1
addi t4,t4,1
lbu a7,-1(a0)
lbu a6,-1(t4)
lbu a5,-1(t3)
lbu t5,-1(1)
mul s1,a7,a6
mul a7,a7,a5
add s0,s0,s1
mul a6,a6,t5
add t0,t0,a7
mul a5,a5,t5
add t2,t2,a6
add t6,t6,a5
bne s5,a0,1c000bc
```

- multiply accumulate of two 8 bits of words (dot product of two vectors)
- with the baseline instruction set it is inefficient
- we need instruction extension to make it more efficient!

RISC-V architecture and instruction set

Many DNN operations are just non-data dependent loops!

- Automatic address update
  1. update base register with computed address after the memory access
  2. save instruction to update address
- post-increment
  1. base address serves as memory address
- offset can be stored in
  - register
  - immediate

```
for(int i=0; i<128; i++){
        c += a[i]*b[i];
}
```

- combine load instruction with increment
- load instruction with post increment

Orginal RISC-V

```
addi x4,x0,64
Lstart:
lb x2,0(x10)
lb x3,0(x12)
addi x10,x10,1
addi x12,x12,1
...
bne x2,x3, Lstart
```

Auto-incr load/store

```
addi x4,x0,64
Lstart:
lb x2,0(x10!)
lb x3,0(x12!)
...
bne x2,x3, Lstart
```

- save 2 instructions

Outline

Conclusions

- TinyML enables practical ML on low-power MCUs
- RISC-V provides a flexible, efficient architecture
- Custom instruction extensions significantly boost efficiency
- Many real-world applications in various domains