Low-Power Neural Network Accelerators: Advancements in Custom Floating-Point Techniques

Yarib Nevarez

Universität Bremen

May 22, 2024

Introduction

- Advancements in AI/ML for IoT and TinyML

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach
- Design challenges

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach
- Design challenges
- Approximate computing and quantization

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach
- Design challenges
- Approximate computing and quantization
- Quality, interoperability, and compatibility

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
    - Optimal custom FP number representation

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
  - Optimal custom FP number representation
  - Energy-efficient design strategies and custom FP arithmetic units

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
    - Optimal custom FP number representation
    - Energy-efficient design strategies and custom FP arithmetic units
    - Precision and quantization impact analysis

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
  - Optimal custom FP number representation
  - Energy-efficient design strategies and custom FP arithmetic units
  - Precision and quantization impact analysis
  - Hardware performance evaluation and bench marking

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
    - Optimal custom FP number representation
    - Energy-efficient design strategies and custom FP arithmetic units
    - Precision and quantization impact analysis
    - Hardware performance evaluation and bench marking
    - Practical application

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
  - Optimal custom FP number representation
  - Energy-efficient design strategies and custom FP arithmetic units
  - Precision and quantization impact analysis
  - Hardware performance evaluation and bench marking
  - Practical application
  - Future research

Outline

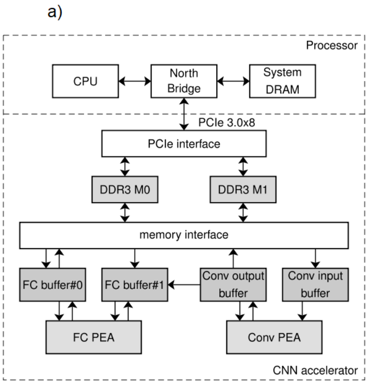
1. State-of-the-Art

2. Hybrid 8-bit Floating-Point and 4-bit Logarithmic Computation

1 State-of-the-Art

2 Hybrid 8-bit Floating-Point and 4-bit Logarithmic Computation

High-Performance FPGA-Based CNN Accelerator With Block-Floating-Point Arithmetic



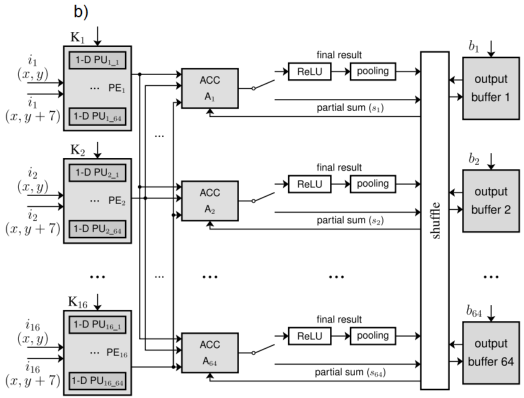


Figure: (a) System architecture. (b) Processing element array.

A 200MHZ 202.4GFLOPS@10.8W VGG16 Accelerator in Xilinx VX690T

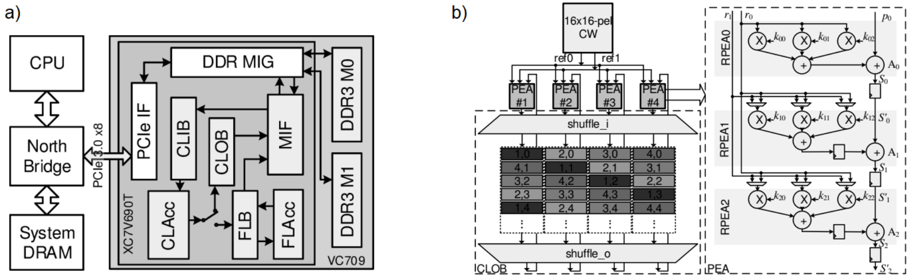
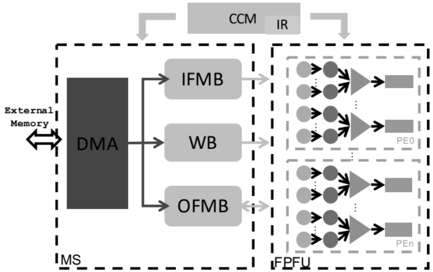


Figure: (a) System architecture. (b) Convolution accelerator.

Low-precision Floating-point Arithmetic for High-performance FPGA-based CNN Acceleration

a)



b)

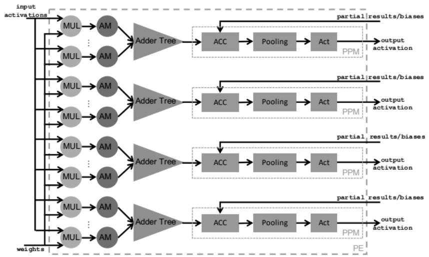
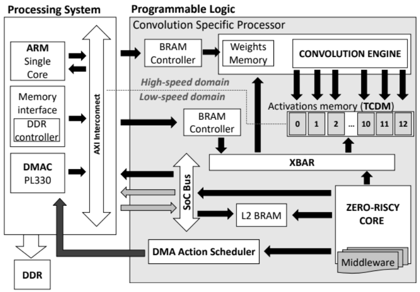


Figure: (a) System architecture. (b) Processing element.

CNN Hardware Acceleration on a Low-Power and Low-Cost APSoC

a)



b)

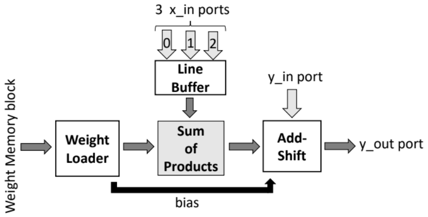


Figure: (a) System architecture. (b) Convolution engine.

Spike-by-Spike Neural Network

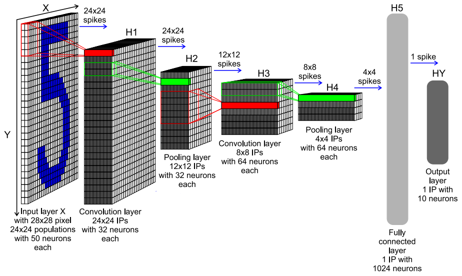

X
Y
24x24 spikes
H1
24x24 spikes
H2
12x12 spikes
H3
8x8 spikes
H4
4x4 spikes
H5
1 spike
HY
Input layer X with 28x28 pixel 24x24 populations with 50 neurons each
Convolution layer 24x24 IPs with 32 neurons each
Pooling layer 12x12 IPs with 32 neurons each
Convolution layer 8x8 IPs with 64 neurons each
Pooling layer 4x4 IPs with 64 neurons each
Fully connected layer 1 IP with 1024 neurons
Output layer 1 IP with 10 neurons

Figure: Spike-by-Spike (SbS) neural network architecture for handwritten digit classification task.

Spike-by-Spike Neural Network

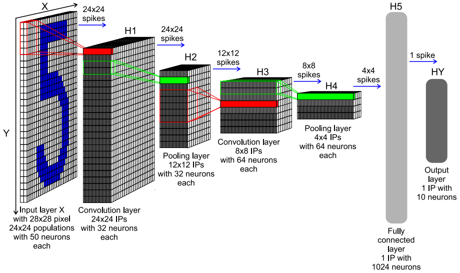

X
Y
24x24 spikes
H1
24x24 spikes
H2
12x12 spikes
H3
8x8 spikes
H4
4x4 spikes
H5
1 spike
HY
Input layer X with 28x28 pixel 24x24 populations with 50 neurons each
Convolution layer 24x24 IPs with 32 neurons each
Pooling layer 12x12 IPs with 32 neurons each
Convolution layer 8x8 IPs with 64 neurons each
Pooling layer 4x4 IPs with 64 neurons each
Fully connected layer 1 IP with 1024 neurons
Output layer 1 IP with 10 neurons

Figure: Spike-by-Spike (SbS) neural network architecture for handwritten digit classification task.

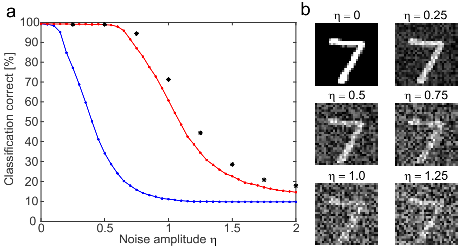


Figure: Performance classification of SbS NN versus equivalent CNN.

Spike-by-Spike Layer Update

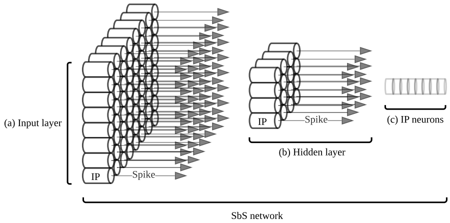


Figure: SbS inference population (IP) as independent computational entities.

$$h_{\mu}^{new}(i) = \frac{1}{1+\epsilon}\left(h_{\mu}(i) + \epsilon \frac{h_{\mu}(i)W(s_t|i)}{\sum_j h_{\mu}(j)W(s_t|j)}\right)$$

HW/SW Co-Design Framework

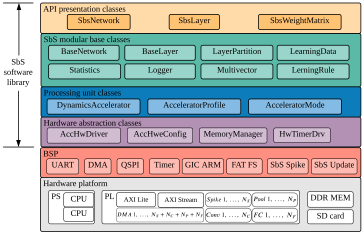


Figure: System-level overview of the embedded software architecture.

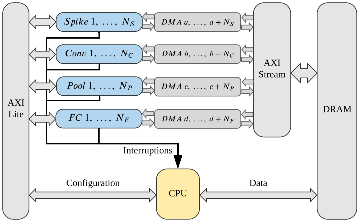


Figure: System-level hardware architecture with scalable number of heterogeneous processing units (PU).

Processing Unit

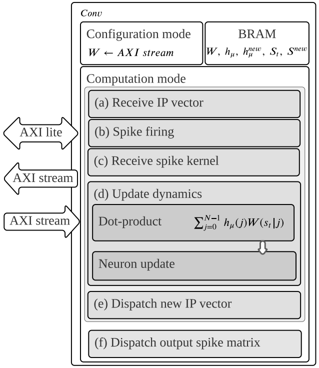


Figure: Conv processing unit.

Processing Unit

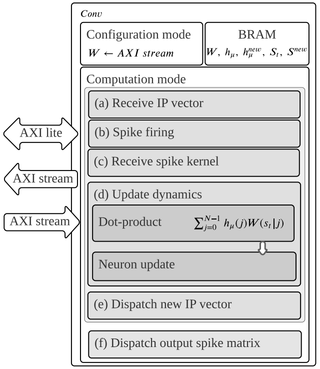

Conv
Configuration mode
$W \leftarrow AXI\ stream$
BRAM
$W,\ h_\mu,\ h_\mu^{new},\ S_t,\ S^{new}$
Computation mode
(a) Receive IP vector
(b) Spike firing
(c) Receive spike kernel
(d) Update dynamics
Dot-product $\sum_{j=0}^{N-1} h_\mu(j)W(s_t|j)$
Neuron update
(e) Dispatch new IP vector
(f) Dispatch output spike matrix
AXI lite
AXI stream
AXI stream

Figure: Conv processing unit.

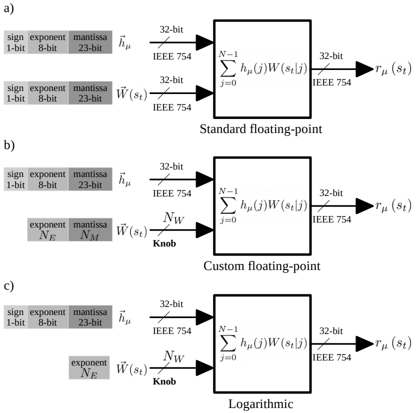


Figure: Dot-product hardware module.

Hybrid Dot-Product Approximation

$$r_\mu\,(s_t) = \sum_{j=0}^{N-1} h_\mu(j) W(s_t|j) \qquad (1)$$

$$E_{\min} = \log_2(\min_{\forall i}(W(i))) \qquad (2)$$

$$N_E = \lceil \log_2(|E_{\min}|) \rceil \qquad (3)$$

$$N_W = N_E + N_M \qquad (4)$$

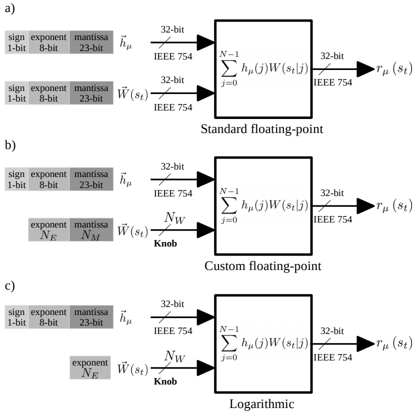


Figure: Dot-product hardware module.

Dot-Product with Standard Floating-Point (IEEE 754)

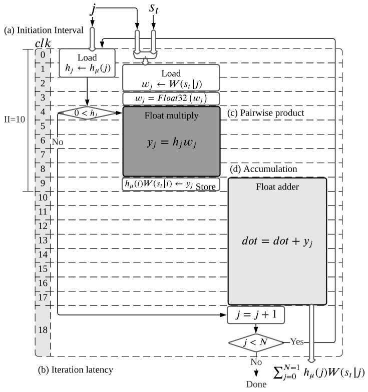


Figure: Dot-product hardware module with standard floating-point computation.

$$L_{f32} = 10N + 9$$

Dot-Product with Hybrid Custom Floating-Point Approximation

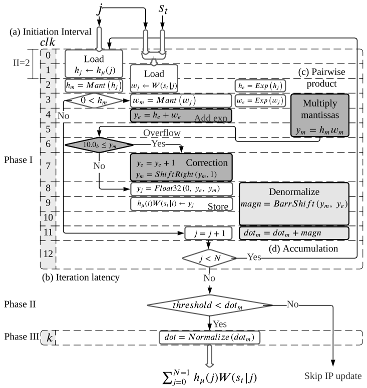


Figure: Dot-product hardware module with hybrid custom floating-point approximation.

$$L_{custom} = 2N + 11$$

Dot-product with Hybrid Logarithmic Approximation

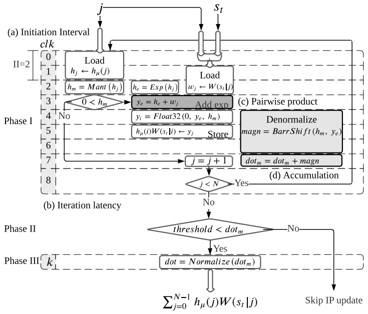


Figure: Dot-product hardware module with hybrid logarithmic approximation.

$$L_{custom} = 2N + 7$$

Acceleration with Standard Floating-Point

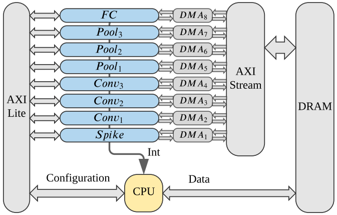


Figure: System overview of the top-level architecture with 8 processing units.

Acceleration with Standard Floating-Point

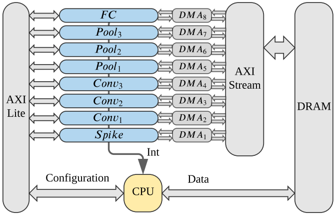


Figure: System overview of the top-level architecture with 8 processing units.

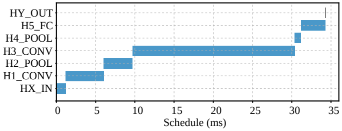


Figure: Computation on embedded CPU.

Acceleration with Standard Floating-Point

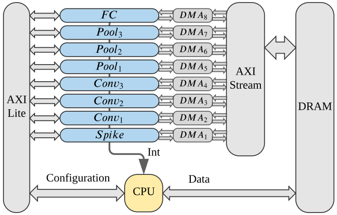


Figure: System overview of the top-level architecture with 8 processing units.

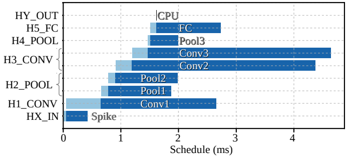


Figure: Performance of processing units with standard floating-point.

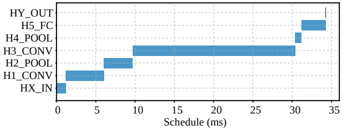


Figure: Computation on embedded CPU.

Acceleration with Standard Floating-Point

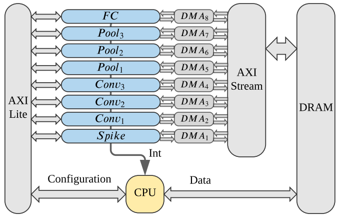


Figure: System overview of the top-level architecture with 8 processing units.

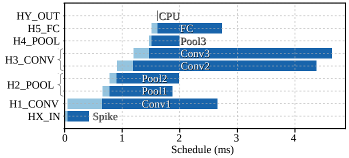


Figure: Performance of processing units with standard floating-point.

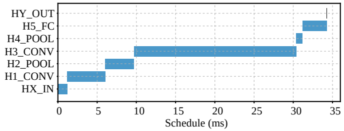


Figure: Computation on embedded CPU.

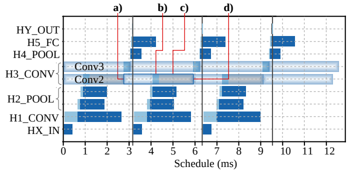


Figure: Performance bottleneck of cyclic computation on processing units with standard floating-point.

Acceleration with Custom Floating-Point

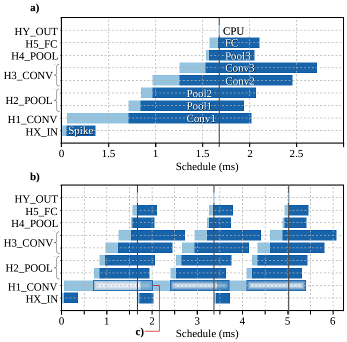


Figure: Performance on processing units with hybrid 8-bit floating-point.

Acceleration with Custom Floating-Point

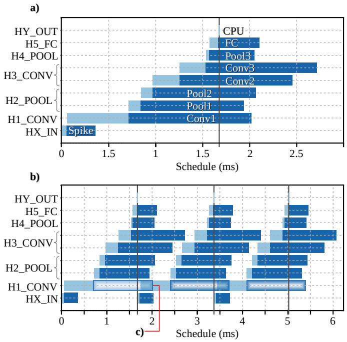


Figure: Performance on processing units with hybrid 8-bit floating-point.

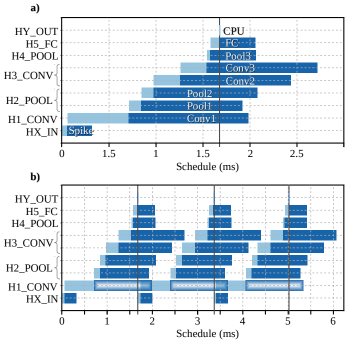


Figure: Performance of processing units with hybrid 4-bit logarithmic approximation.

Noise tolerance

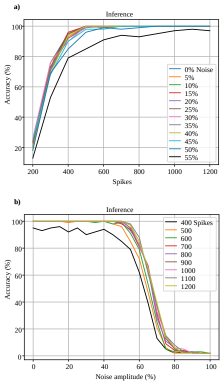


Figure: Noise tolerance with standard floating-point.

Noise tolerance

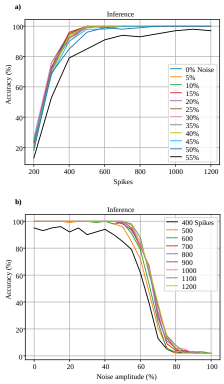


Figure: Noise tolerance with standard floating-point.

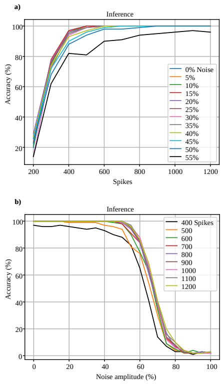


Figure: Noise tolerance with hybrid 8-bit floating-point approximation.

Noise tolerance

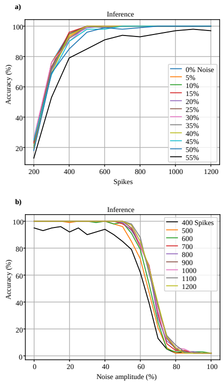


Figure: Noise tolerance with standard floating-point.

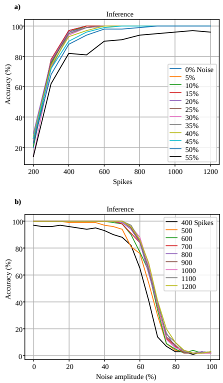


Figure: Noise tolerance with hybrid 8-bit floating-point approximation.

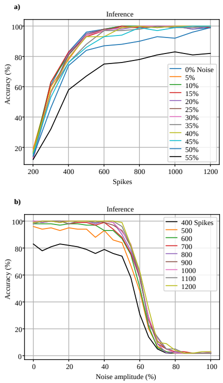


Figure: Noise tolerance with hybrid 4-bit logarithmic approximation.

Accelerator Implementations

Table: Accelerator implementations.

Platform implementation	Power (W)	Clk (MHz)	Latency (ms)	Acceleration	Accuracy (%)
Standard floating-point	2.420	200	3.18	10.7x	98.98
Hybrid floating-point 8-bit	2.369	200	1.67	20.5x	98.97
Hybrid Logarithmic 4-bit	2.324	200	1.67	20.5x	98.84