Effiicient FPGA implementations of Machine Learning Algorithms

Philip Leong (梁恆惠) | Computer Engineering Laboratory
School of Electrical and Information Engineering,
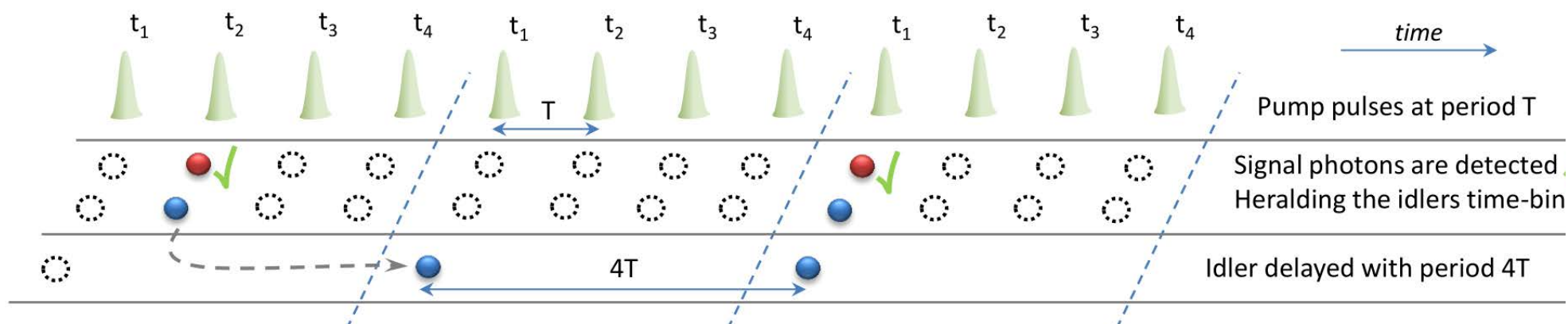The University of Sydney

THE UNIVERSITY OF
SYDNEY

› Focuses on how to use parallelism to solve demanding problems

  - Novel architectures, applications and design techniques using VLSI, FPGA and parallel computing technology

› Research

  - Reconfigurable computing

  - Machine learning

  - Nanoscale interfaces

Initially expectation : Heralded single photon rate should enhance significantly without degrading coincidence to accidental ratio (CAR)
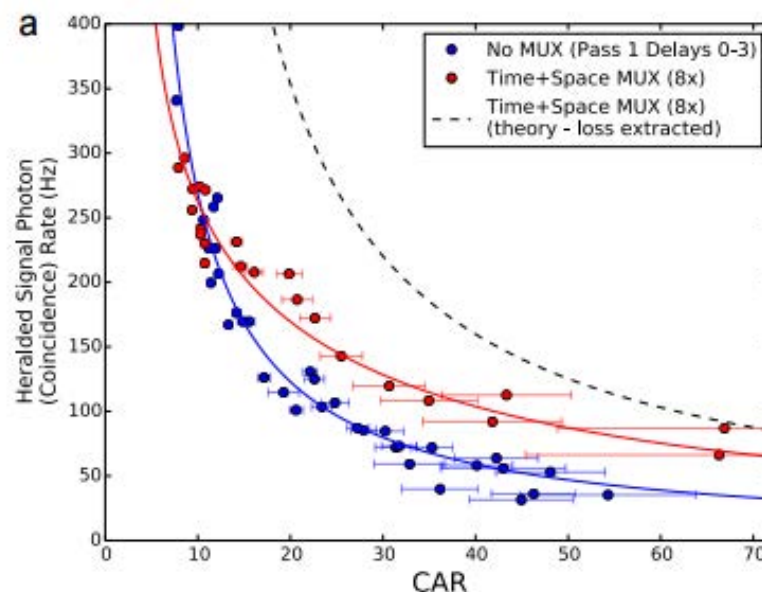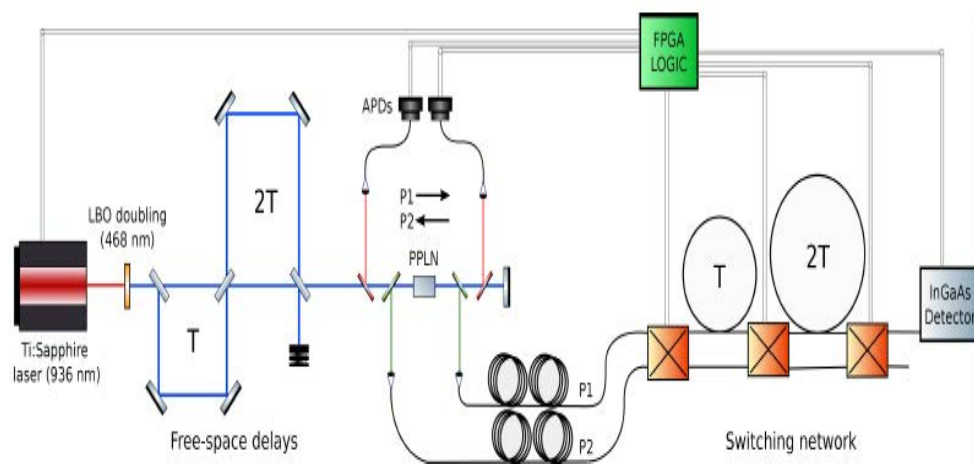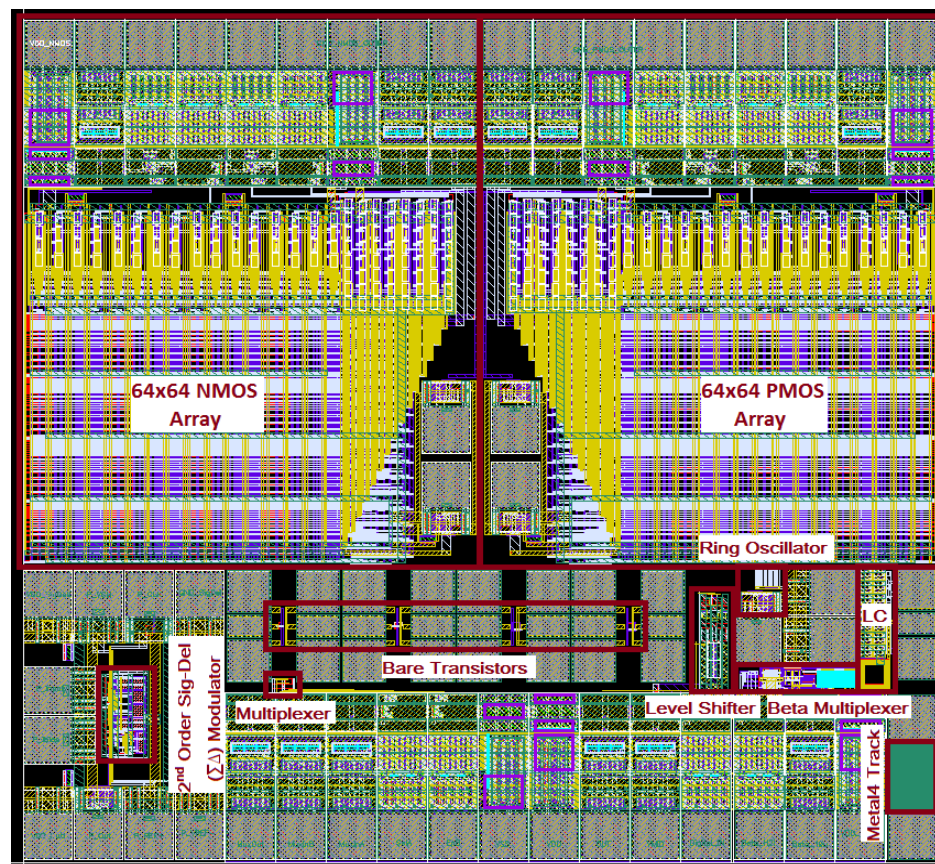

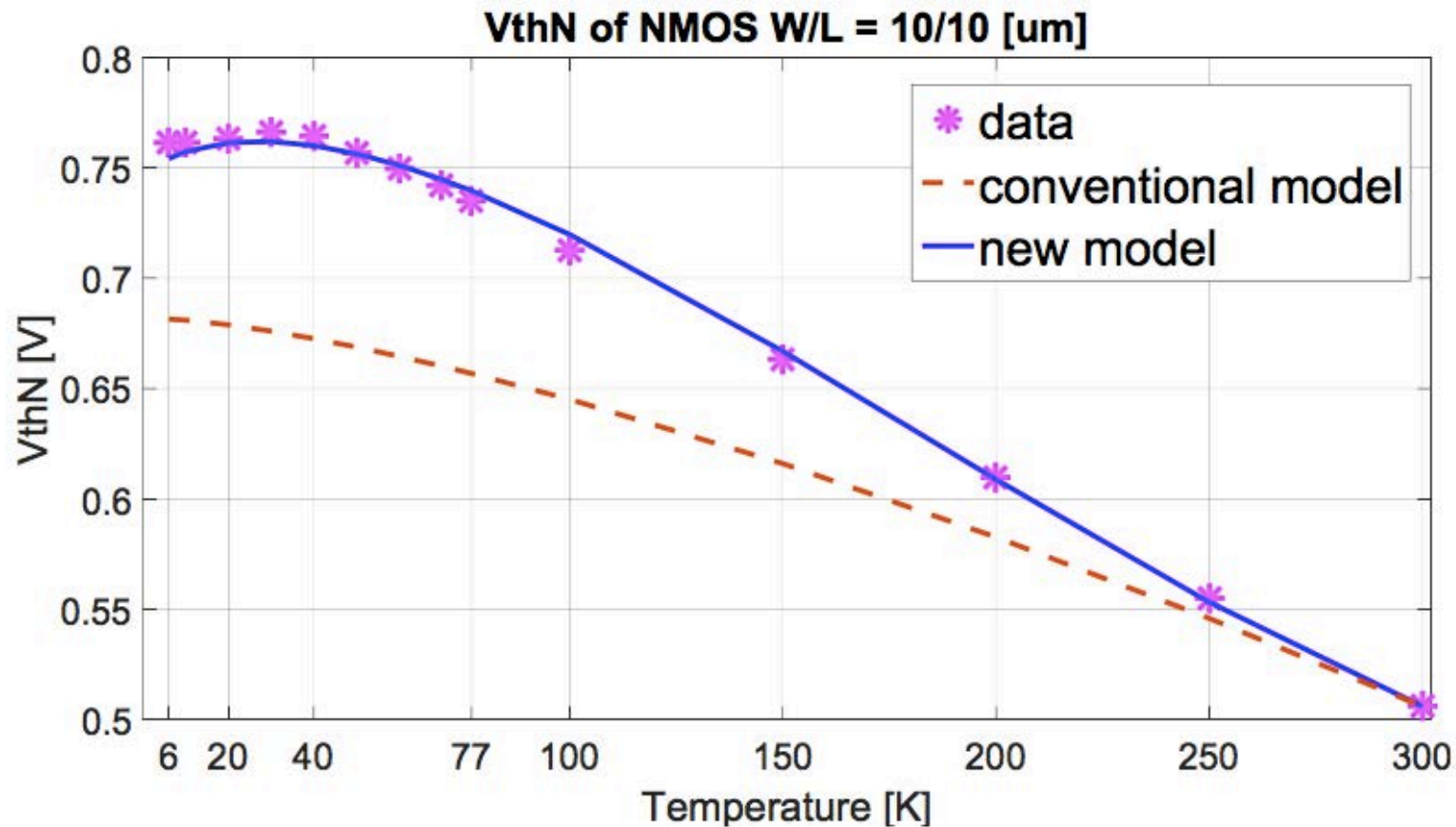Nature Comms 7(10853), 2016

Enhancement : 33%~59%

Purposes:

- To characterize CMOS transistors

- Evaluate matching property of CMOS transistors

- Test analog circuits: ADC, Level Shifter, Ring Oscillator, Beta Multiplier, Passive LC circuit, Metal tracks, …

Layout of QNL2_CMOS

IEEE Electron Device Letters, 38:847–850, 2017

VthN of NMOS W/L = 10/10 [um]

Modified Booth Radix-4 datapath is split into 2 sections, each with its own critical path

Non-zero encodings take $\overline{K}\tau$ and zero take τ

TVLSI, v. 27, no. 4, 2019
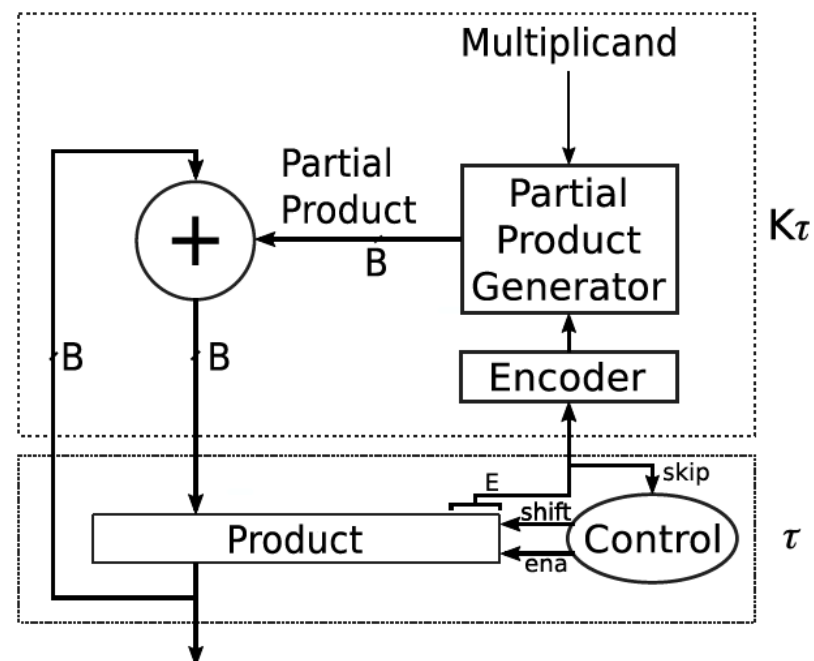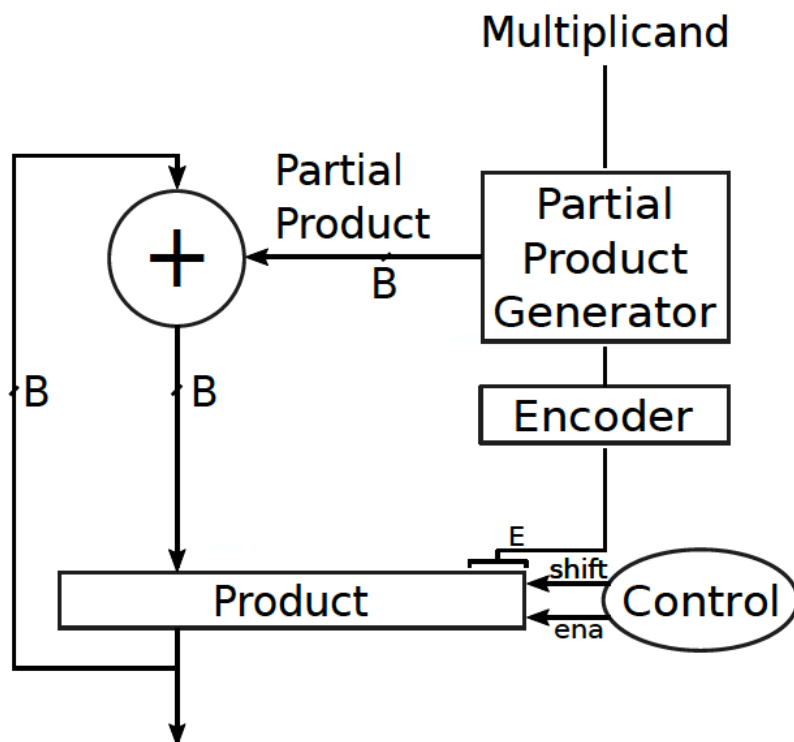
› FPGAs can implement ML algorithms with better performance and energy through

- **E**xploration– easily try different ideas to arrive at a good solution

- **P**arallelism – so we can arrive at an answer faster

- **I**ntegration – so interfaces are not a bottleneck

- **C**ustomisation – problem-specific designs to improve efficiency

› **Describe our work on efficient implementations of ML that use these ideas**

EPIC

› **Exploration (Online kernel methods)**

› **Parallelisation**

› **Integration**

› **Customisation**

THE UNIVERSITY OF
SYDNEY

Challenges in measurement and control are becoming feasible

› Significant improvements in ML algorithms but cannot keep up with sources e.g. hyperspectral imager or wireless transceiver

› Need extremely high throughput

› In control applications we need low latency e.g. triggering data collection in Large Hadron Collider

› Need very low latency
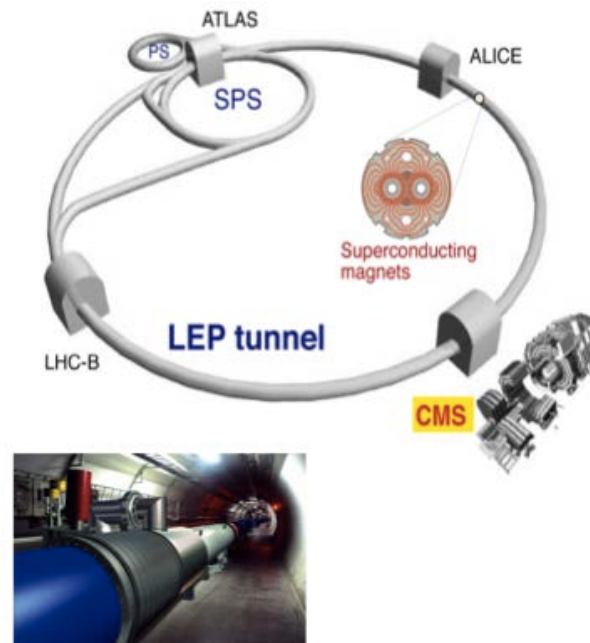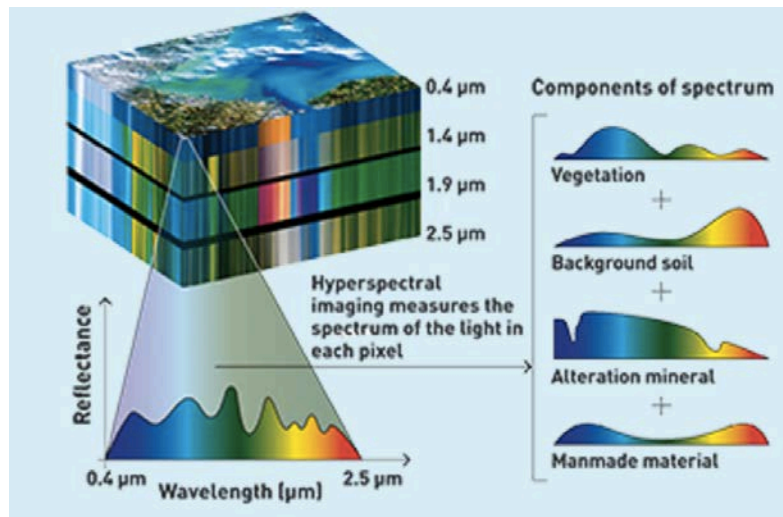
Improvements in throughput and latency enable new applications!

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right)$$

ϕ

Input Space
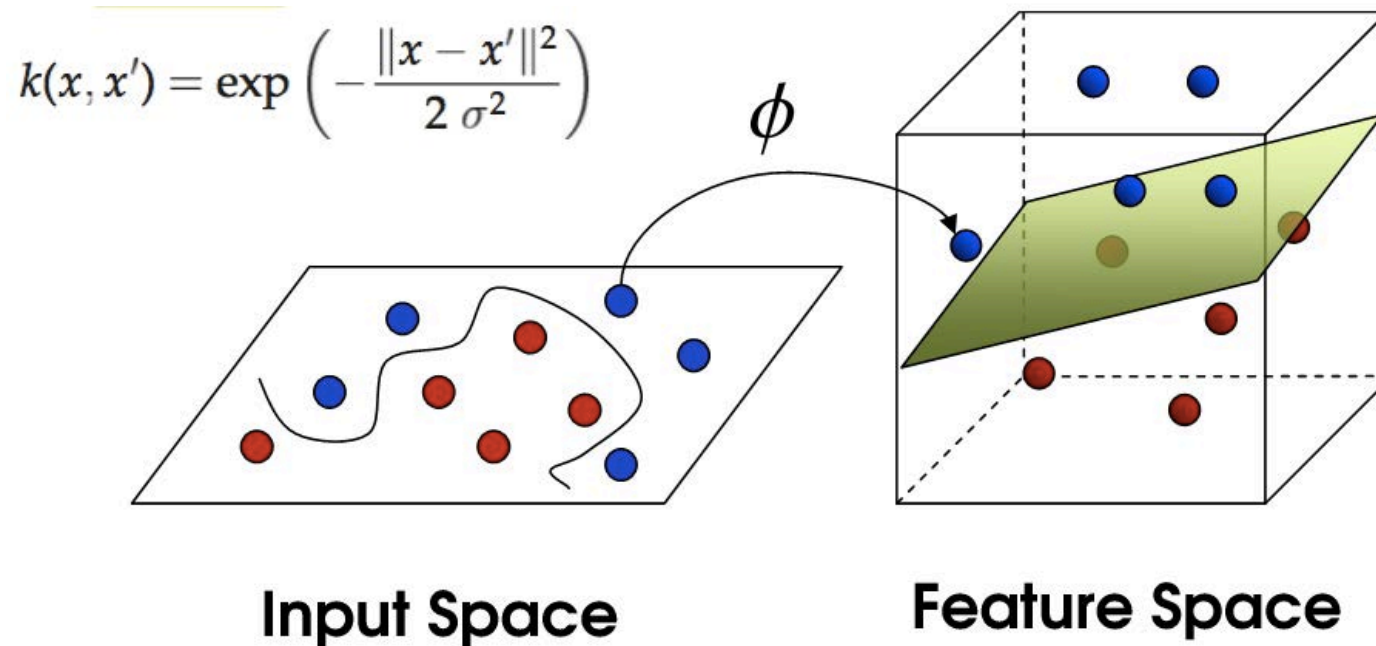
Feature Space

› Choose high dimensional feature space (so easily separable)

› Use kernel trick to avoid computing the mapping (fast)

› Do regression/classification using $\quad f(x_i) = \sum\limits_{j=1}^{N} \alpha_j \kappa(x_i, v_j)$

› Kernel is a similarity function

  - defined by an implicit mapping ϕ, (original space to feature space)

$$\kappa(x,x') = \phi(x)^T \phi(x') = \left\langle \phi(x), \phi(x') \right\rangle$$

  - e.g. Linear kernel $\kappa(x,x')=<x,x'>$

  - e.g. Polynomial kernel $\kappa(x,x')=(1+<x,x'>)^d$ for d=2: $\phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

  - e.g. Gaussian kernel (universal approximator**)** $k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$

    - $\Phi(x)$ infinite in dimension!

› **Modify linear ML techniques to kernel ones by replacing dot products with the kernel function (kernel trick)**

  - e.g. linear discriminant analysis, logistic regression, perceptron, SOM, K-means, PCA, ICA, LMS, RLS, …

  - **While we only describe prediction here, also applied to training equations**
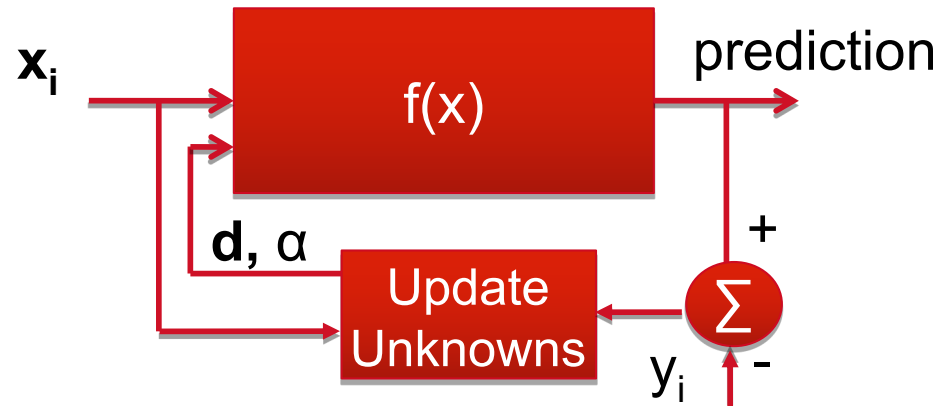
$$f(x) = \sum_{i=1}^{N} \alpha_i \, \kappa(x, d_i)$$

Dictionary Entry

› "Kernel Method" → $\kappa(x, x') : \mathbb{R}^d \to \mathbb{R}^D$, *where* $D \gg d$

› Dictionary → subset of the input data of length **N**

› Computation and Memory scale O(**N**d)

› **BUT…** **N** scales linearly with the dataset size

Exact Kernel Methods

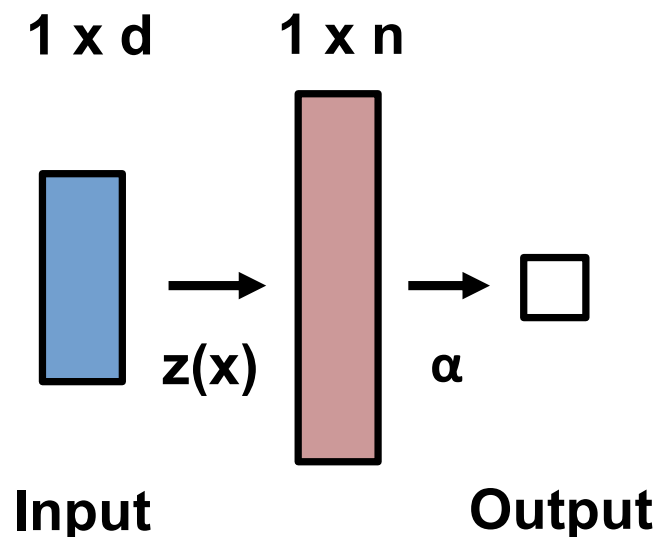$$f(x) = \sum_{i=1}^{N} \alpha_i \, \kappa(x, d_i)$$

Random Kernel Expansion

$$f(x) = \sum_{i=1}^{n} \alpha_i \, z(x)$$

$$z(x) = \frac{1}{\sqrt{n}} cos(\mathbf{W}x)$$

** Only for k(x,x') = k(x-x',0)

Define z(x):

› Approximates $\kappa(x, x')$
› MV + Non-Linear Activation
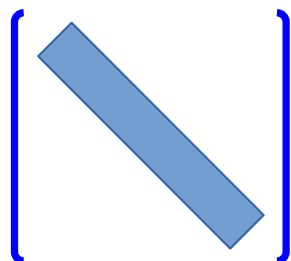(i.e. like Multilayer Perceptron)
› W is **fixed** and **random**

1 x d   **1 x n**

z(x)   **α**

Input   **Output**

› Computes **z(x)** efficiently by replacing **Wx** with combinations of random diagonal matrices and Hadamard transforms

$$z(x) = \frac{1}{\sqrt{n}} \cos(\boldsymbol{V}x), \qquad \text{where} \quad \mathbf{V}x = [\boldsymbol{Q}_1 x, \boldsymbol{Q}_2 x, \cdots, \boldsymbol{Q}_h x]$$

$$\boldsymbol{Q}_j x = \boldsymbol{SHGPHB}x$$

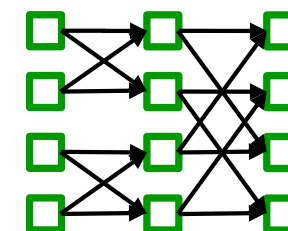** Each $Q_j x$ is an independent dxd transform

B, G, S

P

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

H

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$
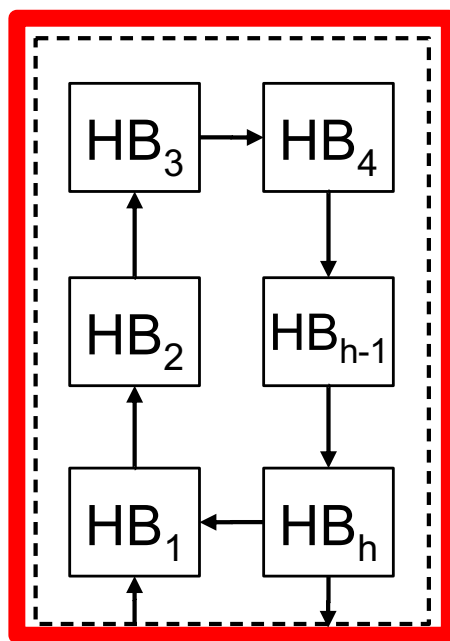
Fast Hadamard Transform (dxd)

Memory = O(3n)   **Time = O(n log d)**

› $Vx = [\boldsymbol{Q}_1 x, \boldsymbol{Q}_2 x, \cdots, \boldsymbol{Q}_h x]$

Top

HB

PE
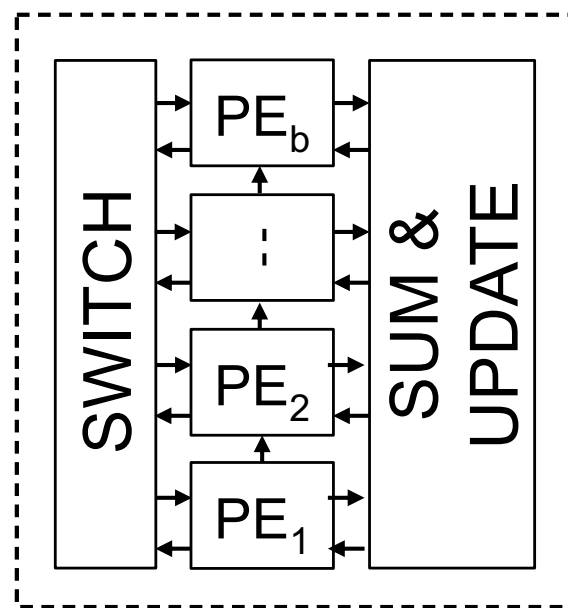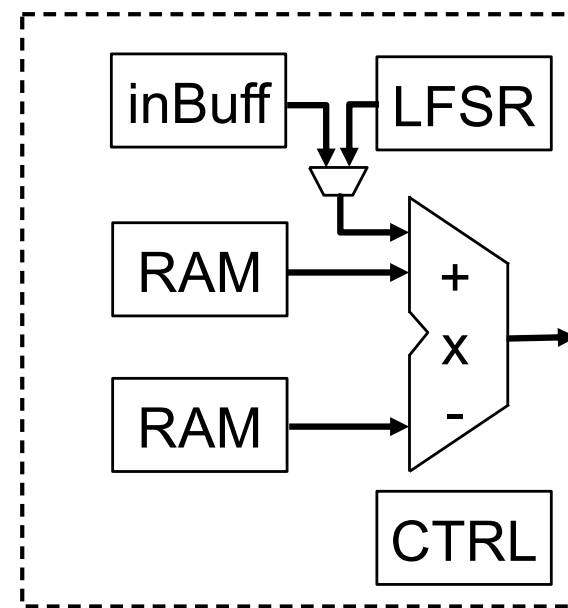
› $\mathbf{V}x = [\boldsymbol{Q}_1 x, \boldsymbol{Q}_2 x, \cdots, \boldsymbol{Q}_h x]$

› Block of **b** PEs (i.e. $\boldsymbol{Q}_q x$)

Top

HB

PE
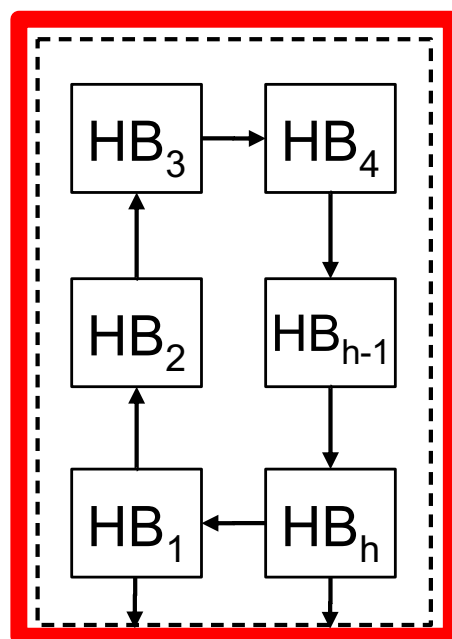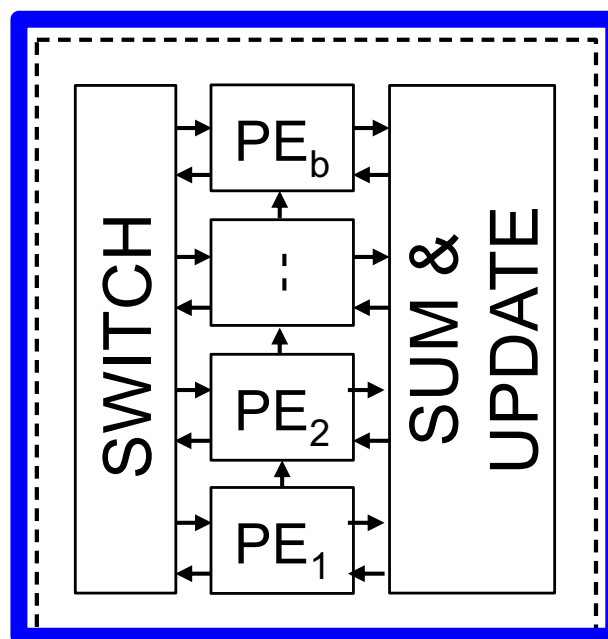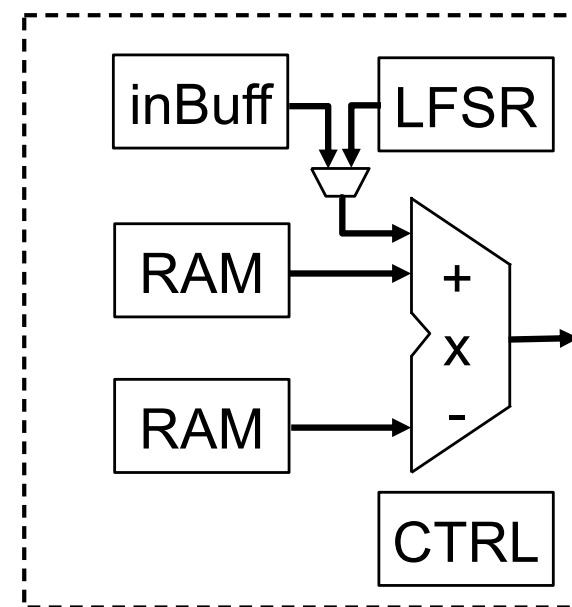
› $\mathbf{V}x = [\boldsymbol{Q}_1 x, \boldsymbol{Q}_2 x, \cdots, \boldsymbol{Q}_h x]$

› Block of **b** PEs (i.e. $\boldsymbol{Q}_q x$)

› General PE: 18-bit ALU, RAMs, Control Unit, LFSR
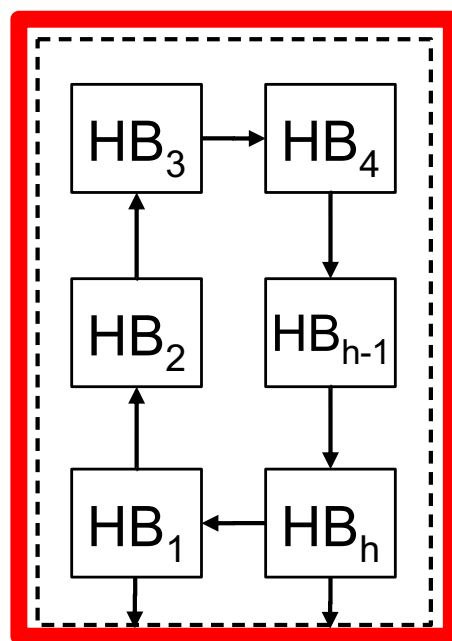
Top   **HB**   **PE**

Impl.	dim.	n	bw	Lat. (cyc)	Fmax (MHz)	Exec (ns)	Th.put (Gb/s)
NORMA (V7, '15)	8	200	18	10	127	7.87	18.3
KNLMS (V7, '15)	8	16	32	207	314	3.18	80.4
CPU (Le et. '13)	1024	16.4k	32			58e4	0.06
FASTFOOD (V7)	1024	16.4k	18	1893	432	23.7e3	7.77
FASTFOOD (KU035)	**8192**	**90.1k**	18	16930	**508**	17.2e3	8.57

Impl.	dim.	n	bw	Lat. (cyc)	Fmax (MHz)	Exec (ns)	Th.put (Gb/s)
NORMA (V7, '15)	8	200	18	10	127	7.87	18.3
KNLMS (V7, '15)	8	16	32	207	314	3.18	80.4
CPU (Le et. '13)	1024	16.4k	32			58e4	0.06
FASTFOOD (V7)	1024	16.4k	18	1893	432	23.7e3	7.77
FASTFOOD (KU035)	**8192**	**90.1k**	18	16930	**508**	17.2e3	8.57

› Supports much larger problems

Impl.	dim.	n	bw	Lat. (cyc)	Fmax (MHz)	Exec (ns)	Th.put (Gb/s)
Braiding (V7, '15)	8	200	18	10	127	7.87	18.3
KNLMS (V7, '15)	8	16	32	207	314	3.18	80.4
CPU (Le et. '13)	1024	16.4k	32			58e4	0.06
FASTFOOD (V7)	1024	16.4k	18	1893	432	23.7e3	7.77
FASTFOOD (KU035)	**8192**	**90.1k**	18	16930	**508**	17.2e3	8.57

› Supports much larger problems

› High speed design

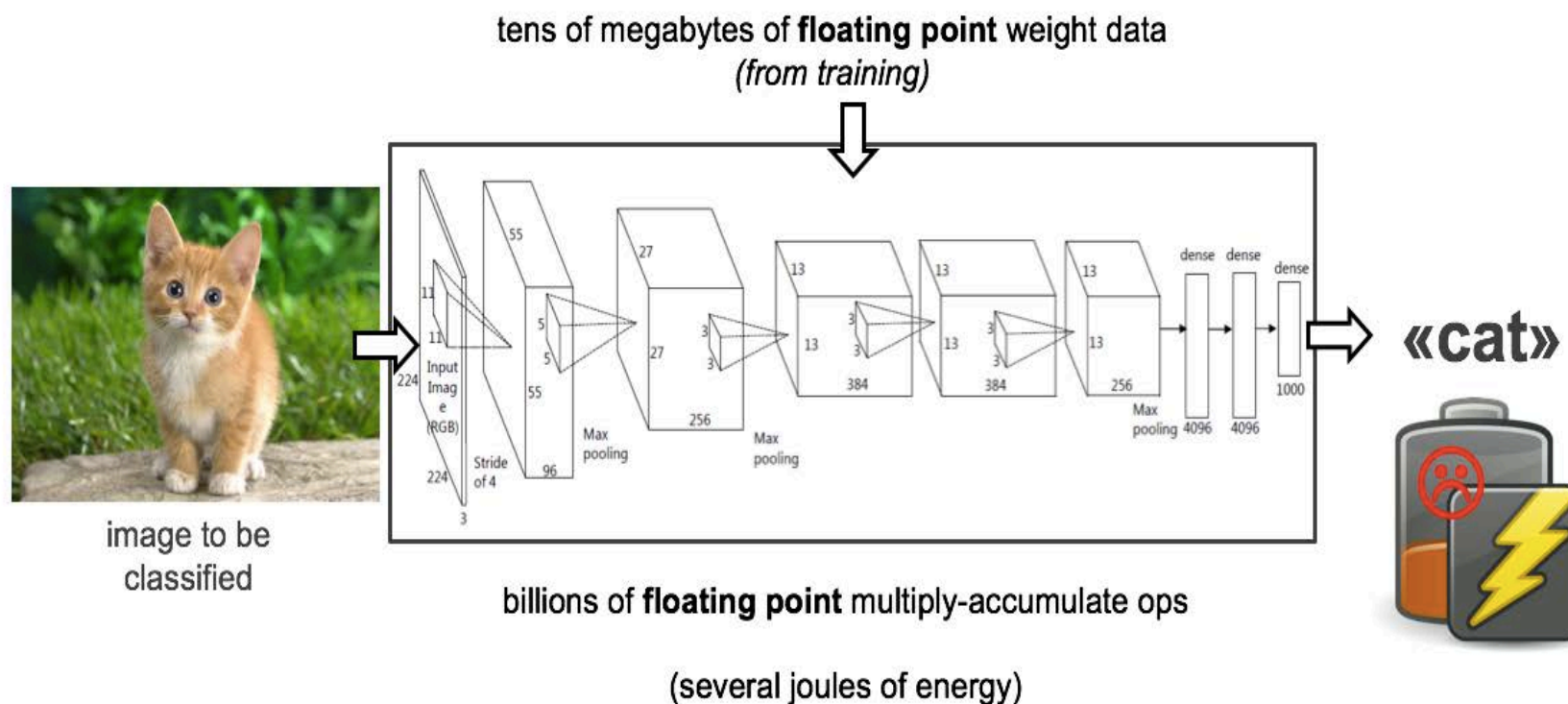| Impl.	dim.	n	bw	Lat. (cyc)	Fmax (MHz)	Exec (ns)	Th.put (Gb/s)
Braiding (V7, '15)	8	200	18	10	127	7.87	18.3
KNLMS (V7, '15)	8	16	32	207	314	3.18	80.4
CPU (Le et. '13)	1024	16.4k	32			58e4	0.06
FASTFOOD (V7)	1024	16.4k	18	1893	432	23.7e3	7.77
FASTFOOD (KU035)	**8192**	**90.1k**	18	16930	**508**	17.2e3	8.57

› Supports much larger problems

› High speed design

› **245x** speed-up over a CPU

› **Exploration**

› **Parallelisation (Low Precision Neural Network)**

› **Integration**

› **Customisation**

Slides from Yaman Umuroglu et. al., "FINN: A framework for fast, scalable binarized neural network inference," FPGA'17

tens of megabytes of **floating point** weight data
(from training)

image to be classified

billions of **floating point** multiply-accumulate ops

(several joules of energy)

«cat»

› The extreme case of quantization

- Permit only two values: +1 and -1

- Binary weights, binary activations

- Trained from scratch, not truncated FP
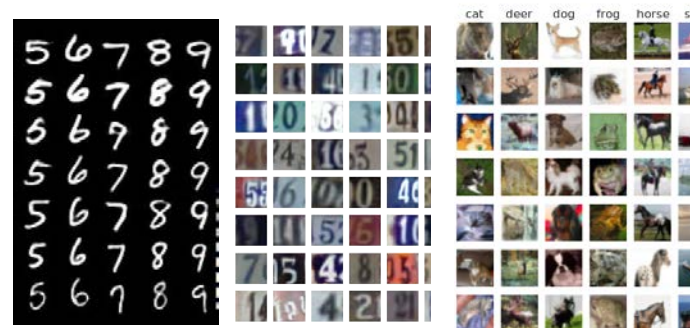
› Courbariaux and Hubara et al. (NIPS 2016)

- Competitive results on three smaller benchmarks

- Open source training flow

- Standard "deep learning" layers

  - Convolutions, max pooling, batch norm, fully connected…

	MNIST	SVHN	CIFAR-10
Binary weights & activations	0.96%	2.53%	10.15%
FP weights & activations	0.94%	1.69%	7.62%
BNN accuracy loss	-0.2%	-0.84%	-2.53%

% classification error (lower is better)

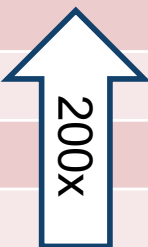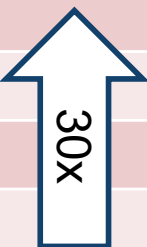## Vivado HLS estimates on Xilinx UltraScale+ MPSoC ZU19EG

› *Much* smaller datapaths

- Multiply becomes XNOR, addition becomes popcount

- No DSPs needed, everything in LUTs

- Lower cost per op = more ops every cycle

› *Much* smaller weights

- Large networks can fit entirely into on-chip memory (OCM)

- More bandwidth, less energy compared to off-chip

Precision	Peak TOPS	On-chip weights
1b	~66	~70 M
8b	~4	~10 M
16b	~1	~5 M
32b	~0.3	~2 M

200x   30x

› **fast** inference with **large BNNs**

		Accuracy	FPS	Power (chip)	Power (wall)	kFPS / Watt (chip)	kFPS / Watt (wall)	Precision
FINN	MNIST, SFC-max	95.8%	12.3 M	7.3 W	21.2 W	1693	583	1
	MNIST, LFC-max	98.4%	1.5 M	8.8 W	22.6 W	177	269	1
	CIFAR-10, CNV-max	80.1%	21.9 k	3.6 W	11.7 W	6	2	1
	SVHN, CNV-max	94.9%	21.9 k	3.6 W	11.7 W	6	2	1
Prior Work	MNIST, Alemdar et al.	97.8%	255.1 k	0.3 W	-	806	-	2
	CIFAR-10, TrueNorth	83.4%	1.2 k	0.2 W	-	6	-	1
	SVHN, TrueNorth	96.7%	2.5 k	0.3 W	-	10	-	1

Max accuracy loss: ~3%

10 – 100x better performance

CIFAR-10/SVHN energy efficiency comparable to TrueNorth ASIC

› Who would be willing to incur a loss in accuracy?

› Can we get better accuracy with a little more hardware?

- To compute quantised weights from FP weights

$$\boldsymbol{Q}_l = sign(\boldsymbol{W}_l) \odot \boldsymbol{M}_l$$

with,

$$M_{l_{i,j}} = \begin{cases} 1 & \text{if} & \left| W_{l_{i,j}} \right| \geq \eta_l \\ 0 & \text{if} & -\eta_l < W_{l_{i,j}} < \eta_l \end{cases}$$

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

where $\boldsymbol{M}$ represents a masking matrix, η is the quantization threshold hyperparameter (0 for binarised)

- Make approximation $W_l \approx \alpha_l Q_l$, $Q_l \in \mathcal{C}$
- $\mathcal{C}$ is the codebook, $\mathcal{C} \in \{C_1, C_2, \ldots\}$ e.g. $\mathcal{C} = \{-1, +1\}$ for binary, $\mathcal{C} = \{-1, 0, +1\}$ for ternary
- A diagonal matrix α_l is defined by the vector $\alpha_l = \left[\alpha_l^1, \ldots, \alpha_l^m\right]$:

$$\alpha = diag(\alpha) := \begin{bmatrix} \alpha^1 & 0 & .. & 0 & 0 \\ 0 & \alpha^2 & .. & : & 0 \\ : & : & .. & \alpha^{m-1} & : \\ 0 & 0 & .. & 0 & \alpha^m \end{bmatrix}$$

- Train by solving

$$\alpha_l^{\cdot} = \underset{\alpha}{\arg\min}\, E(\alpha, \boldsymbol{Q}) \quad s.t. \quad \alpha \geq 0,\ \boldsymbol{Q}_{l_{i,j}} \in \mathbb{C}$$

› **More fine-grained quantisation can improve approximation of weights**

Pixel-wise scaling   Row-wise scaling (layer-wise also option)
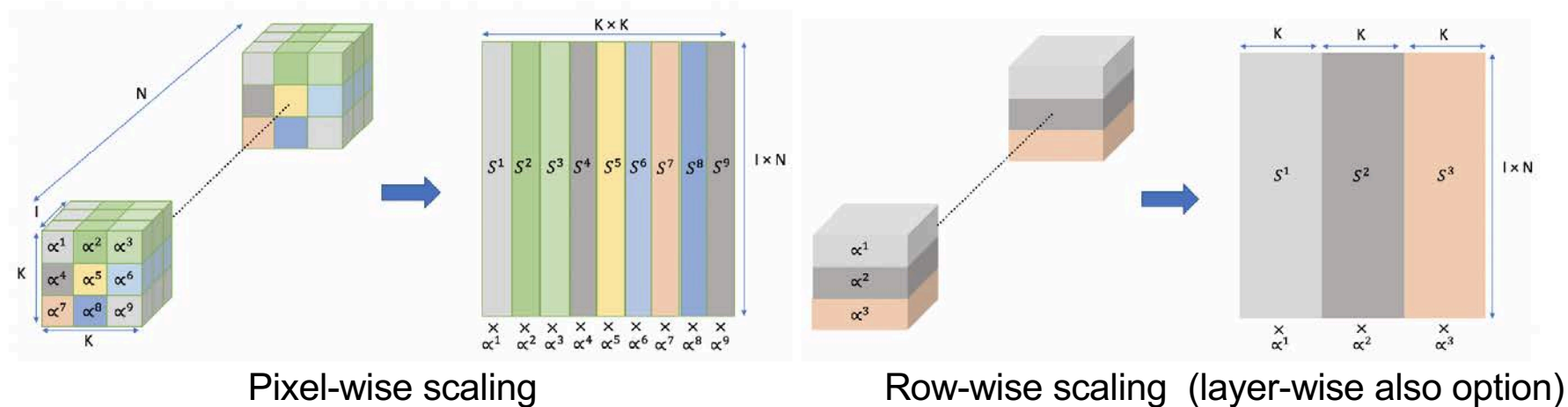
› For K filters, I Input feature maps of dimension FxF, N output feature maps

› $P = K^2 I N F^2$

Method	Scalars	Ops
Layer (DoReFa)	1	P
Row (SYQ)	K	P
Pixel (SYQ)	K^2	P
Asymmetric (TTQ)	2	$P + Z$
Grouping (FGQ)	$K^2 N/4$	P
Channel (HWGQ/BWN)	N	P
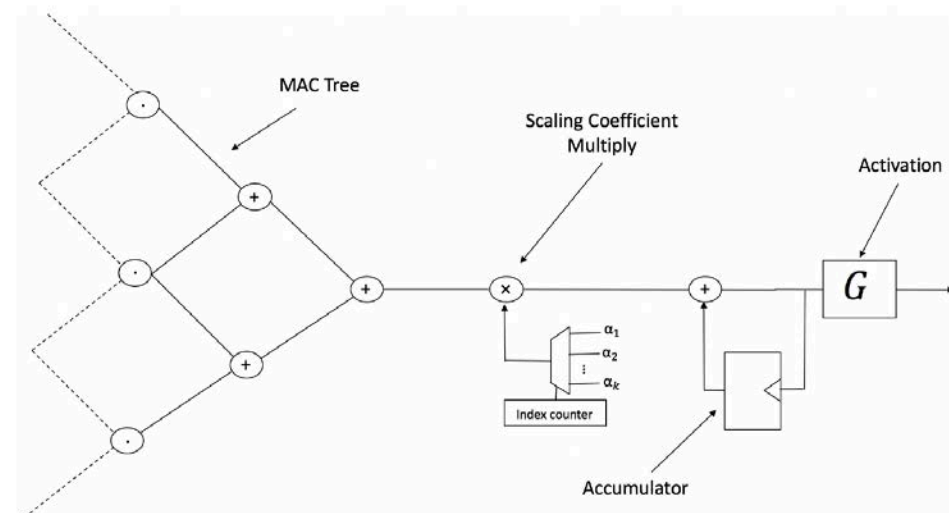
› Full precision for 1st and last layers, CONV layers pixel-wise, FC layer-wise

Model		1-8	2-8	Baseline	Reference
AlexNet	Top-1	**56.6**	**58.1**	56.6	57.1
	Top-5	**79.4**	**80.8**	80.2	80.2
VGG	Top-1	**66.2**	**68.7**	69.4	-
	Top-5	**87.0**	**88.5**	89.1	-
ResNet-18	Top-1	**62.9**	**67.7**	69.1	69.6
	Top-5	**84.6**	**87.8**	89.0	89.2
ResNet-34	Top-1	**67.0**	**70.8**	71.3	73.3
	Top-5	**87.6**	**89.8**	89.1	91.3
ResNet-50	Top-1	**70.6**	**72.3**	76.0	76.0
	Top-5	**89.6**	**90.9**	93.0	93.0

Baseline is floating-point, reference https://github.com/facebook/fb.resnet.torch (ResNet)
and https://github.com/BVLC/caffe (AlexNet)

Model	Weights	Act.	Top-1	Top-5
DoReFa-Net [33]	1	2	49.8	-
QNN [15]	1	2	51.0	73.7
HWGQ [2]	1	2	52.7	76.3
SYQ	**1**	**2**	**55.4**	**78.6**
DoReFa-Net [33]	1	4	53.0	-
SYQ	**1**	**4**	**56.2**	**79.4**
BWN [24]	1	32	56.8	79.4
SYQ	**1**	**8**	**56.6**	**79.4**
SYQ	**2**	**2**	**55.8**	**79.2**
FGQ [21]	2	8	49.04	-
TTQ [34]	2	32	57.5	79.7
SYQ	**2**	**8**	**58.1**	**80.8**

Model	Weights	Act.	Top-1	Top-5
BWN [24]	1	32	60.8	83.0
SYQ	**1**	**8**	**62.9**	**84.6**
TWN [19]	2	32	65.3	86.2
INQ [32]	2	32	66.0	87.1
TTQ [34]	2	32	66.6	87.2
SYQ	**2**	**8**	**67.7**	**87.8**

ResNet-18

Model	Weights	Act.	Top-1	Top-5
HWGQ [2]	1	2	64.6	85.9
SYQ	**1**	**4**	**68.8**	**88.7**
SYQ	**1**	**8**	**70.6**	**89.6**
FGQ [21]	2	4	68.4	-
SYQ	**2**	**4**	**70.9**	**90.2**
FGQ [21]	2	8	70.8	-
SYQ	**2**	**8**	**72.3**	**90.9**

ResNet-50

EPIC

› **Exploration**

› **Parallelisation**

› **Integration (radio frequency machine learning)**

› **Customisation**

THE UNIVERSITY OF
SYDNEY

› Processing radio frequency signals remains a challenge

- high bandwidth and low latency difficult to achieve

› Autoencoder to do anomaly detection

Train so $\tilde{x}$ x (done in an unsupervised manner)
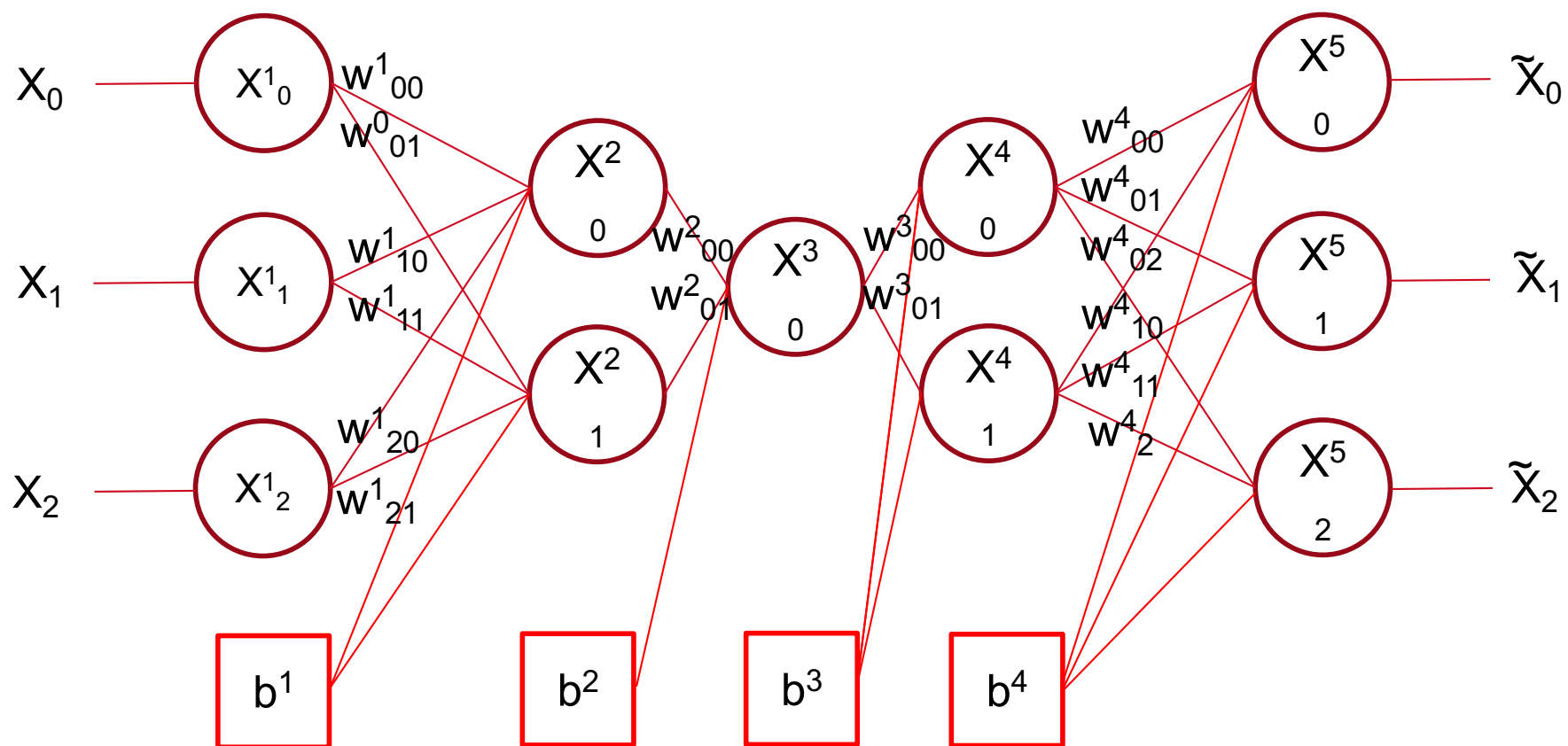
› Anomaly if distance between autoencoder output and input large

› FPGA has sufficiently high performance to process each sample of waveform at 200 MHz!

- This minimises latency and maximises throughput

- Weights trained on uP and updated on FPGA without affecting inference
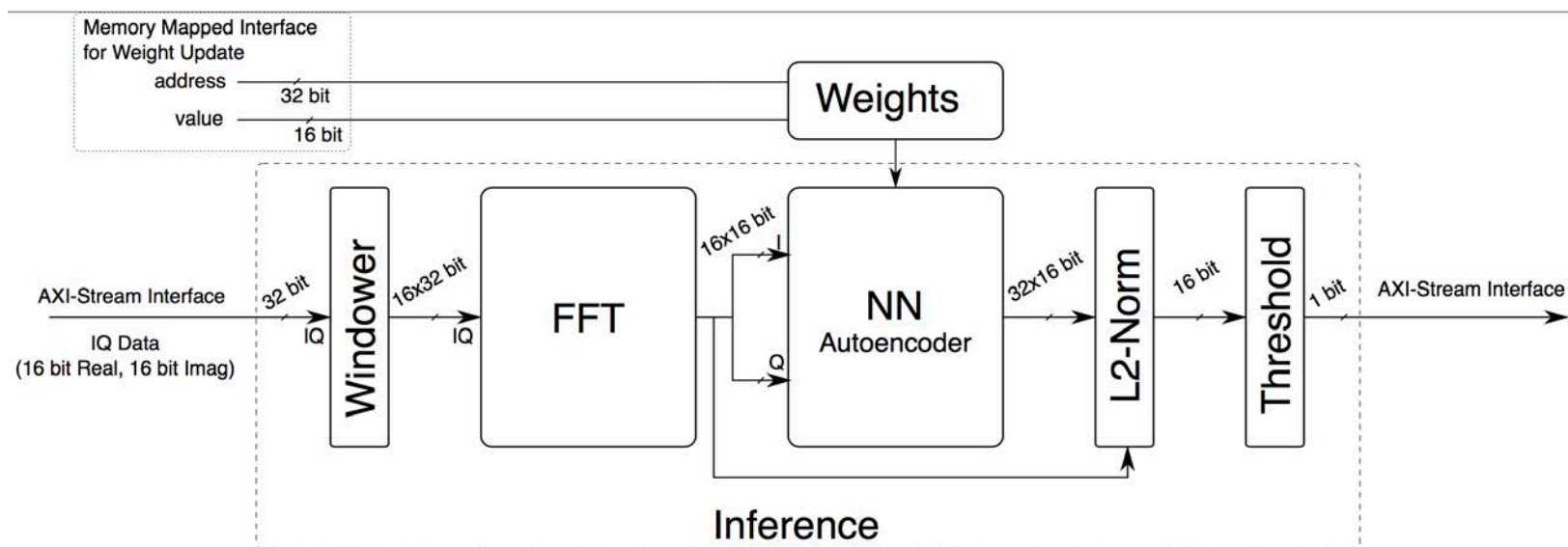
Implemented on Ettus X310 platform
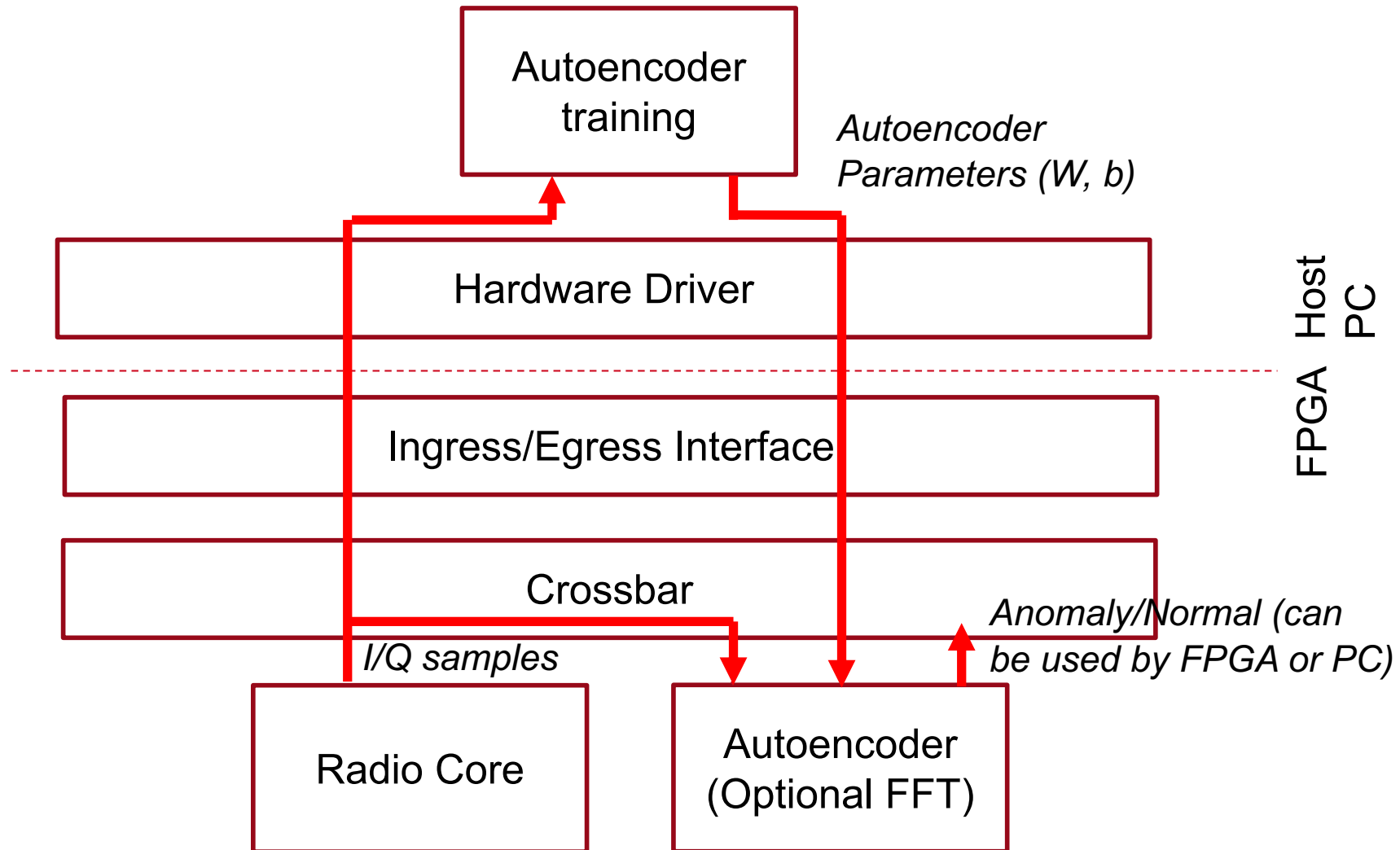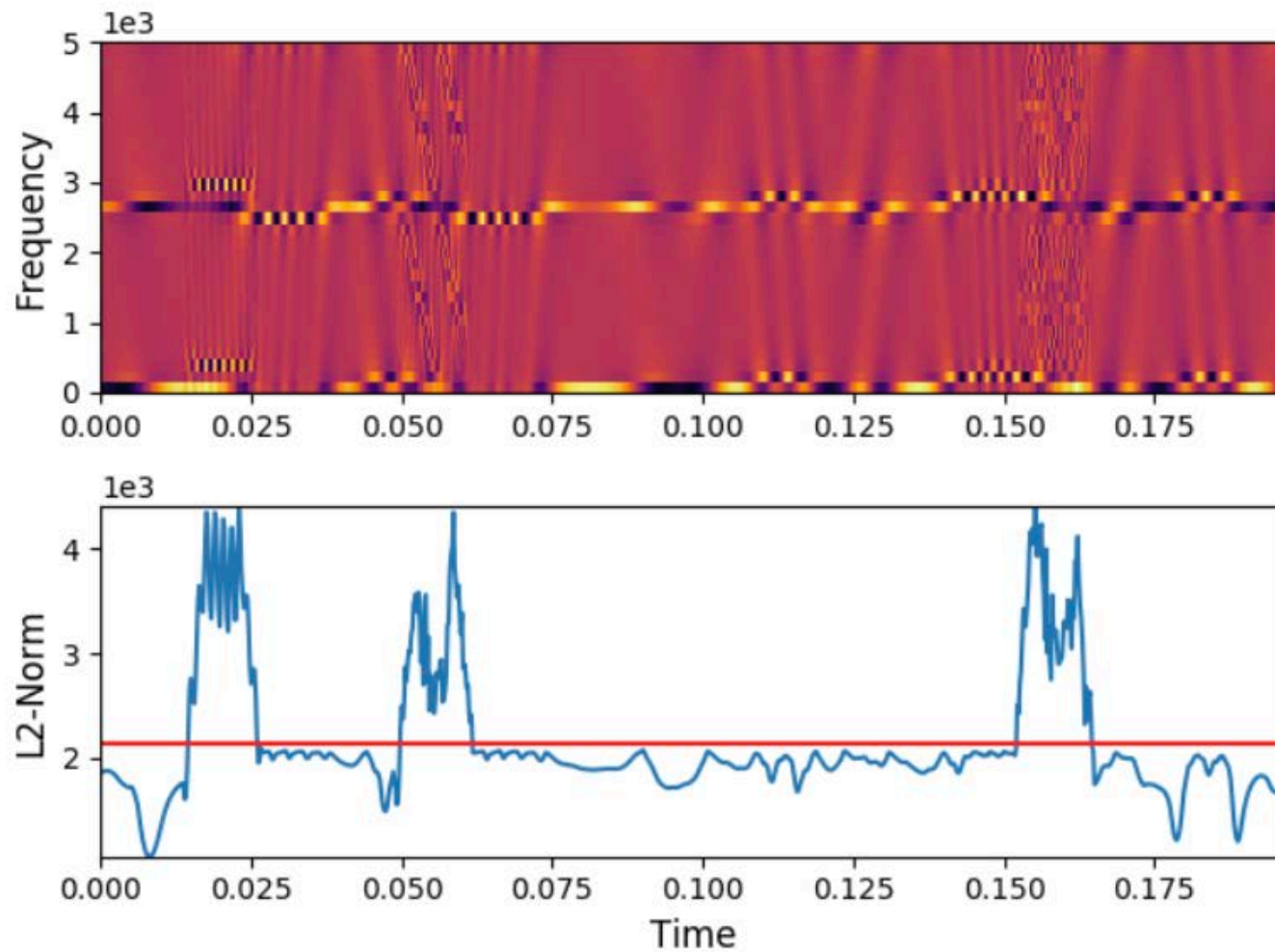
Typical SDR latency >> 1 ms

Module	II	Latency (cycles)	BRAM	DSP	FF	LUT
Windower	1	0	0	0	1511	996
FFT	1	8	0	48	4698	2796
NN	1	17	4	1280	213436	13044
L_2-Norm	1	4	0	32	1482	873
Thres	1	0	0	0	3	21
Weight Update	258	257	0	0	21955	4528
Inference (FFT+NN)	1	37	1068	1360	241522	45448
Inference (NN)	1	29	1068	1312	236824	42652
Total	N/A	N/A	1068	1360	263477	49976
Total Util.	N/A	N/A	67%	88%	51%	19%

Operation	Throughput	Latency
Inference(FFT+NN)	5ns	185ns
Inference(NN)	5ns	105ns
Weight Update	1290ns	1285ns

› **Exploration**

› **Parallelisation**

› **Integration**

› **Customisation (PIR-DSP)**

THE UNIVERSITY OF
SYDNEY

› DNNs for embedded applications share two features to reduce computation and storage requirements

- Low precision (from 1-16 bits)

- Depthwise separable convolutions

Standard Convolution (standard)

Depthwise Convolution (DW)

Pointwise Convolution (PW)
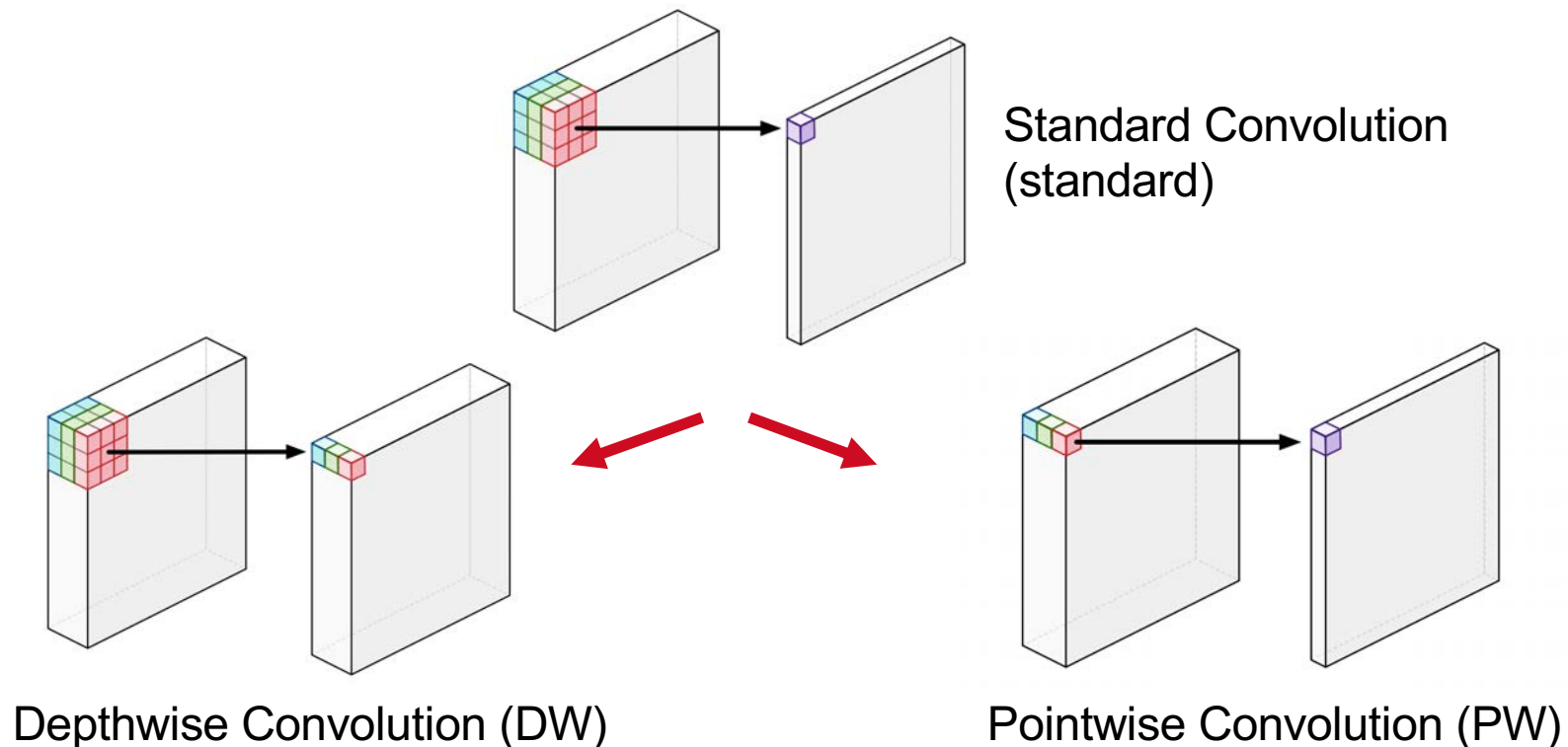
Computation and Storage for Embedded DNNs

Distribution of # of MACs

Distribution of # of parameters

SqueezeNet
ShuffleNet-v2
MobileNet-v2
NASNet-A4@1056
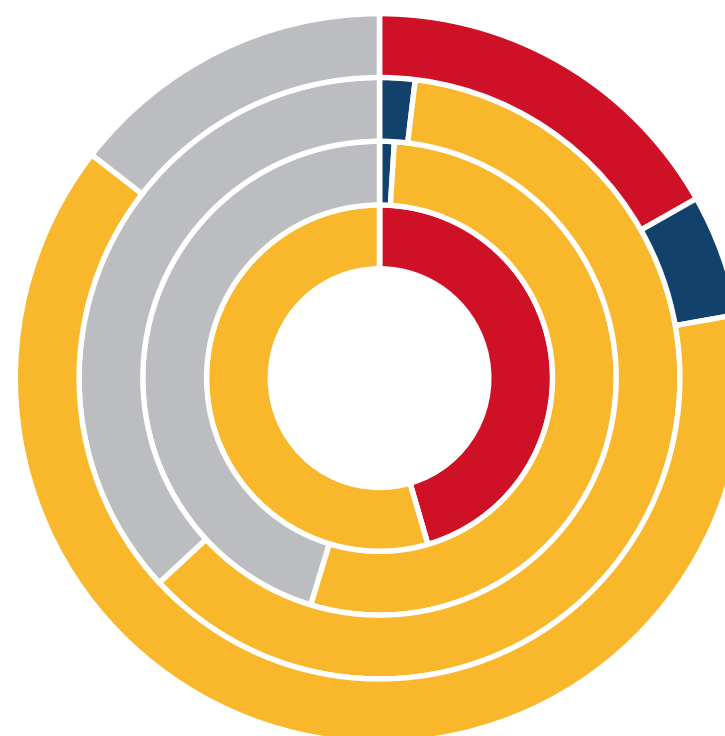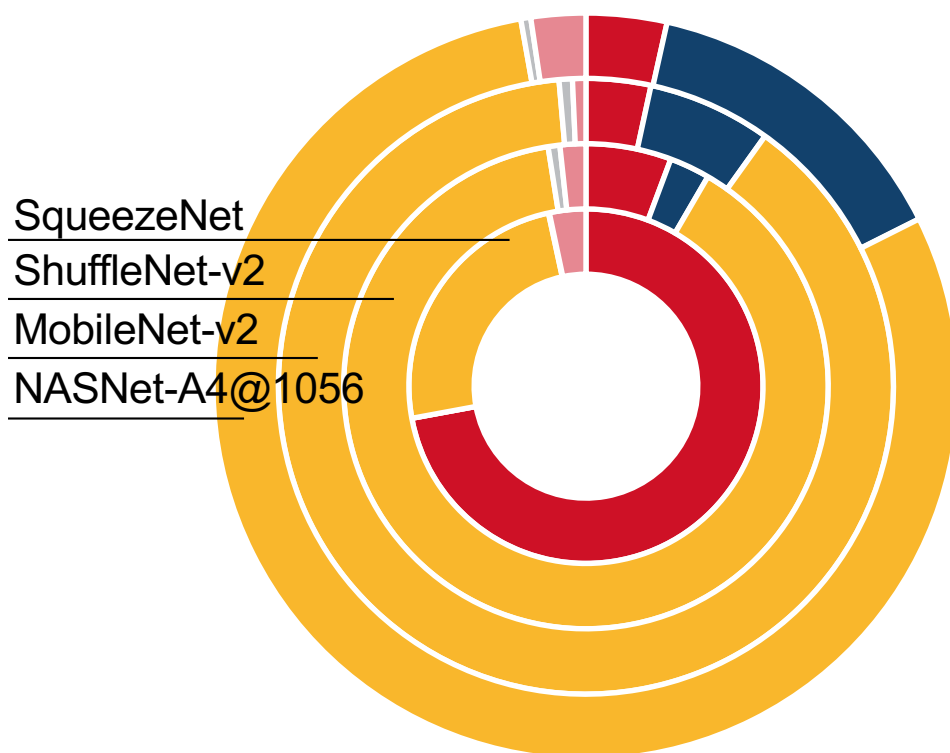
■ Standard ■ DW ■ PW ■ FC ■ Other   ■ Standard ■ DW ■ PW ■ FC ■ Other

› Optimise FPGA DSP architecture to better support

- Efficient implementation of embedded DNNs

- Wordlengths down to ternary and binary

› Talk will focus on convolutions

› PIR-DSP: Optimized version of DSP48

- **P**recision: Multiplier architecture

- **I**nterconnect: Shift-Reg

- **R**euse : RF/FIFO

DSP48
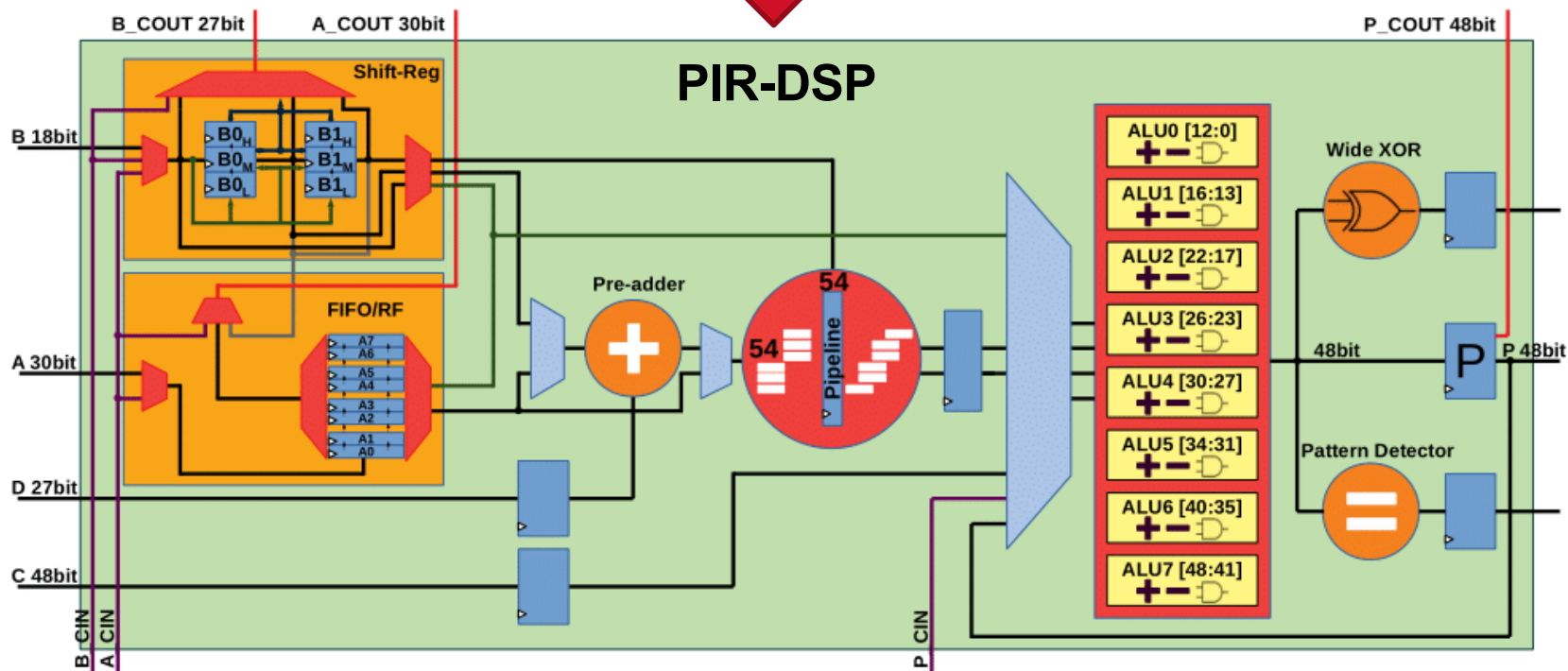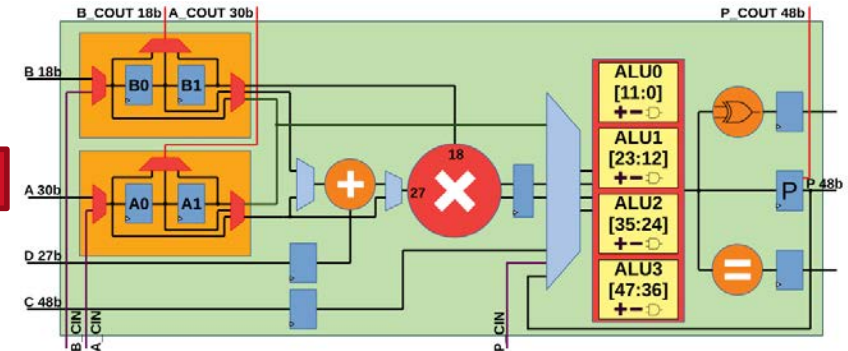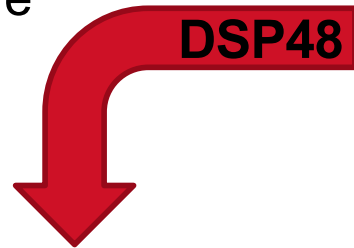
PIR-DSP

› Based on two approaches:

1. Chopping

2. Recursive decomposition
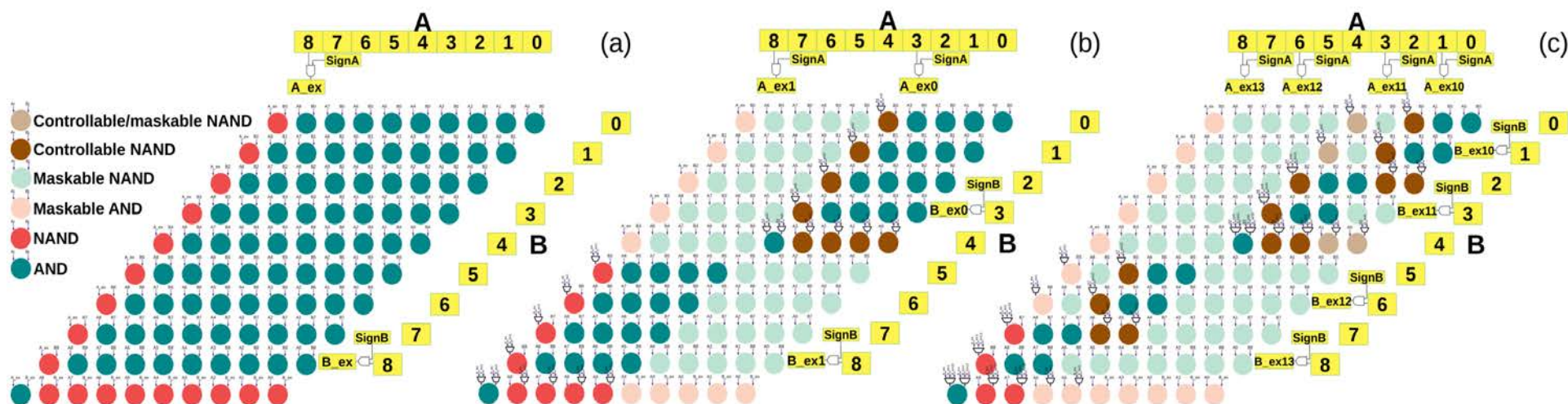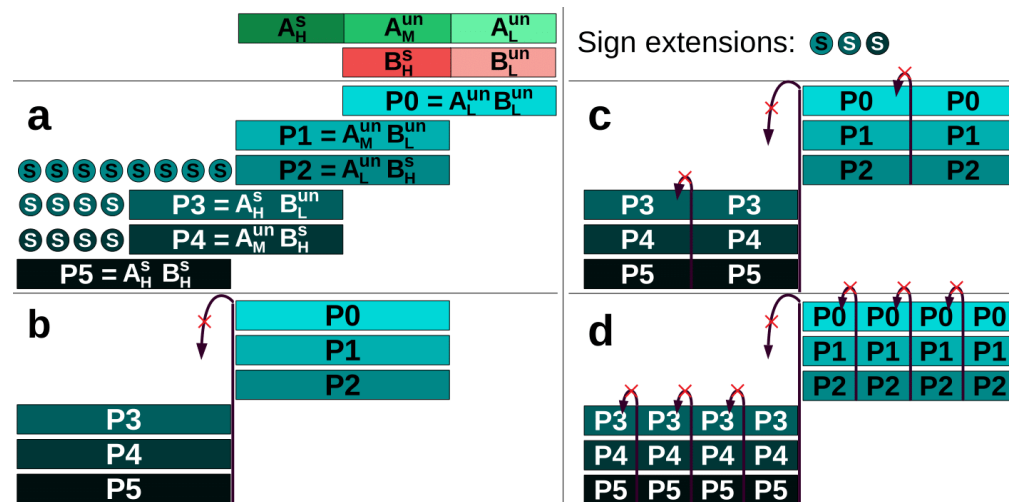
Parameterised Decomposable MAC unit

› Notation: M×NCijDk

› PIR-DSP multiplier: 27×18C32D2

  - **C**hopping factors 3 and 2 respectively for 27 and 18

    - (27=9+9+9)×(18=9+9)

    - Six 9×9 multiplier

  - **D**ecomposing factor is 2

    - Each 9×9 multiplier decomposes to Two **4×4** or Four **2×2** multipliers

› **PIR-DSP** Modes:

  - One   27×18   → 1 MAC

  - Two   9×9 + 9×9 + 9×9   → 6 MACs

  - Four   4×4 + 4×4 + 4×4   → 12 MACs

  - Eight 2×2 + 2×2 + 2×2   → 24 MACs
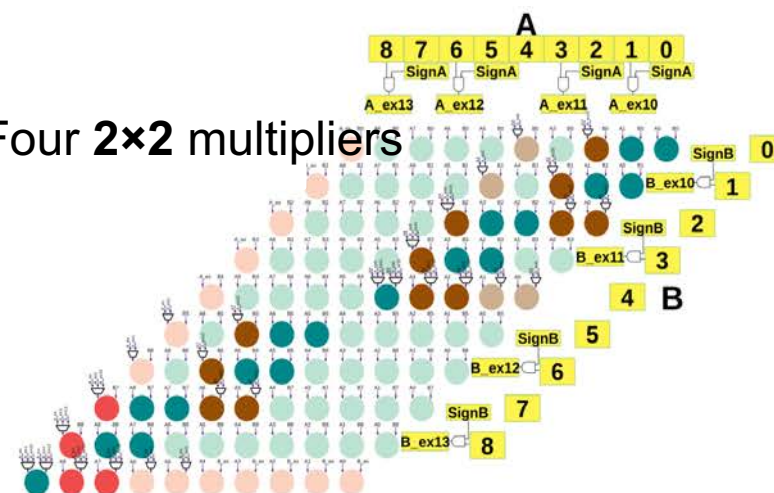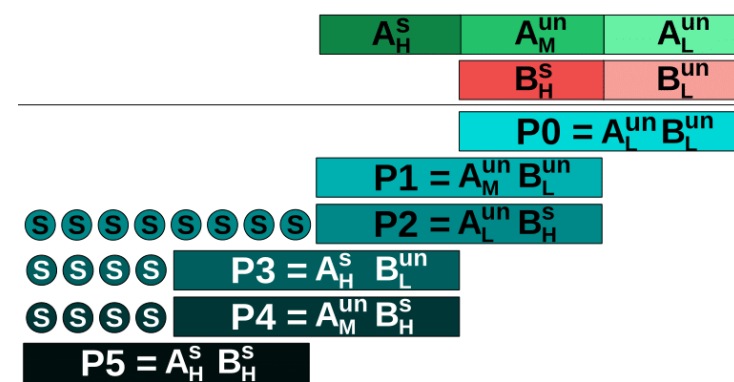
› Three types of convolutions

1- **Depth-wise**: using three PIR-DSPs

2- **Standard**: based on depth-wise convolution implementation and adding the partial results
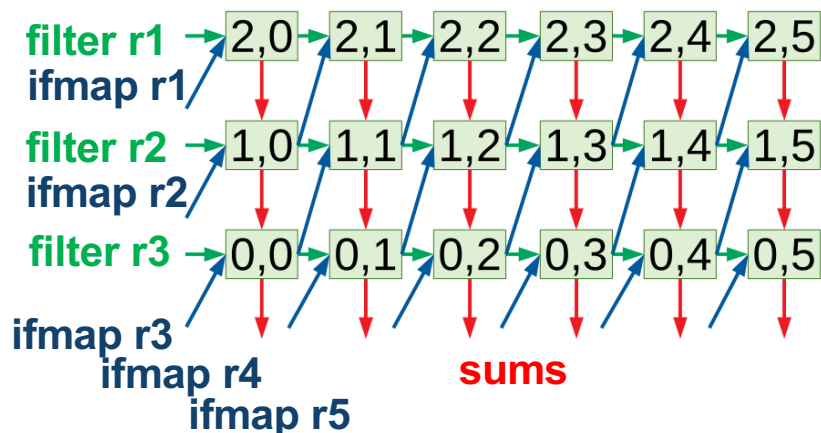
2D systolic array (Eyeriss)   conventional   ours   depthwise convolution
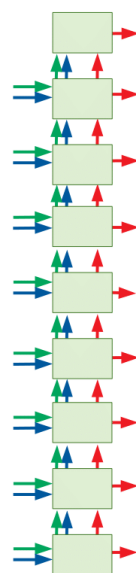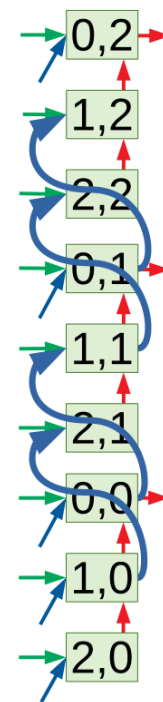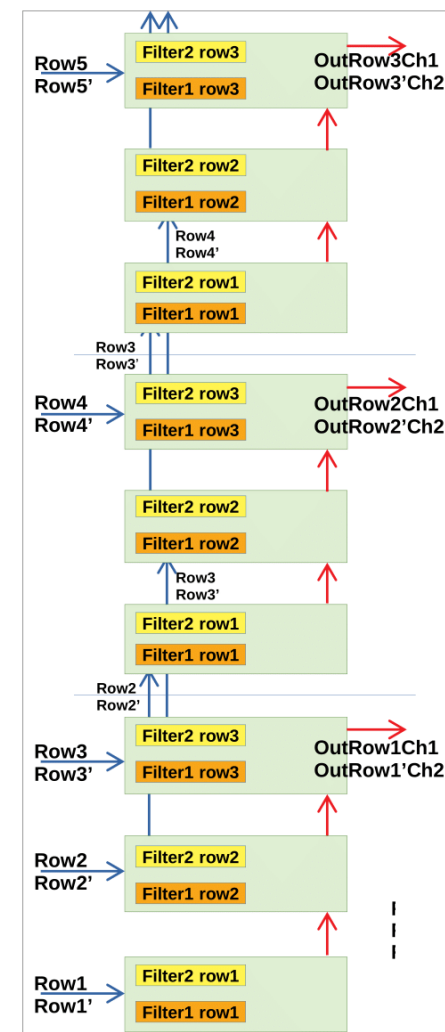
3- **Point-wise**
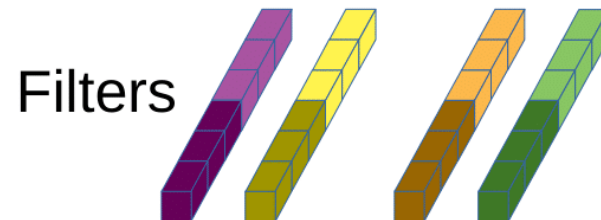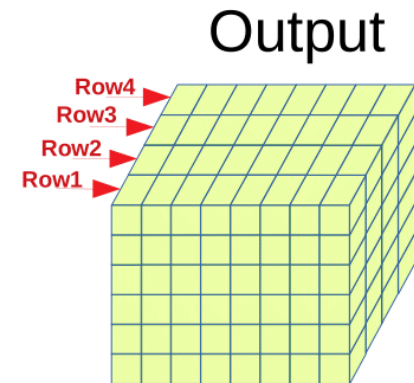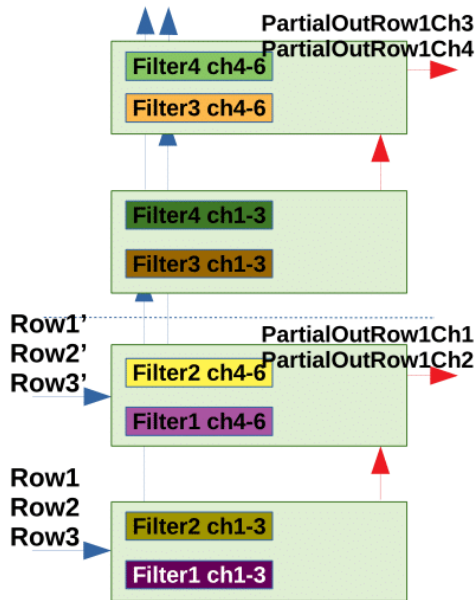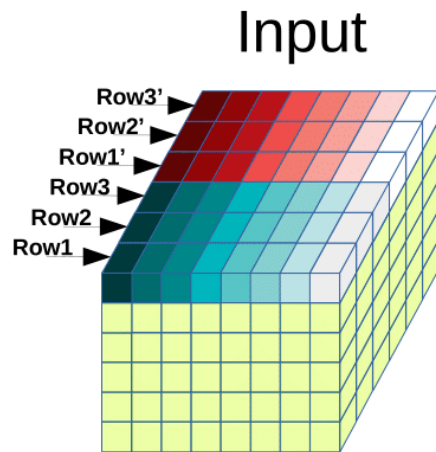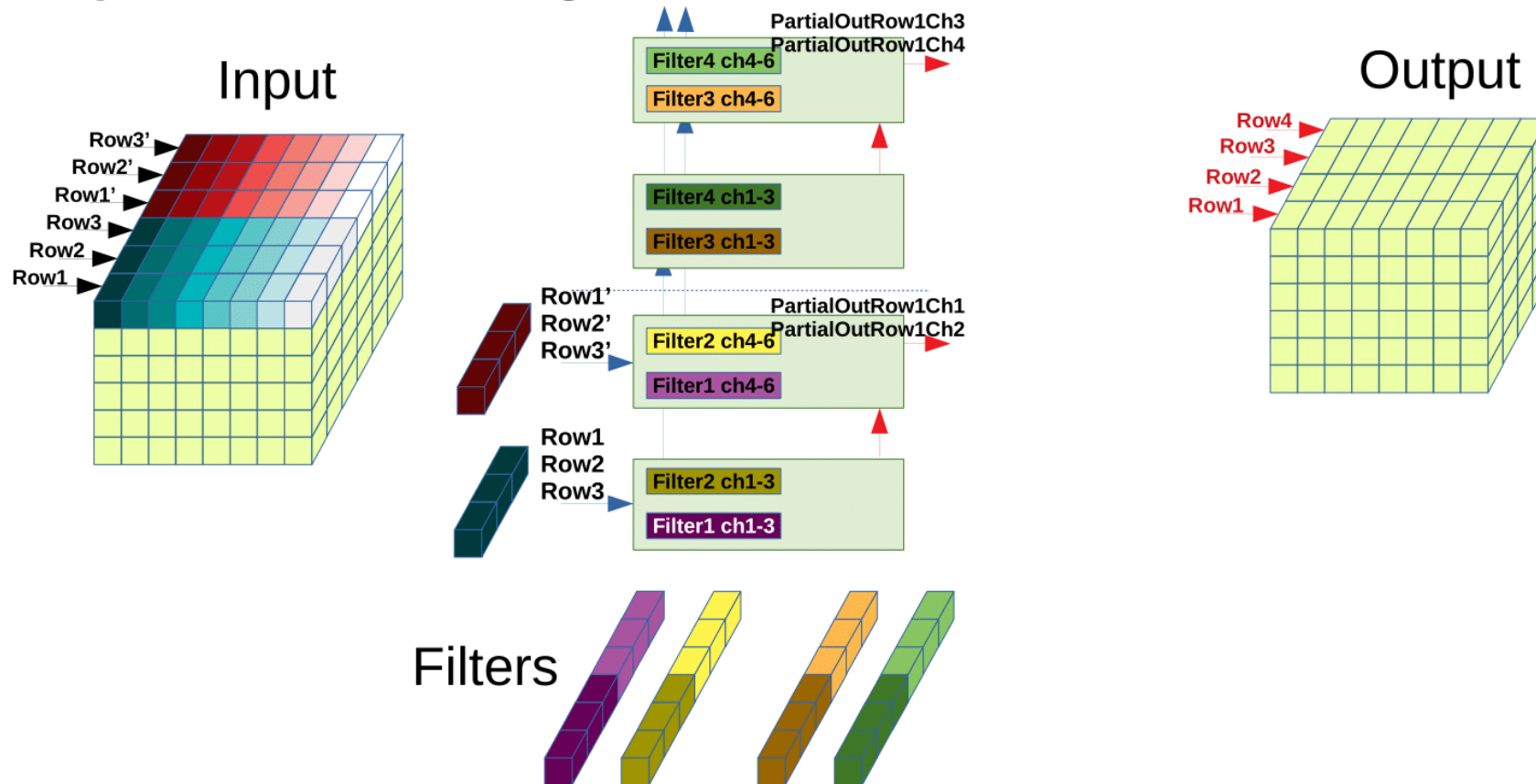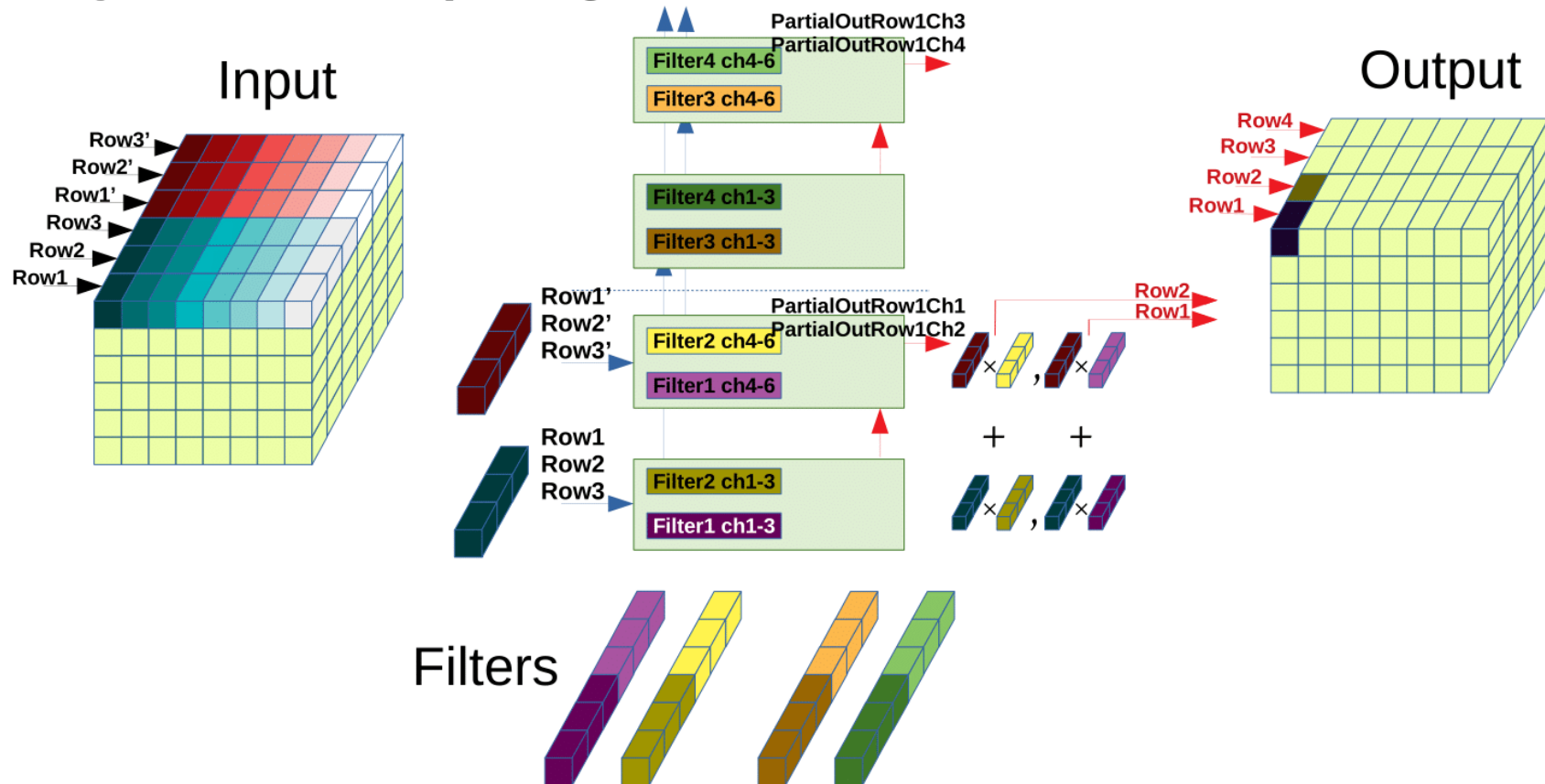
3- Point-wise

3- Point-wise

3- **Point-wise**

3- **Point-wise**

3- Point-wise
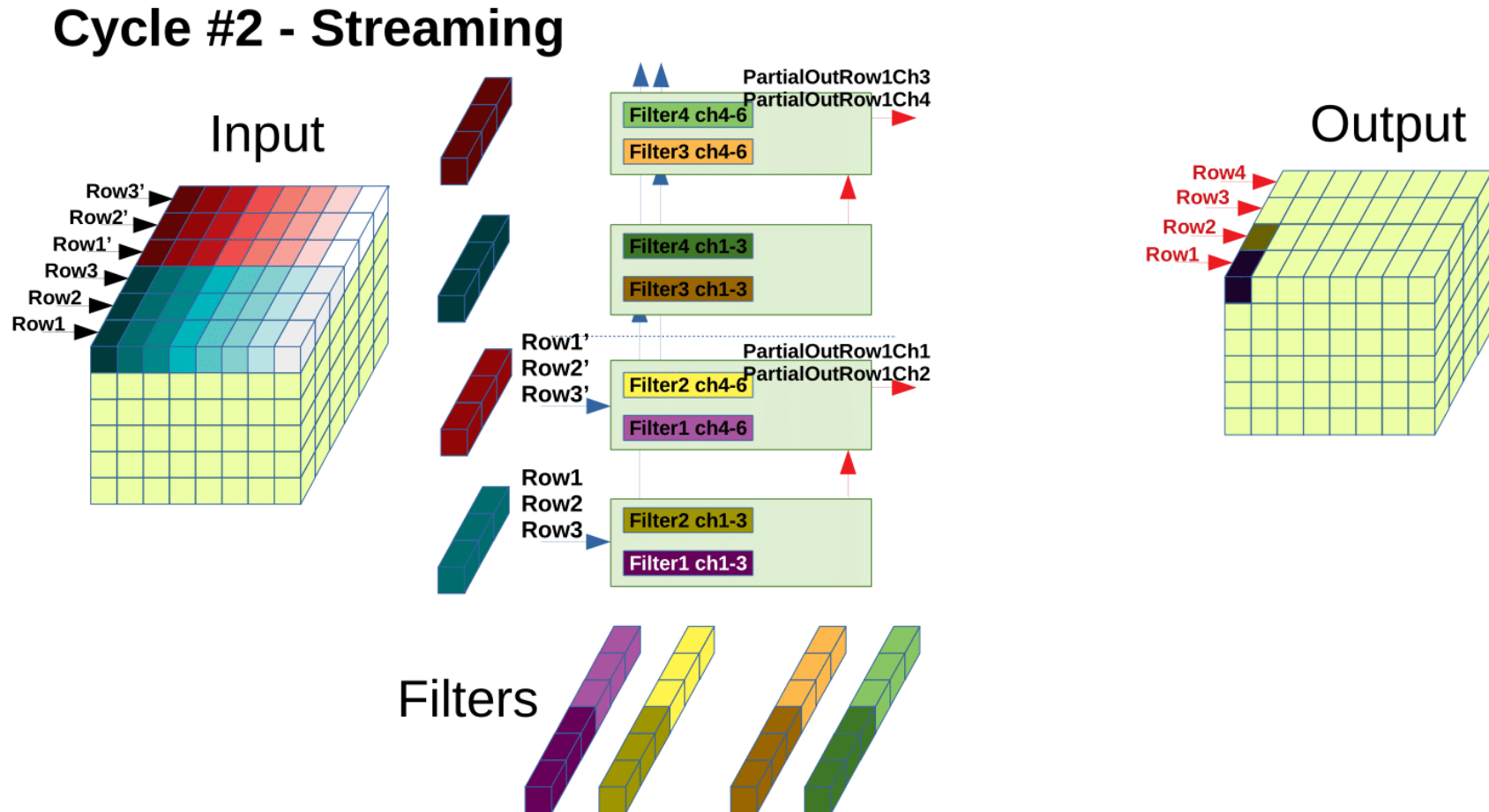
Cycle #2 - Computing
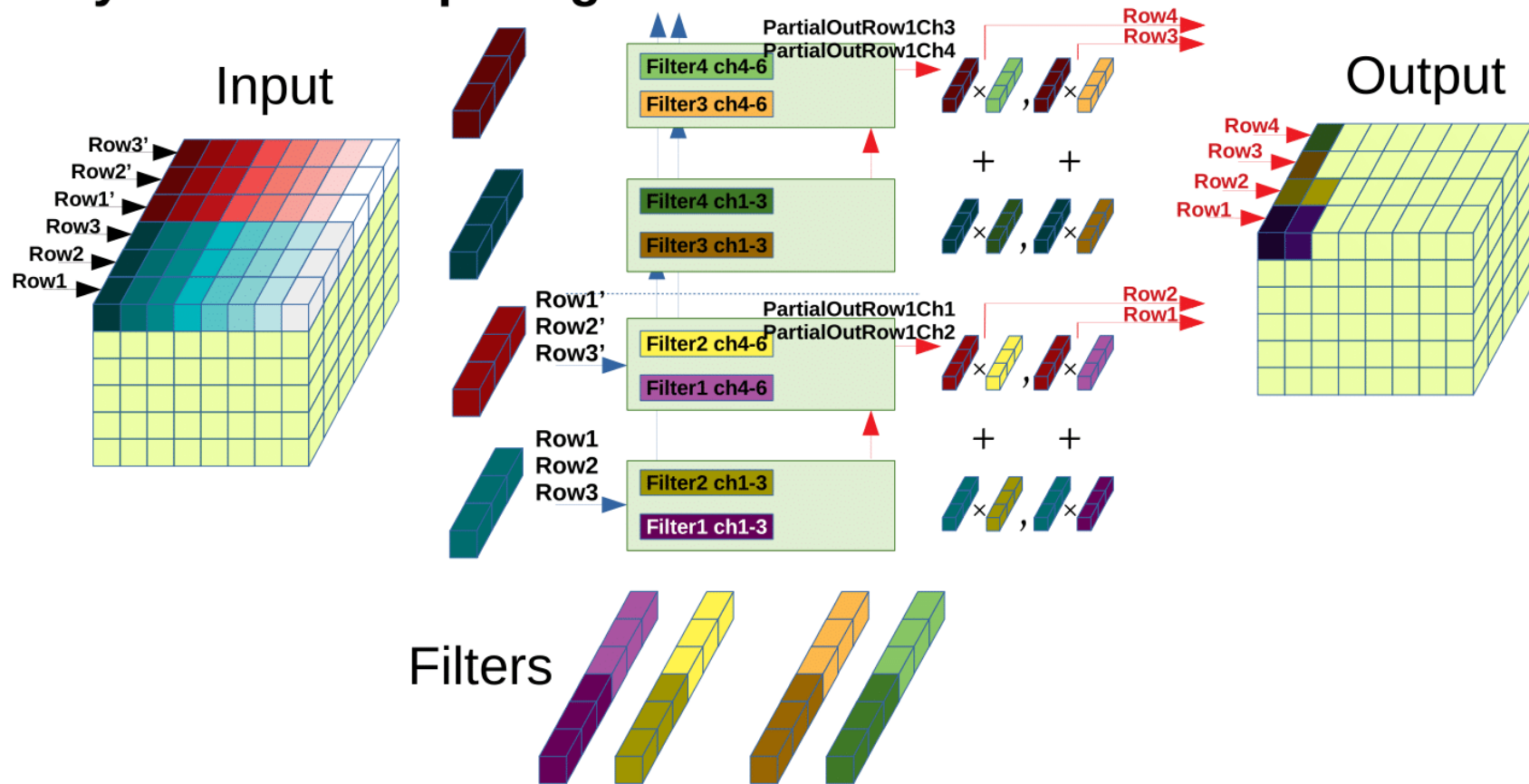
3- **Point-wise**
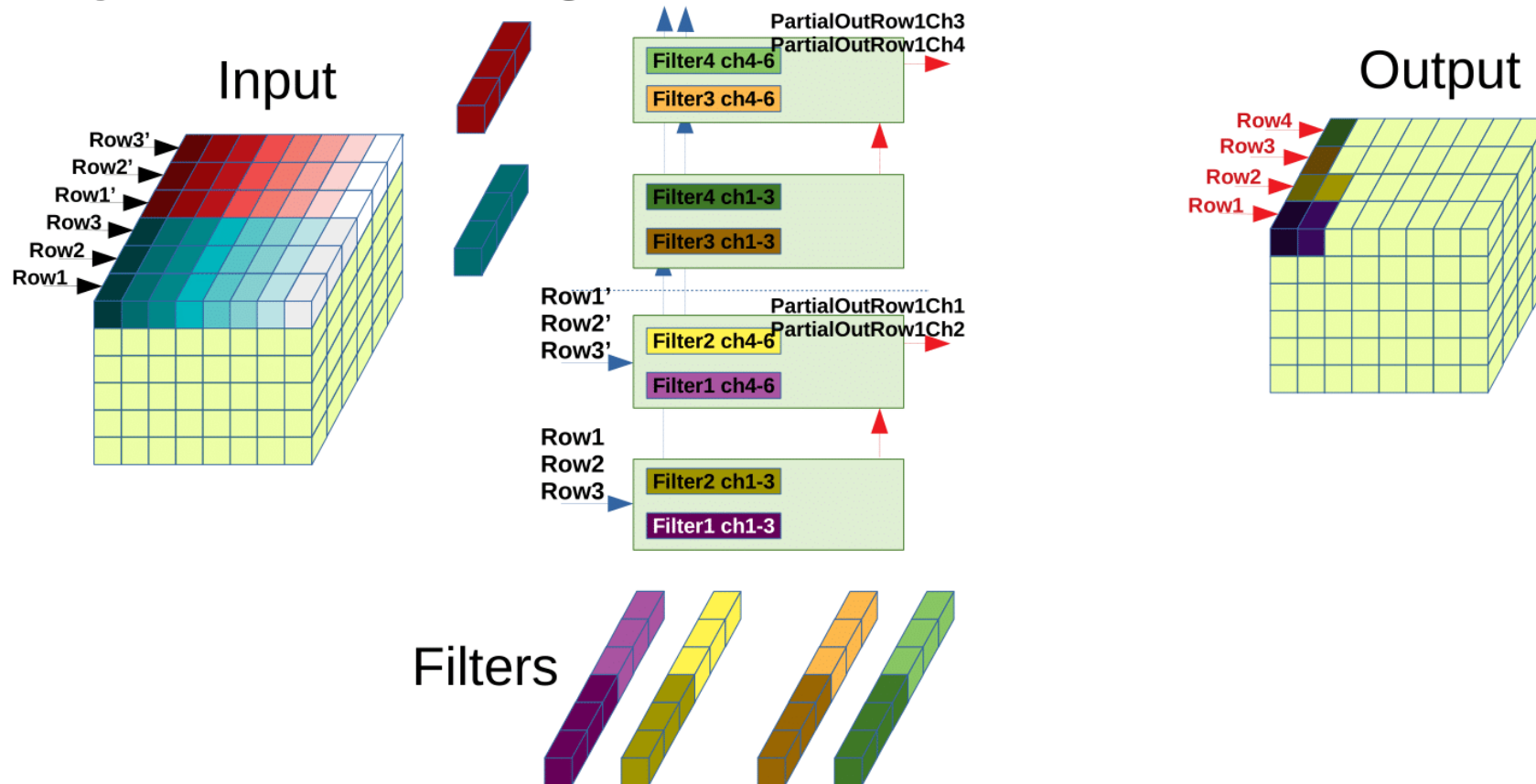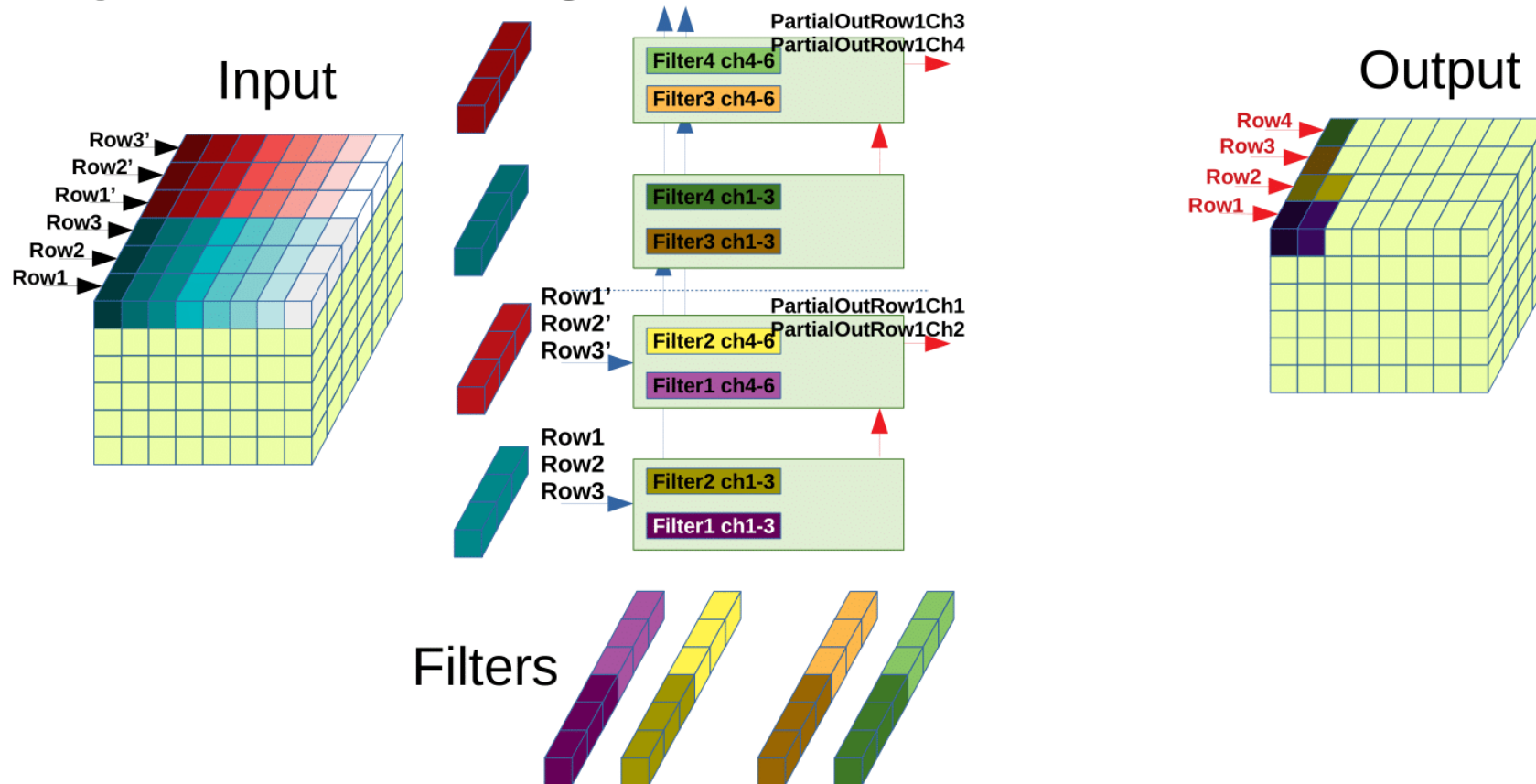
3- **Point-wise**
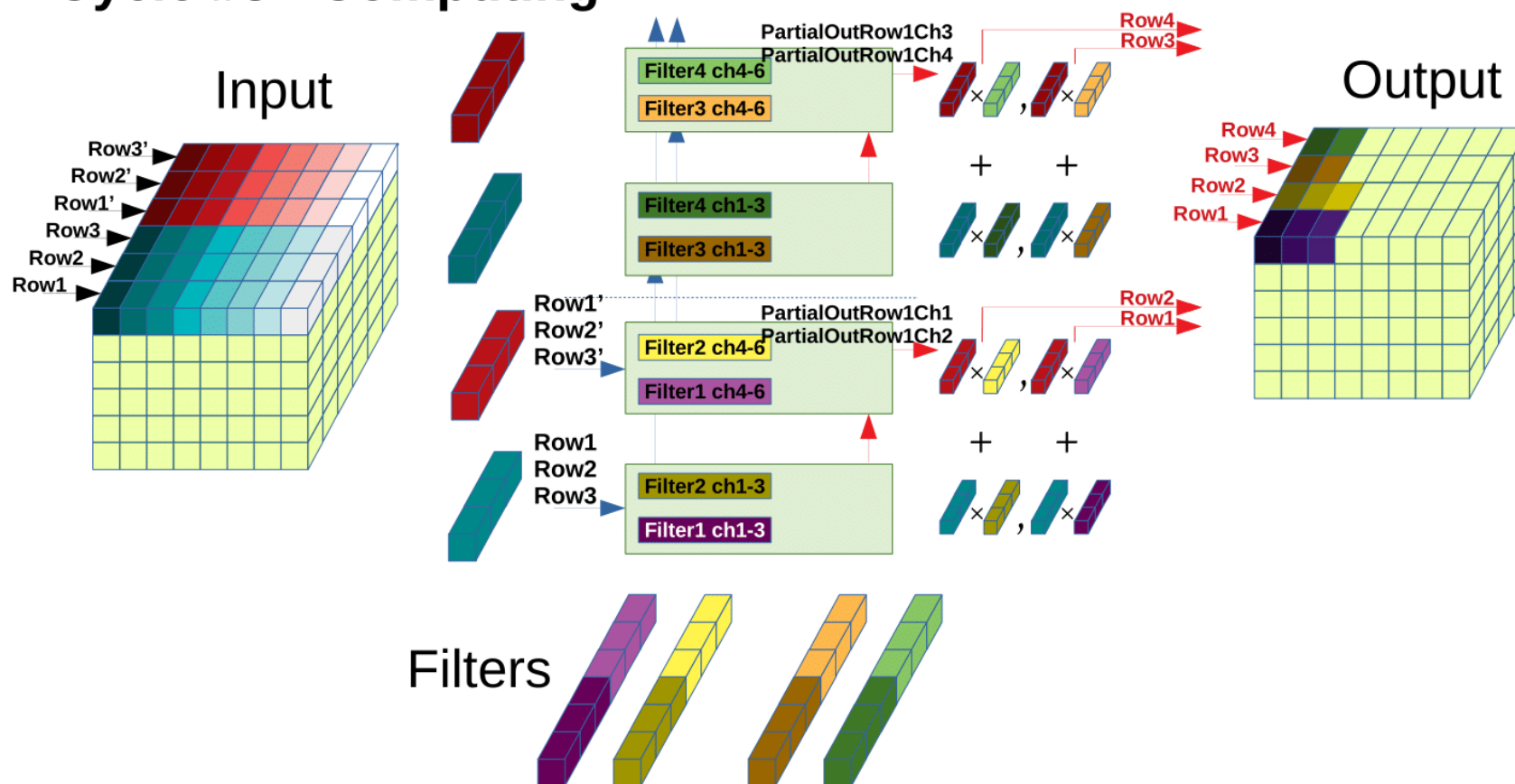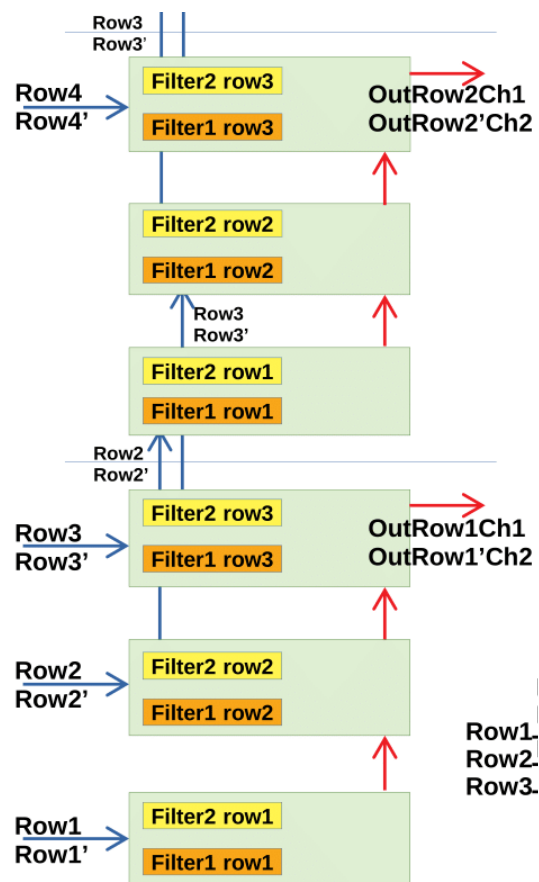
3- **Point-wise**

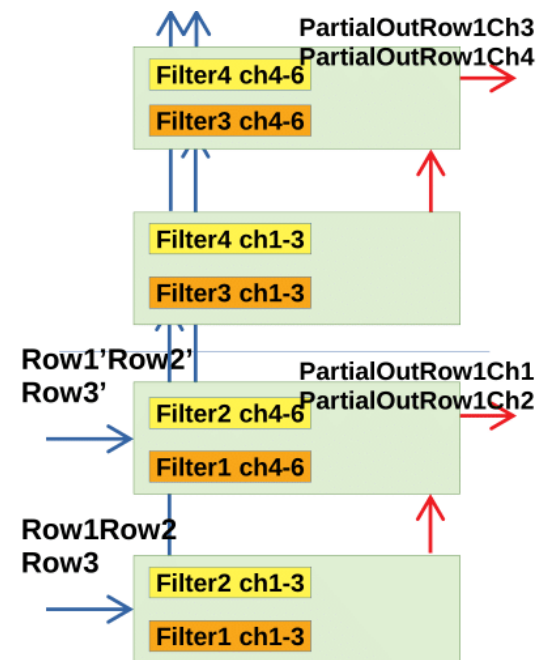Depthwise Convolution (DW)

Pointwise Convolution (PW)

› SMIC 65-nm standard cell technology

- Synopsis Design Compiler 2013.12

Version	Area Ratio	Fmax
DSP48E2	1.0	463
+ M27×18C32D2 MAC-IP	1.14	358
+ interconnect	1.18	362
+ reuse	1.28	357

DATA MOVEMENT ENERGY RATIOS IN 65 NM TECHNOLOGY ($1\times = 90$FJ).

› Other networks are simila

Energy	FF	SR_e	RF_e	Chain	RF	SR	BRAM(B)	MAC
Ratio	1	2	12.5	23	40	44	205	89-22
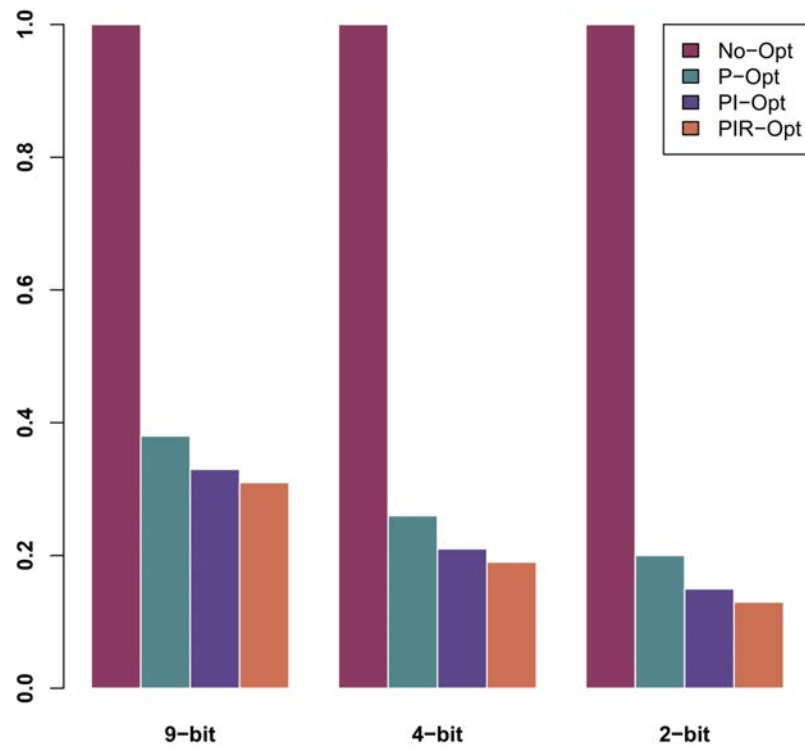
MobileNet-v2

ShuffelNet-v2

› Sits between Sharma (low-precision) and Boutros (high-precision)
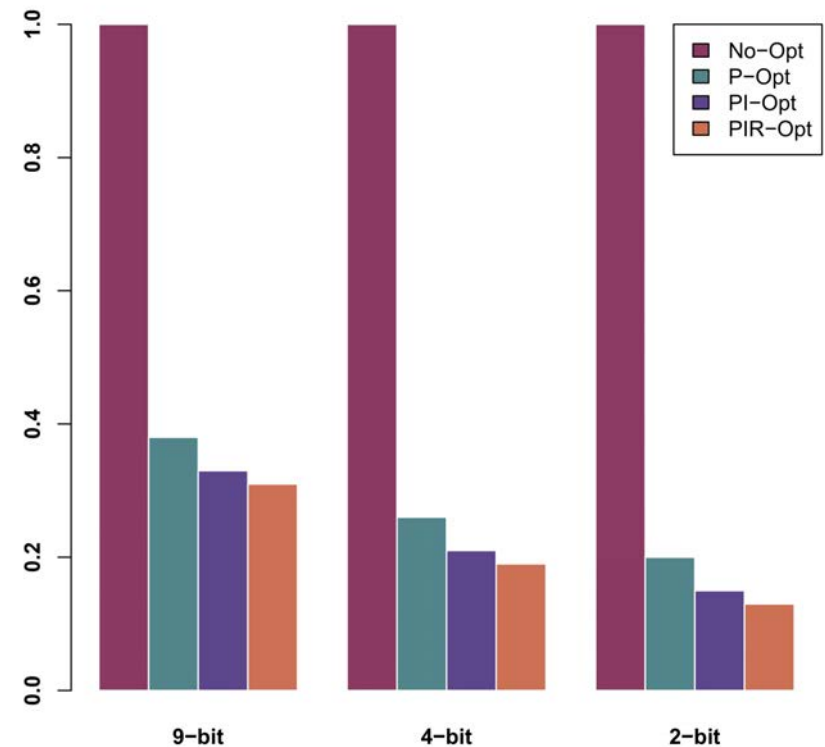
	Bitfusion [56] ISCA'18	Ours	Boutros [44] FPL'18	Ours
Area	0.24	1	0.77	1
Performance Per Area				
2x2	1	0.4		
4x4	1	0.7	1	1.2
8x8	1	1.4	1	1.2
16x16			1	0.4
27x18			1	0.8

EPIC

› **Exploration (Online kernel methods)**

› **Parallelisation**

› **Integration**

› **Customisation**

› **Described some of our efforts to develop efficient ML implementations within the EPIC framework**

› **Exploration**

› Kernel methods optimised using different algorithms, mathematical techniques, computer architectures, arithmetic

› **Parallelism**

› Increase parallelism by reducing precision

› Keep weights on-chip to devote more hardware to arithmetic

› **Integration**

› In radio frequency, this allows latency to be reduced by 4 orders of magnitude

› **Customisation**

› Supplement conventional FPGA with different DSP to support DNN implementation
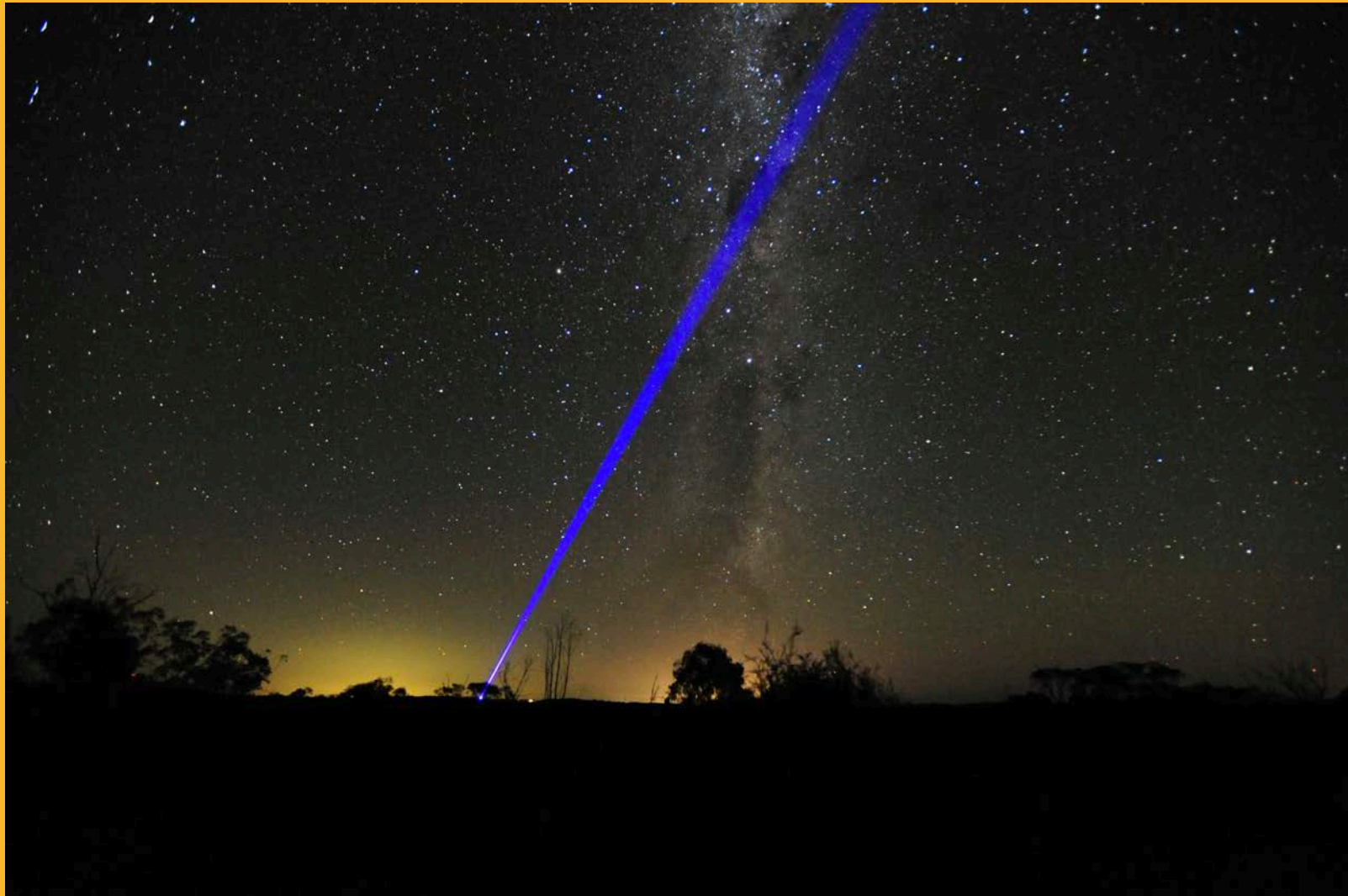
Thank you!

Philip Leong (philip.leong@sydney.edu.au)
http://phwl.org

› Sean Fox, David Boland, and Philip Leong. FPGA Fastfood - a high speed systolic implementation of a large scale online kernel method. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, pages 279–284, New York, NY, USA, 2018. ACM. (doi:10.1145/3174243.3174271)

› Julian Faraone, Nicholas Fraser, Michaela Blott, and Philip H.W. Leong. SYQ: Learning symmetric quantization for efficient deep neural networks. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, June 2018. (doi:10.1109/CVPR.2018.00452)

› Siddhartha, Yee Hui Lee, Duncan J.M. Moss, Julian Faraone, Perry Blackmore, Daniel Salmond, David Boland, and Philip H.W. Leong. Long short-term memory for radio frequency spectral prediction and its real-time FPGA implementation. In *Proc. MILCOM*, October 2018.

› Lingli Wang SeyedRamin Rasoulinezhad, Hao Zhou and Philip H.W. Leong. PIR-DSP: An FPGA DSP block architecture for multi-precision deep neural networks. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–8, 2019.