Fixed-point FPGA implementations of Machine Learning Algorithms

Philip H.W. Leong, Computer Engineering Laboratory
School of Electrical and Information Engineering

THE UNIVERSITY OF SYDNEY

Computer Engineering Laboratory

- Mission: to discover new ways of exploiting parallelism and customisation to solve computationally demanding problems.
  - Study how to achieve improved latency, throughput, energy and area efficiency using field programmable gate array (FPGA), VLSI and cluster computing technology.
- Research
  - FPGA-based computing
  - Machine learning
  - Signal processing
  - Embedded systems

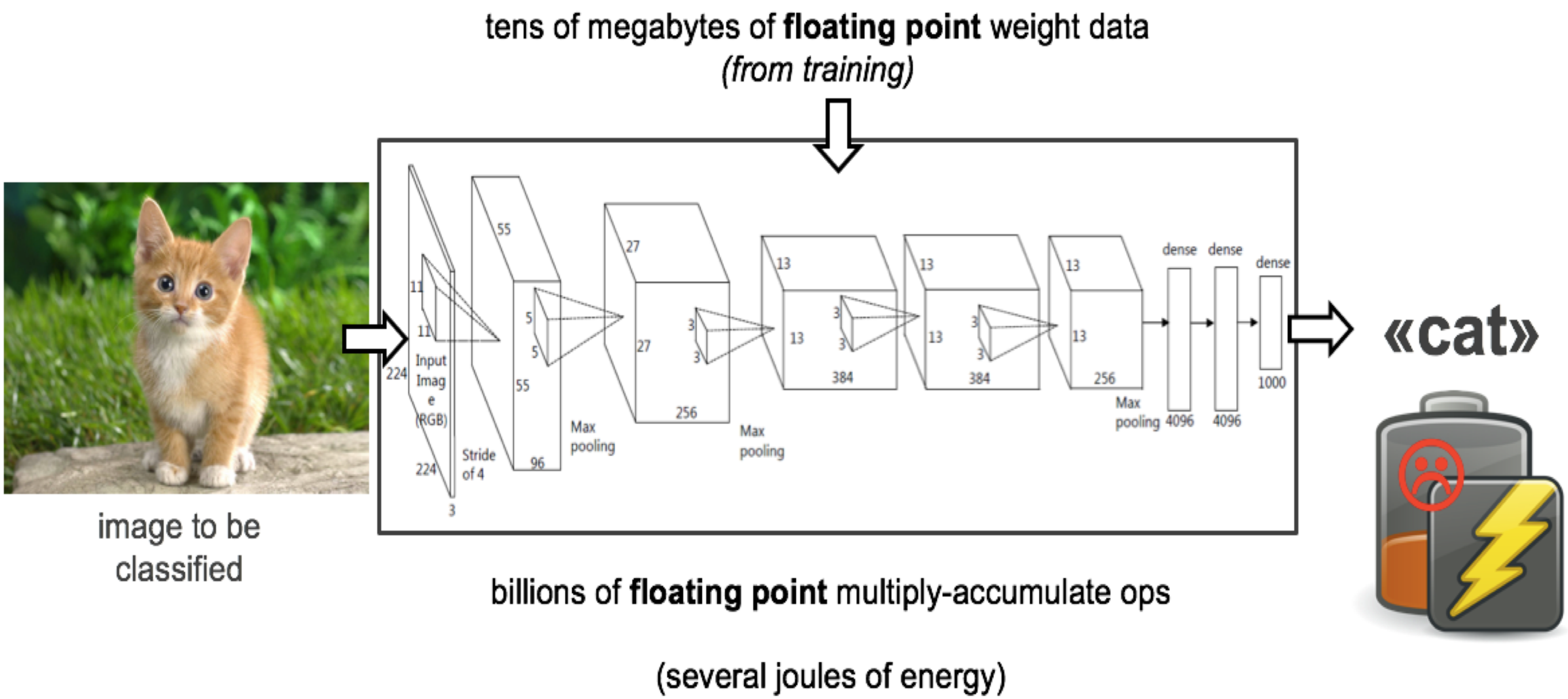
Overview

SYQ technique
Two-speed multiplier
LSTM-based spectral predictor

THE UNIVERSITY OF SYDNEY

Inference with Convolutional Neural Networks

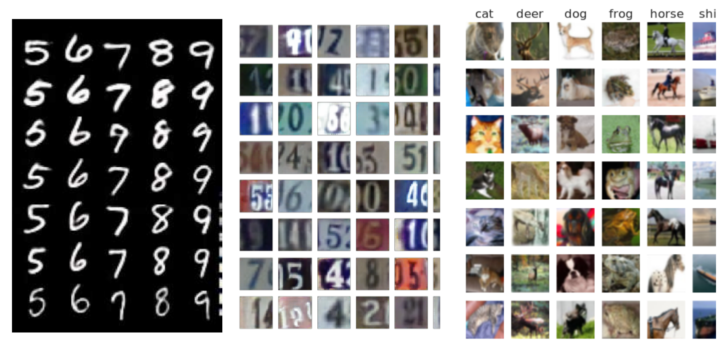
Slides from Yaman Umuroglu et. al., “FINN: A framework for fast, scalable binarized neural network inference,” FPGA’17



Binarized Neural Networks

› 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)

Advantages of BNNs

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**

Comparison

		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
FINN	MNIST, LFC-max	98.4%	1.5 M	8.8 W	22.6 W	177	269	1
FINN	CIFAR-10, CNV-max	80.1%	21.9 k	3.6 W	11.7 W	6	2	1
FINN	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
Prior Work	CIFAR-10, TrueNorth	83.4%	1.2 k	0.2 W	-	6	-	1
Prior Work	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

Issues with Low-Precision

- Who would be willing to incur a loss in accuracy?
- Can we get better accuracy with a little more hardware?

SYQ Quantisation

- To compute quantised weights from FP weights

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

with,

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

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

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

SYQ Quantisation

- 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 $\boldsymbol{\alpha_l}$ is defined by the vector $\boldsymbol{\alpha_l} = [\alpha_l^1, ..., \alpha_l^m]$:

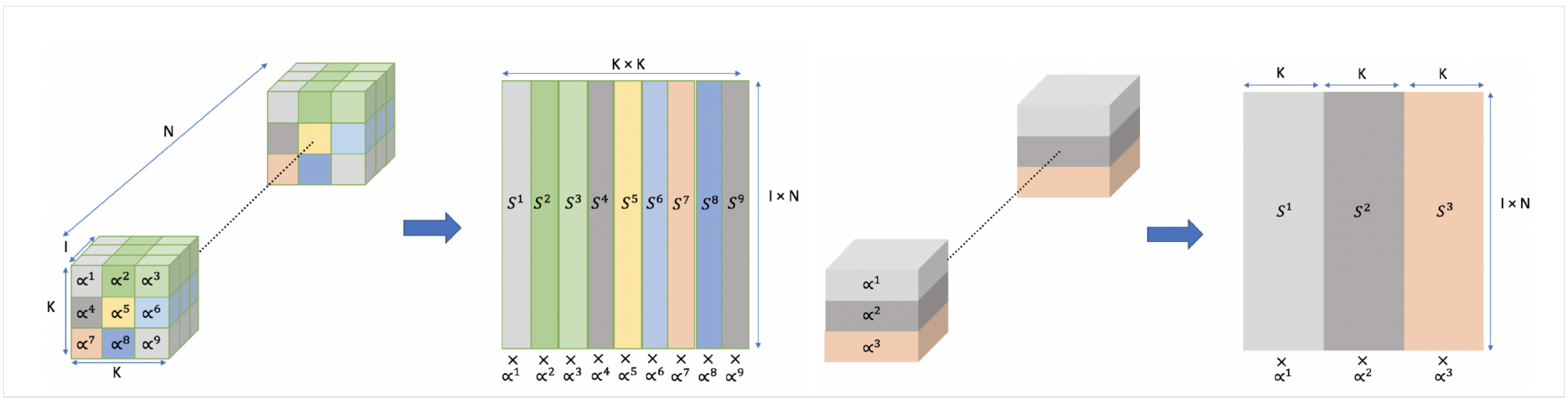
$$\boldsymbol{\alpha} = diag(\boldsymbol{\alpha}) := \begin{bmatrix} \alpha^1 & 0 & .. & 0 & 0 \\ 0 & \alpha^2 & .. & : & 0 \\ : & : & .. & \alpha^{m-1} & : \\ 0 & 0 & .. & 0 & \alpha^m \end{bmatrix}$$

- Train by solving

$$\alpha_l^* = \underset{\alpha}{\operatorname{argmin}}\, E(\alpha, \boldsymbol{Q}) \quad s.t. \quad \alpha \geq 0,\ Q_{l_{i,j}} \in \mathbb{C}$$

Subgrouping

› More fine-grained quantisation can improve approximation of weights



Pixel-wise scaling   Row-wise scaling (layer-wise also option)

Training

- Straight through approximator used to address vanishing gradients problem

Algorithm 1 SYQ Training Summary For DNNs.

Initialize: Set subgrouping granularity for S_l^i and set $\alpha_{l_0}^i$.
Inputs: Minibatch of inputs & targets (I, Y), Error function $E(Y, \hat{Y})$, current weights $\boldsymbol{W_t}$ and learning rate, γ_t
Outputs: Updated $\boldsymbol{W_{t+1}}$, $\boldsymbol{\alpha_{t+1}}$ and γ_{t+1}

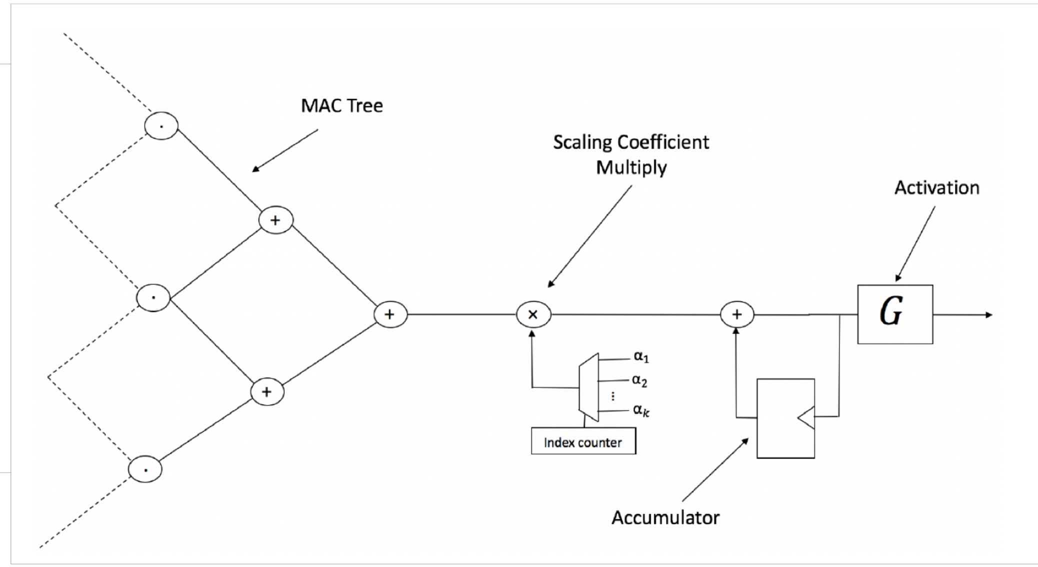
SYQ Forward:
for l=1 to L **do**
  $\boldsymbol{Q}_l = sign(\boldsymbol{W}_l) \odot \boldsymbol{M}_l$ **with** η, using (3) & (4)
  for ith subgroup in lth layer **do**
    Apply α_l^i to S_l^i
  end for
end for
$\hat{Y}$ = **SYQForward** $(I, Y, \boldsymbol{Q}_l, \boldsymbol{\alpha}_l)$ using (14)

SYQ Backward:
$\frac{\partial \hat{E}}{\partial \boldsymbol{Q}_l}$ = **WeightBackward**$(\boldsymbol{Q}_l, \boldsymbol{\alpha}_l, \frac{\partial \hat{E}}{\partial \hat{Y}})$ using (12) & (15)
$\frac{\partial \hat{E}}{\partial \boldsymbol{\alpha}_l}$ = **ScalarBackward**$(\frac{\partial \hat{E}}{\partial \boldsymbol{Q}_l}, \boldsymbol{\alpha}_l, \frac{\partial \hat{E}}{\partial \hat{Y}})$ using (11)
$\boldsymbol{W_{t+1}}$ = **UpdateWeights**$(\boldsymbol{W_t}, \frac{\partial \hat{E}}{\partial \boldsymbol{Q}_l}, \gamma)$
$\boldsymbol{\alpha_{t+1}}$ = **UpdateScalars**$(\boldsymbol{\alpha_t}, \frac{\partial \hat{E}}{\partial \boldsymbol{\alpha}_l}, \gamma)$
γ_{t+1} = **UpdateLearningRate**(γ_t, t)

Resource Utilisation

- For K filters, I Input feature maps of dimension FxF, N output feature maps
- $P=K^2INF^2$

Method	Scalars	Ops
Layer (DoReFa)	1	P
Row (SYQ)	K	P
Pixel (SYQ)	K^2	P
Asymmetric (TTQ)	2	$P+Z$
Grouping (FGQ)	$K^2N/4$	P
Channel (HWGQ/BWN)	N	P



Results

› Full precision for 1st and last layers, CONV layers pixel-wise, FC layer-wise

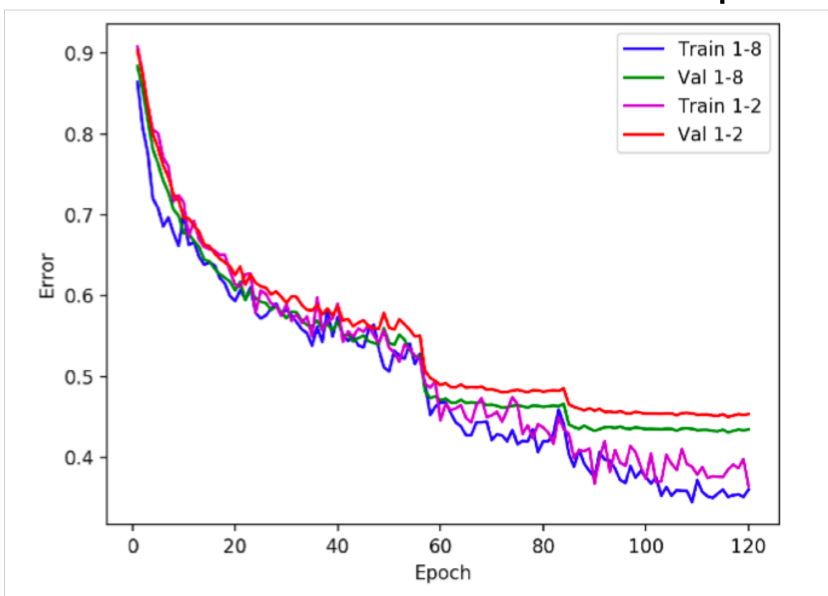
Model		1-8	2-8	Baseline	Reference
AlexNet	Top-1	**56.6**	**58.1**	56.6	57.1
	Top-5	**79.4**	**80.8**	80.2	80.2
VGG	Top-1	**66.2**	**68.7**	69.4	-
	Top-5	**87.0**	**88.5**	89.1	-
ResNet-18	Top-1	**62.9**	**67.7**	69.1	69.6
	Top-5	**84.6**	**87.8**	89.0	89.2
ResNet-34	Top-1	**67.0**	**70.8**	71.3	73.3
	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)

Low-precision Activation and Different Subgrouping

Alexnet example

- Lowering activation precision does not severely alter the training curve
  - Suggests gradient information from pixel-wise scaling compensates for information loss
- Accuracy difference between default pixel-wise and row/layer symmetric quantisation
  - Not much difference between pixel/row-wise except for binary case



		Row-wise		Layer-wise	
Weights	**Act.**	**Top-1**	**Top-5**	**Top-1**	**Top-5**
1	2	-0.7	-0.5	-1.4	-2.2
1	8	-0.1	-0.3	-0.4	-2.2
2	2	+0.1	-0.0	-1.3	-1.5
2	8	-0.1	-0.1	-1.9	-1.7

Comparison with Previous Work

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

Overview

SYQ technique
Two-speed multiplier
LSTM-based spectral predictor

THE UNIVERSITY OF SYDNEY

Introduction

- Multiplication arguably most important computational primitive
- High radix Modified Booth Algorithm with Wallace or Dadda trees generally accepted as the highest performing implementation
- Present technique which introduces a dynamic control structure to remove parts of the computation completely during runtime

Radix 4 Signed Multiplication

- x and y are multiplicand and multiplier
- for n-bit multiplication, radix r, where X_i, Y_i are the digits
- Can be expressed recursively as

$$
\begin{aligned}
p[0] &= 2^{n-2}(Y_1 + Y_0)x \\
p[j+1] &= 2^{-2}(p[j] + 2^n(Y_{2j+1} + Y_{2j} - 2Y_{2j-1})x) \\
&\quad j = 1, \ldots, N-1 \\
p &= p[N]
\end{aligned}
$$

Radix 4 Booth Serial Multiplier

$$
\begin{aligned}
p[0] &= 2^{n-2}(Y_1 + Y_0)x \\
p[j+1] &= 2^{-2}(p[j] + 2^n(Y_{2j+1} + Y_{2j} - 2Y_{2j-1})x) \\
&\quad j = 1, \ldots, N-1 \\
p &= p[N]
\end{aligned}
$$

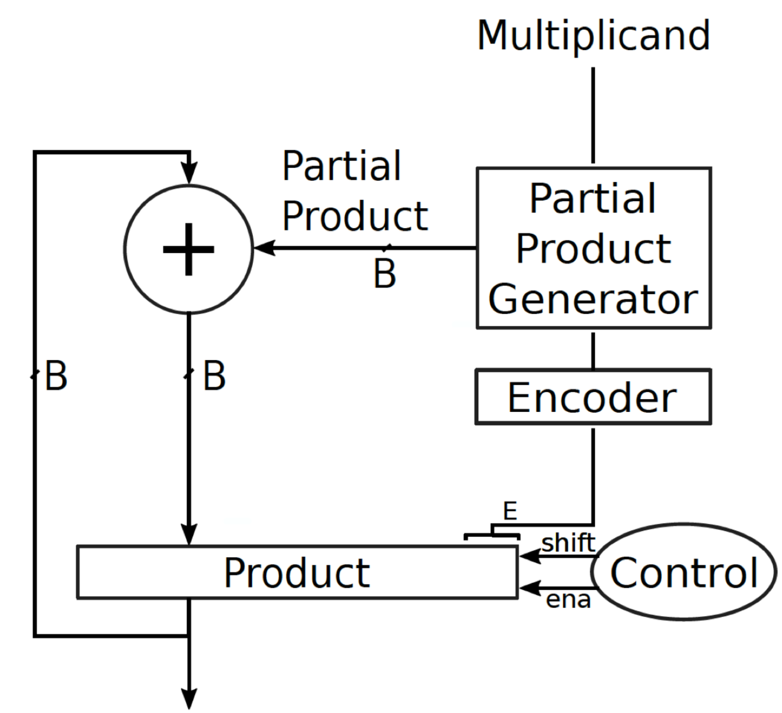


TABLE I: Booth Encoding

Y_{i+2}	Y_{i+1}	Y_i	e_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	$\bar{2}$
1	0	1	$\bar{1}$
1	1	0	$\bar{1}$
1	1	1	0

$\bar{2}$ and $\bar{1}$ represent -2 and -1 respectively.

Radix 4 Booth Serial Multiplier

Key idea: don't need to add for Partial Product = 0 case

$$
\begin{aligned}
p[0] &= 2^{n-2}(Y_1 + Y_0)x \\
p[j+1] &= 2^{-2}(p[j] + 2^n(Y_{2j+1} + Y_{2j} - 2Y_{2j-1})x) \\
& \qquad j = 1, \ldots, N-1 \\
p &= p[N]
\end{aligned}
$$

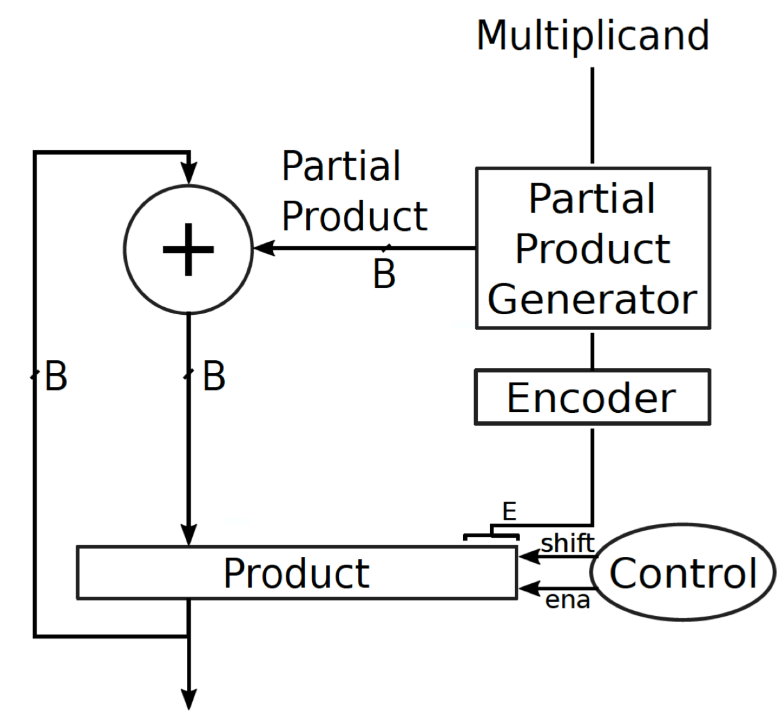


TABLE I: Booth Encoding

Y_{i+2}	Y_{i+1}	Y_i	e_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	$\bar{2}$
1	0	1	$\bar{1}$
1	1	0	$\bar{1}$
1	1	1	0

$\bar{2}$ and $\bar{1}$ represent -2 and -1 respectively.

2-Speed Multiplier Algorithm

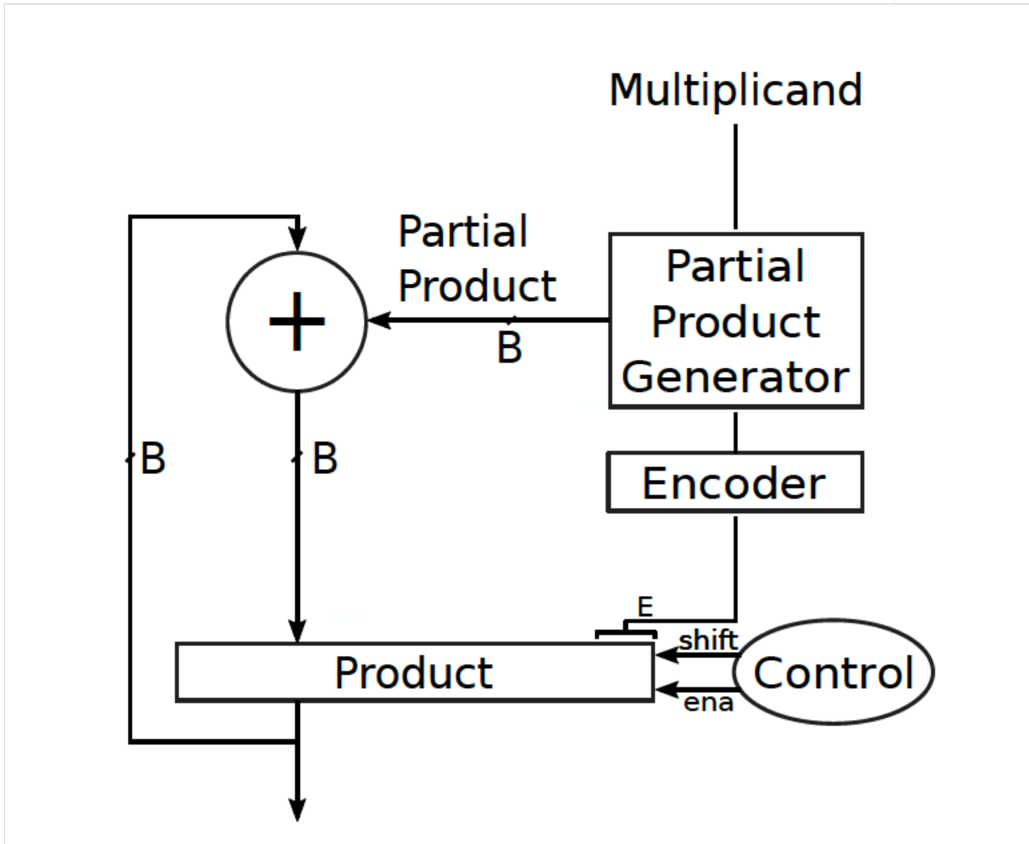
Data: y: Multiplier, x: Multiplicand
Result: p: Product
$p = y$;
$e = (P[0] - 2P[1])$;
for $count = 1$ **to** N **do**
    $PartialProduct = e * x$;
    $p =$ sra(p,2);
    $P[2 * B - 1 : B] += PartialProduct$;
    $e = (P[1] + P[0] - 2P[2])$;
end

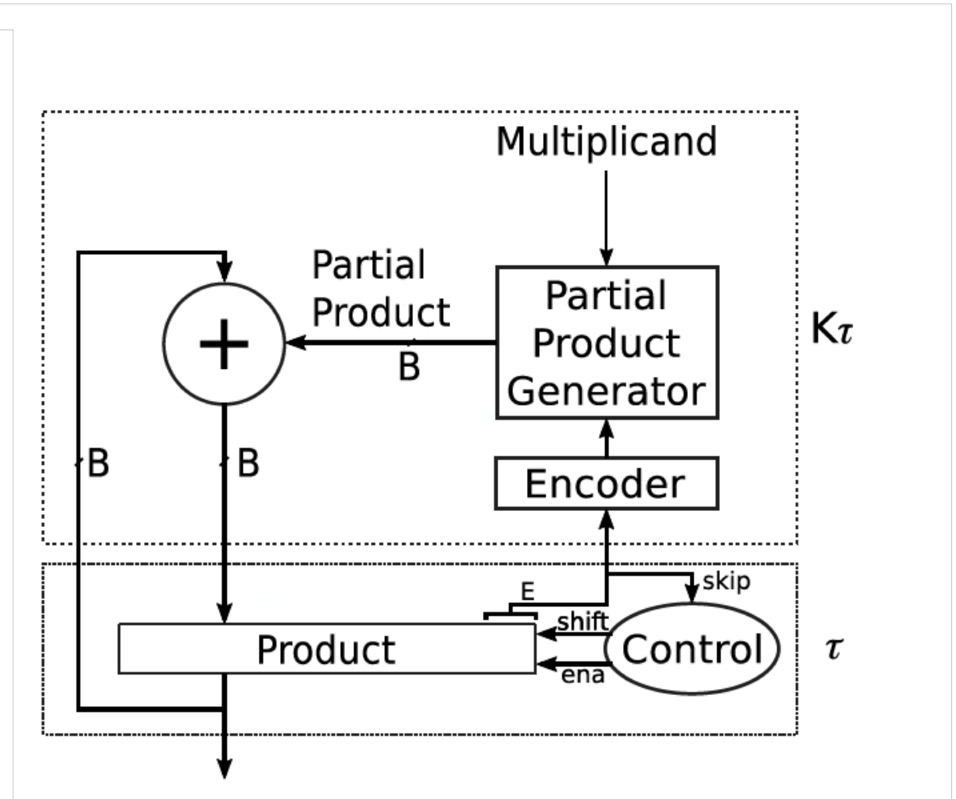
Data: y: Multiplier, x: Multiplicand
Result: p: Product
$p = y$;
$e = (P[0] - 2P[1])$;
for $count = 1$ **to** N **do**
    $p =$ sra(p,2);
    `// If non-zero encoding, take the` $K\tau$ `path, otherwise the` τ `path`
    if $e \neq 0$ **then**
        `// this path is clocked` $\bar{K}$ `times`
        $PartialProduct = e * x$;
        $P[2 * B - 1 : B] += PartialProduct$;
    end
    $e = (P[1] + P[0] - 2P[2])$;
end

2-Speed Datapath

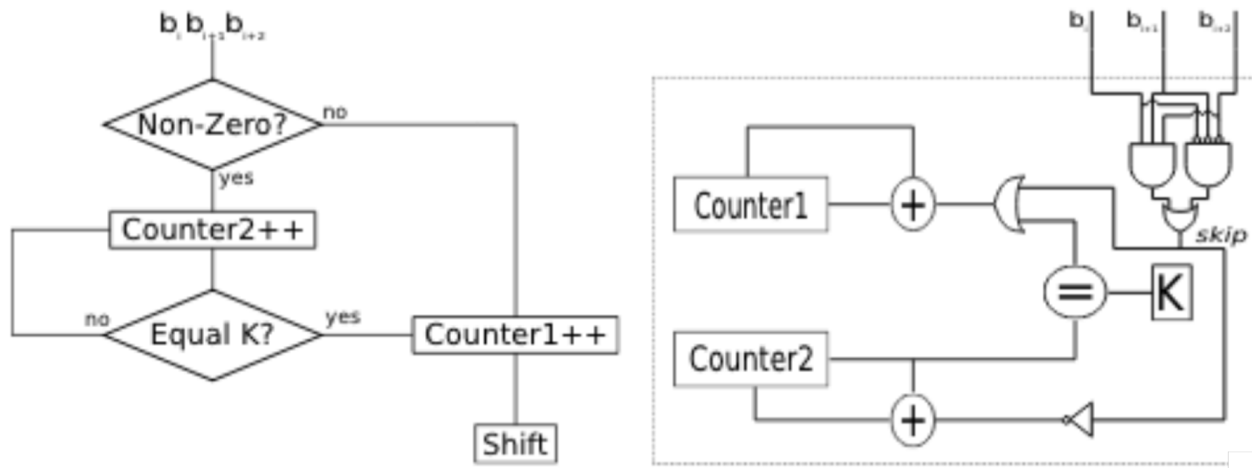
The datapath is split into 2 sections, each with its own critical path

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

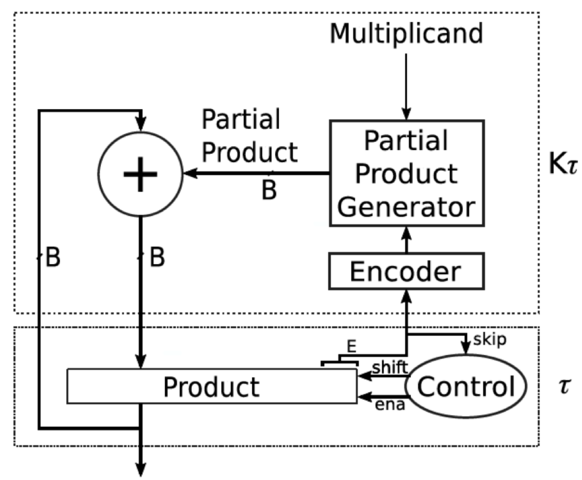




Two-Speed Control Circuit



(a) Controller flowchart (b) Control Circuit

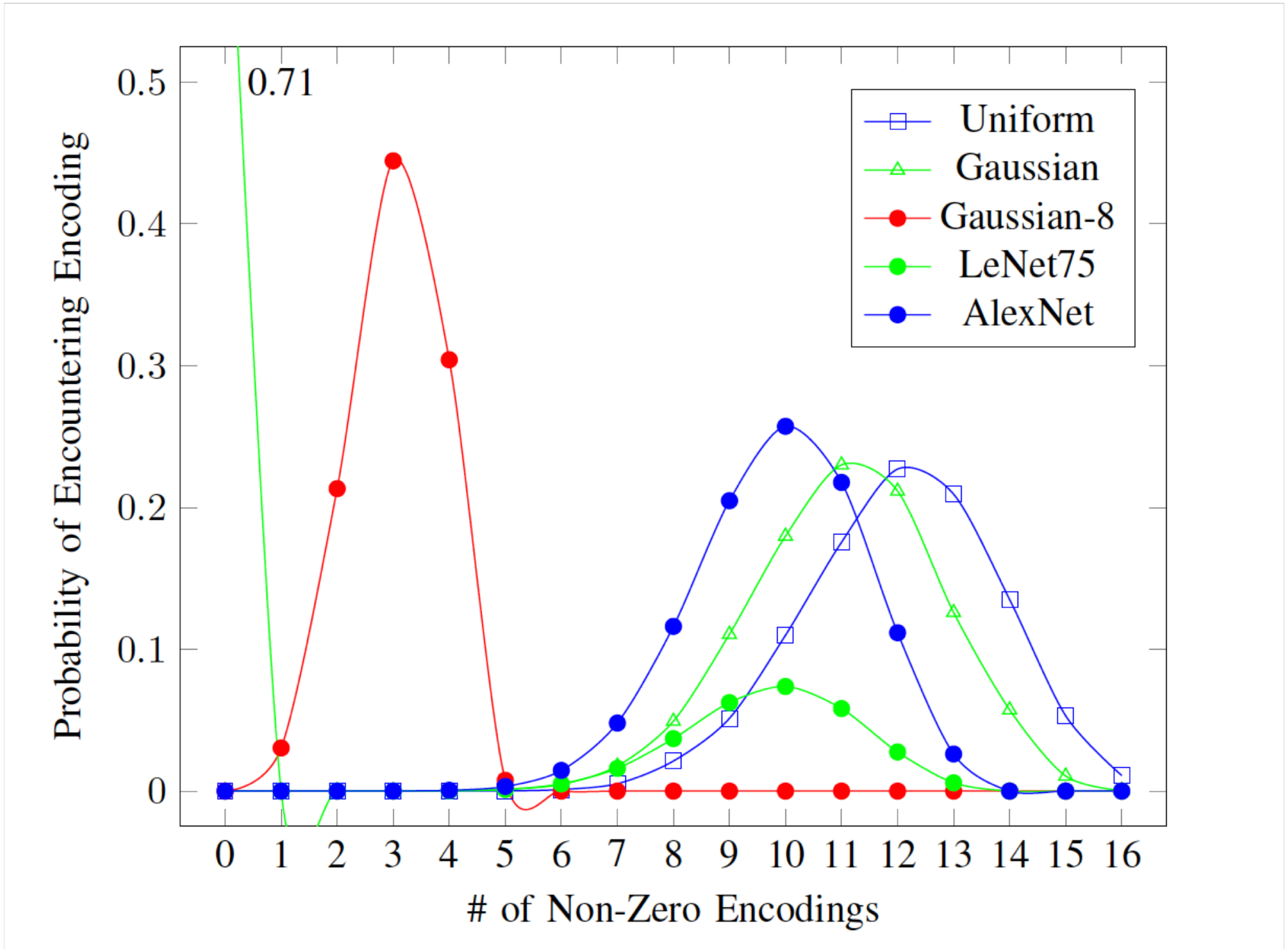


Example

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

Bit Representation	Action	Time	PartialProduct
1 1 1 1 0 1 0 0 0 1 $\boxed{0\ 0\ 0}$	skip	τ	$0x \times 2^0$
1 1 1 1 0 1 0 0 $\boxed{0\ 1\ 0}$	add	$\tau + \overline{K}\tau$	$1x \times 2^2$
1 1 1 1 0 1 $\boxed{0\ 0\ 0}$	skip	$2\tau + \overline{K}\tau$	$0x \times 2^4$
1 1 1 1 $\boxed{0\ 1\ 0}$	add	$2\tau + 2\overline{K}\tau$	$1x \times 2^6$
1 1 $\boxed{1\ 1\ 0}$	add	$2\tau + 3\overline{K}\tau$	$-1x \times 2^8$
$\boxed{1\ 1\ 1}$	skip	$3\tau + 3\overline{K}\tau$	$0x \times 2^{10}$

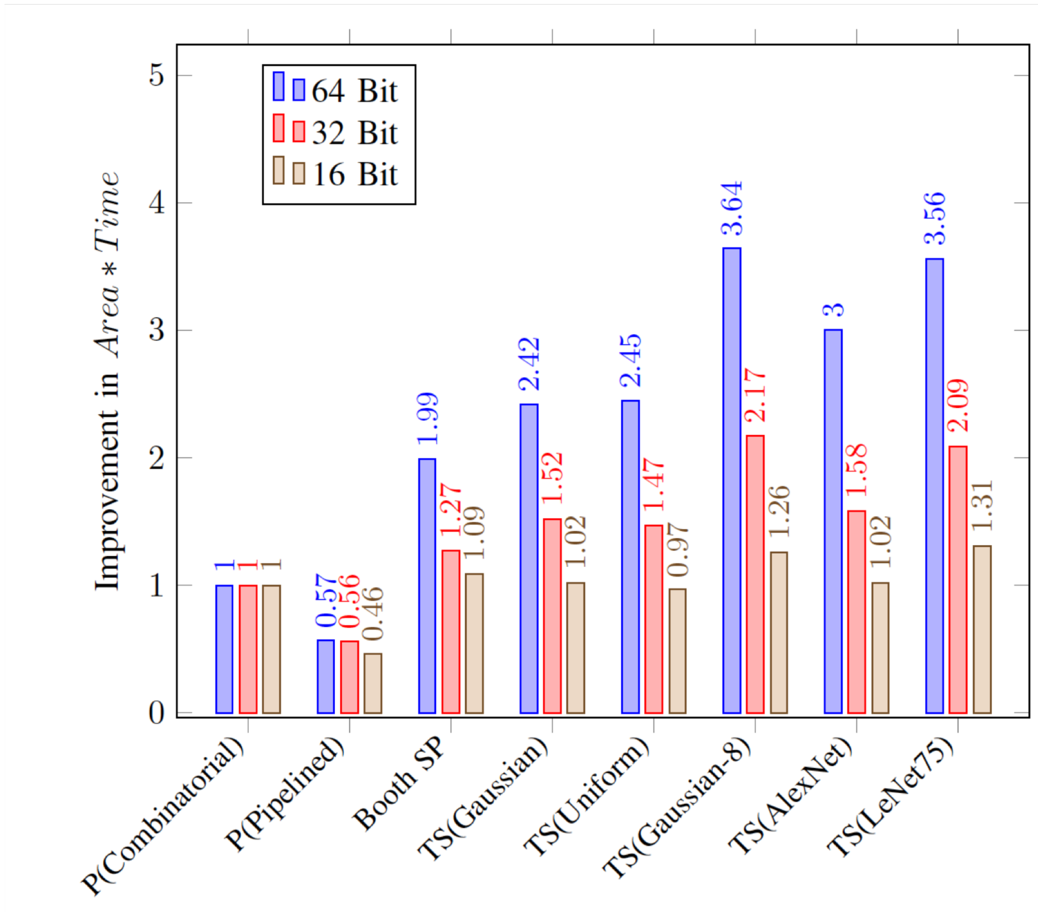
Distribution of Non-zero Encodings



Implementation

B	Type	*Area* (LEs)	Max Delay (ns)	Latency (Cycles)	*Power* (mW)
64	Parallel(Combinatorial)	5104	14.7	1	2.23
	Parallel(Pipelined)	4695	6.99	4**	9.62
	Booth Serial-Parallel	292	3.9	33	2.23
	Two Speed	304	1.83 (τ)	45.2*	5.2
32	Parallel(Combinatorial)	1255	10.2	1	1.33
	Parallel(Pipelined)	1232	4.6	4**	5.07
	Booth Serial-Parallel	156	3.8	17	1.78
	Two Speed	159	1.76 (τ)	25.6*	3.18
16	Parallel(Combinatorial)	319	6.8	1	0.94
	Parallel(Pipelined)	368	3.2	4**	3.49
	Booth Serial-Parallel	81	2.72	9	1.67
	Two Speed	87	1.52 (τ)	14*	4.35

Area-Time Performance



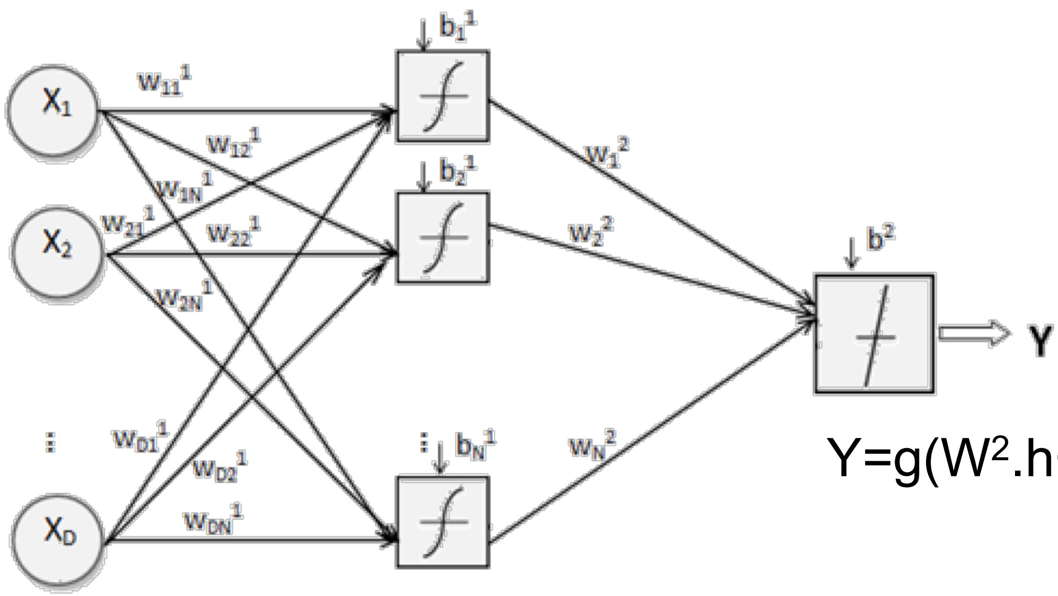
Overview

SYQ technique
Two-speed multiplier
LSTM-based spectral predictor

Motivation

- Highly dynamic and complex environments pose a challenge for current tactical/cognitive radios
- LSTMs have been extremely successful at difficult tasks such as speech recognition and machine translation
- LSTM suitability for real-time radio applications not well studied
- Can we effectively use ML in the next generation of radios?

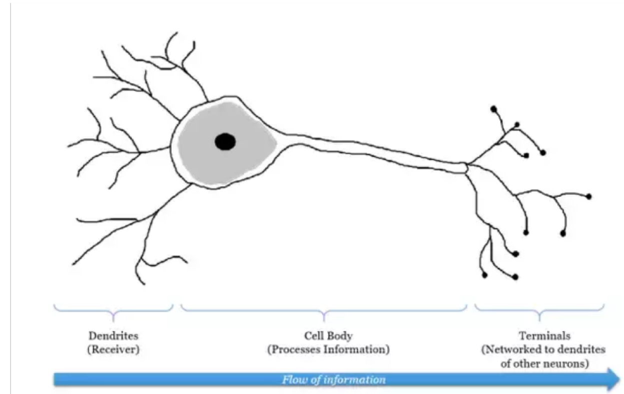
Feedforward Neural Network



(Matheus, 2016)

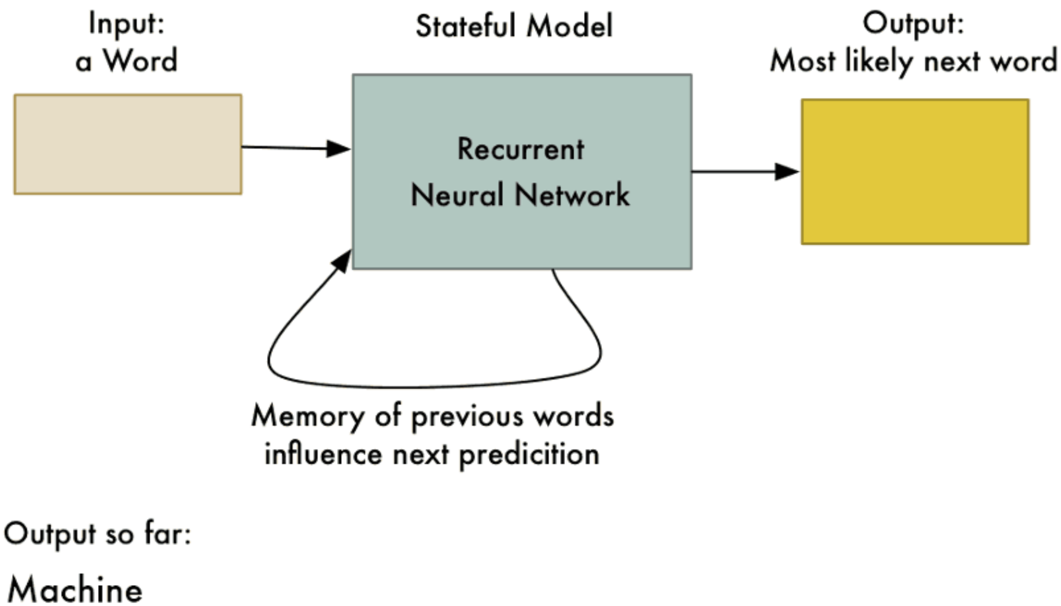
$$Y=g(W^2.h+b^2)$$

$$h=f(W^1.X+b^1)$$

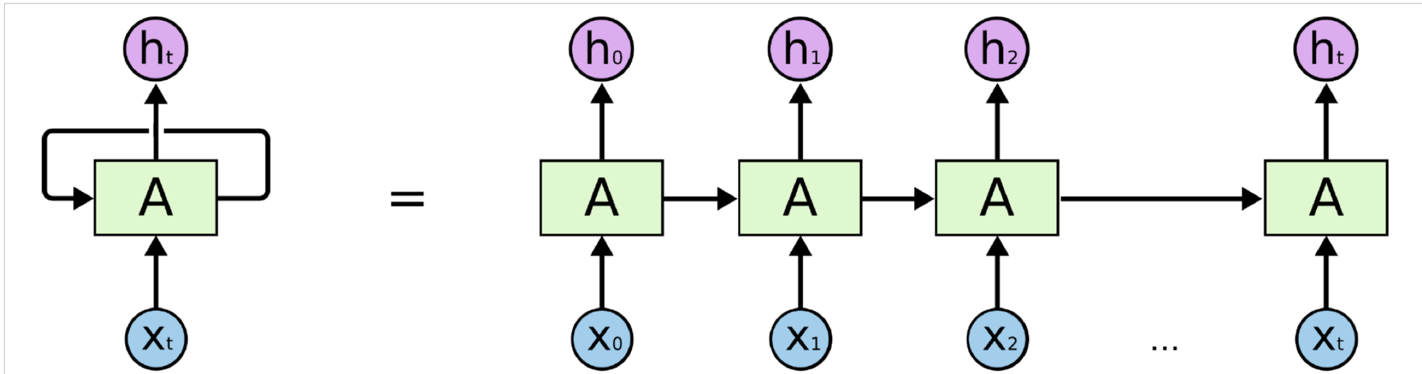


No state → Can't deal with sequential data

Recurrent Neural Networks



Can't deal with large gaps

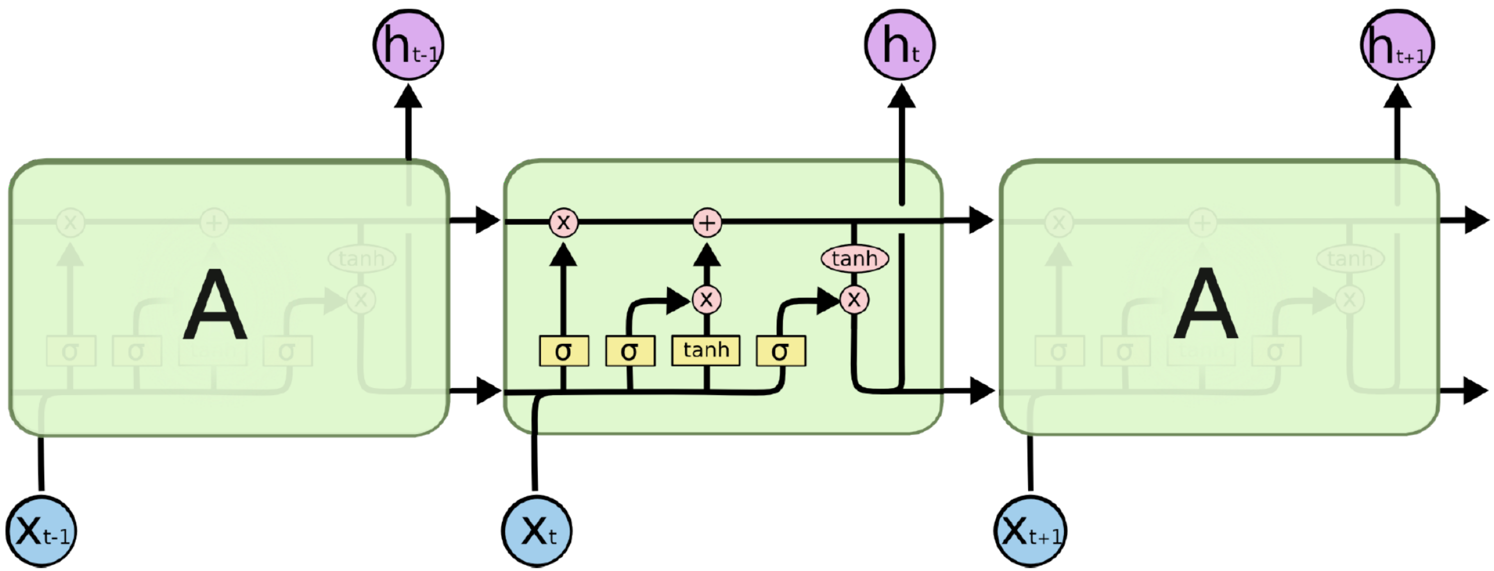


An unrolled recurrent neural network.

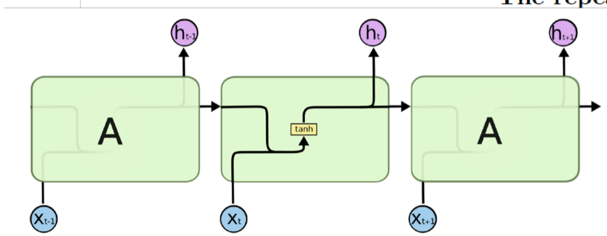
Source: colah's blog (with permission)

Long short-term Memory

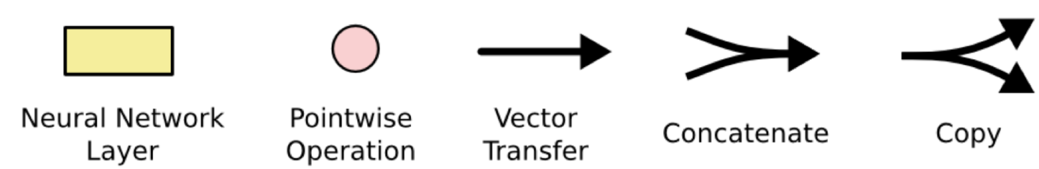
Long Short-Term Memory is a type of **gated** Recurrent Neural Network (RNN)
Proposed by Hocreiter and Schmidhuber in 1997



The repeating module in an LSTM contains four interacting layers.



The repeating module in a standard RNN contains a single layer.



Source: colah's blog (with permission)

LSTM Mathematical Description

› LSTM

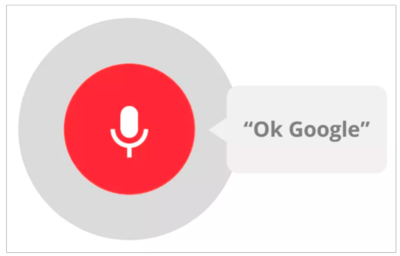
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T^{l}_{(n_{l-1}+n_l),(4n_l)} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix}$$

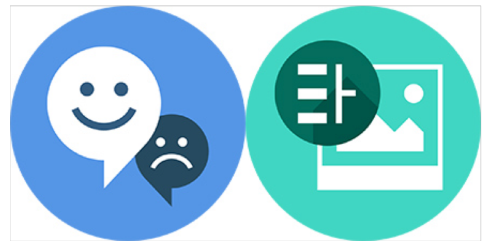
$$\begin{aligned} c_t^l &= f \odot c_{t-1}^l + i \odot g \\ h_l^t &= o \odot \tanh(c_t^l) \end{aligned}$$

› Followed by a single linear fully connected layer

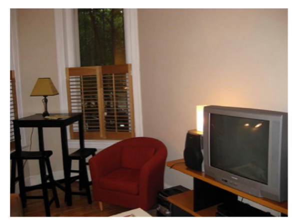
$$f_t = T^{L+1}_{n_L, n_L} h_t^L$$

Applications



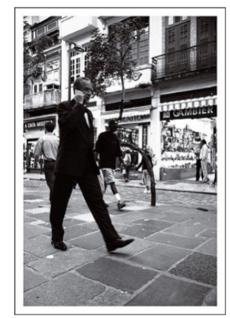


Google Translate



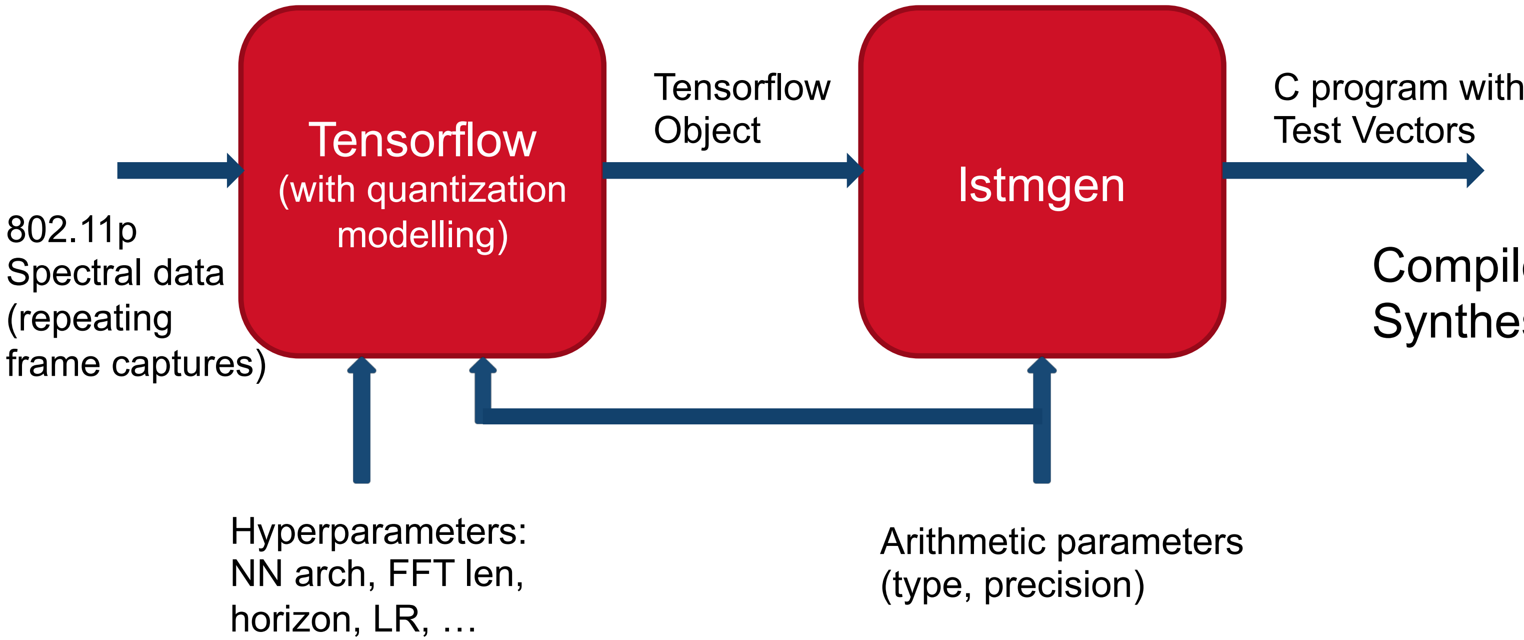
↑ a living room with a couch and a television

↑ a man riding a bike on a beach



a man is walking down the street with a suitcase ↗

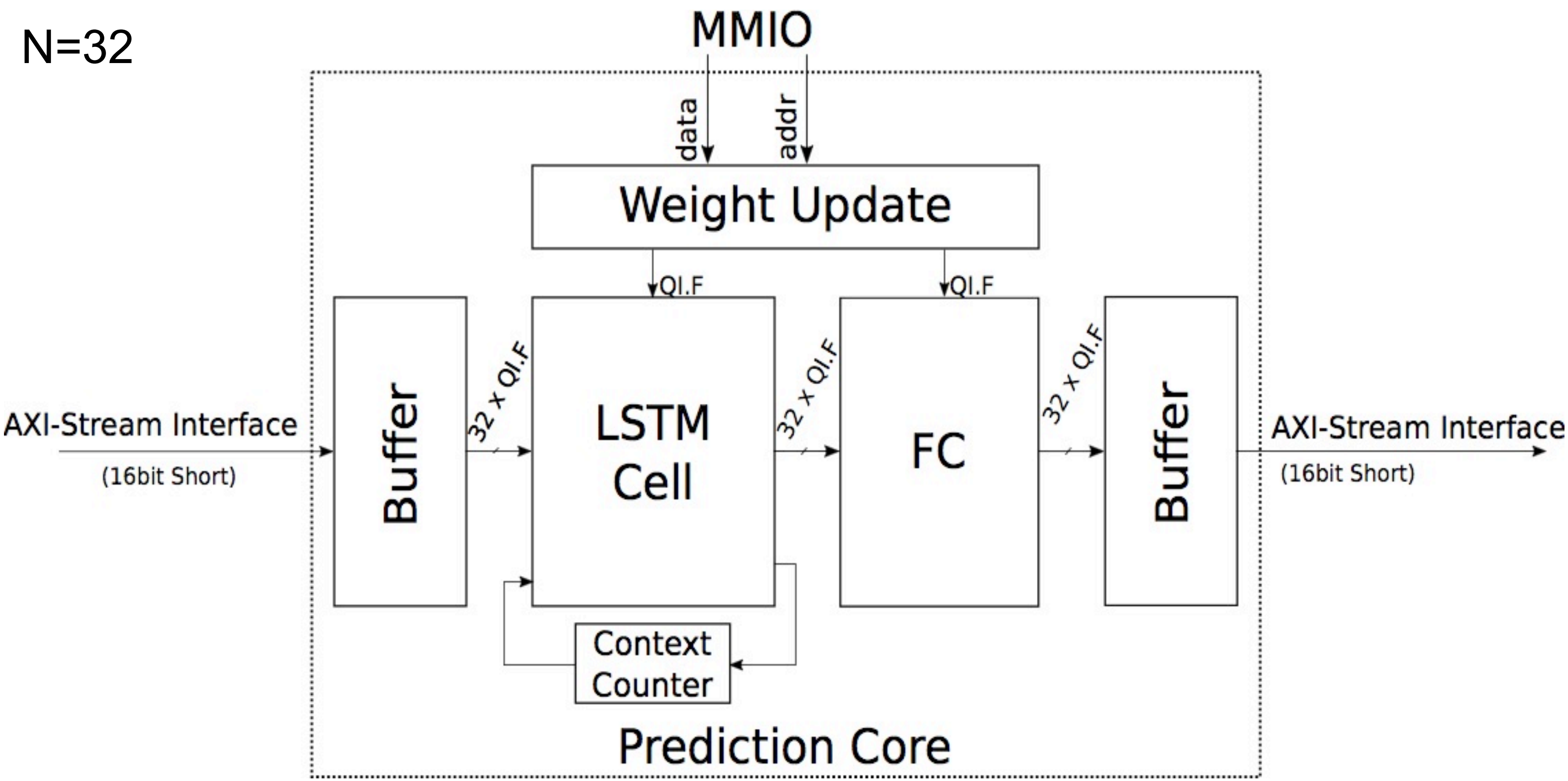
Design Flow



Compile for verification
Synthesize to FPGA

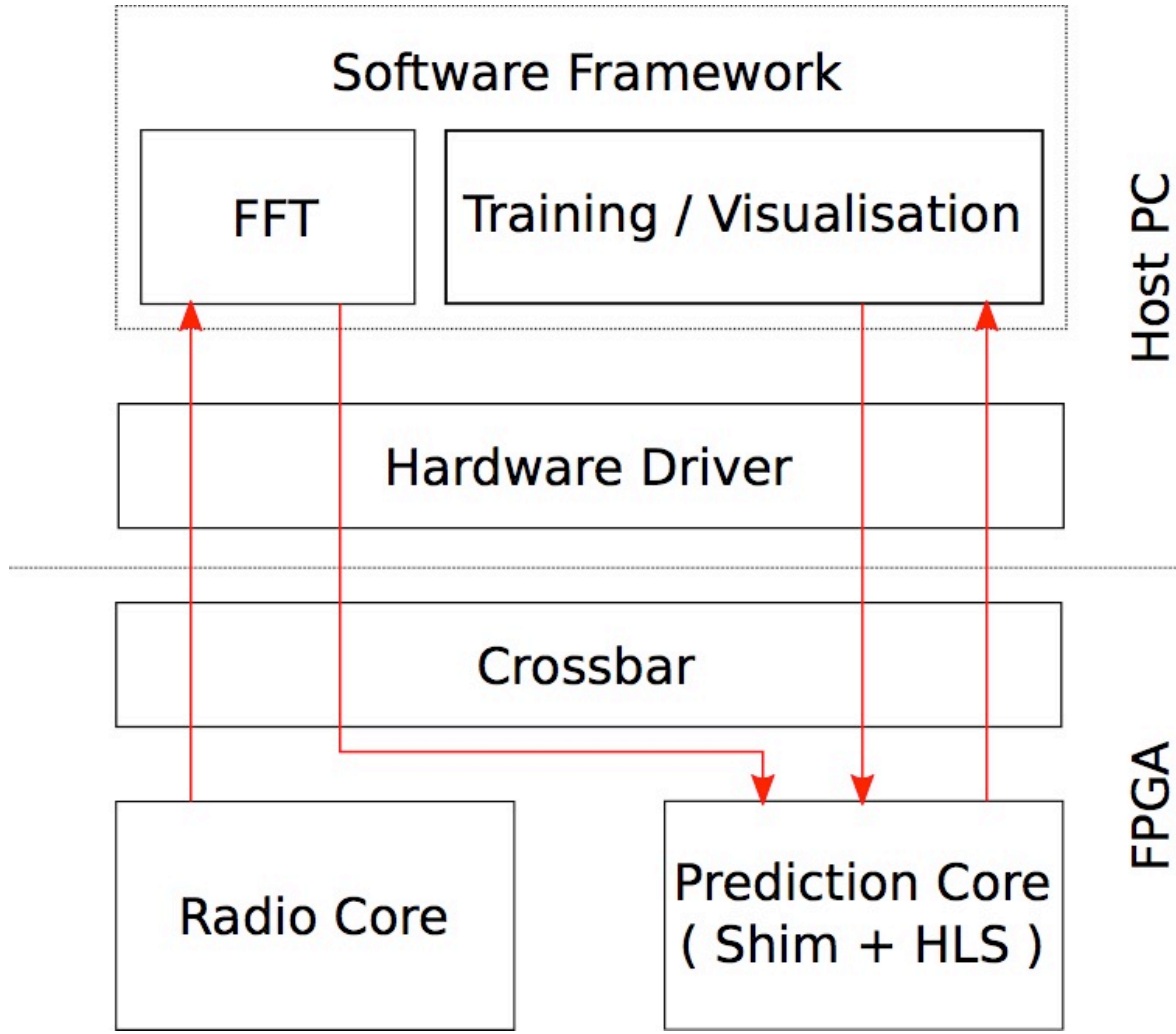
LSTM Core Architecture

N=32

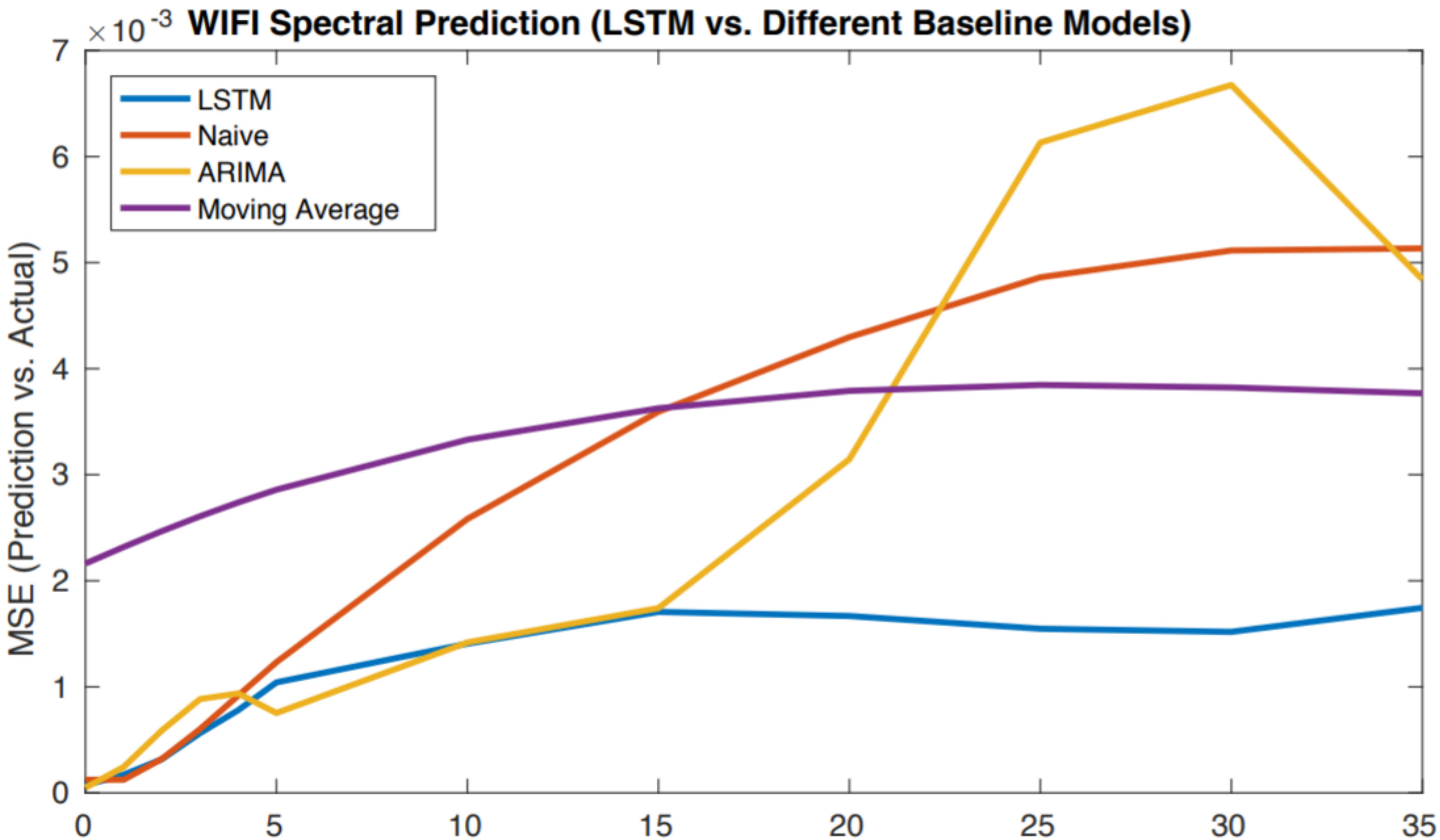


System Architecture

- Implemented on Ettus X310
- Software
  - GNU Radio integration to manage data movement
  - Offline LSTM training
- Hardware Acceleration
  - RFNoC framework
  - Prediction Core on FPGA



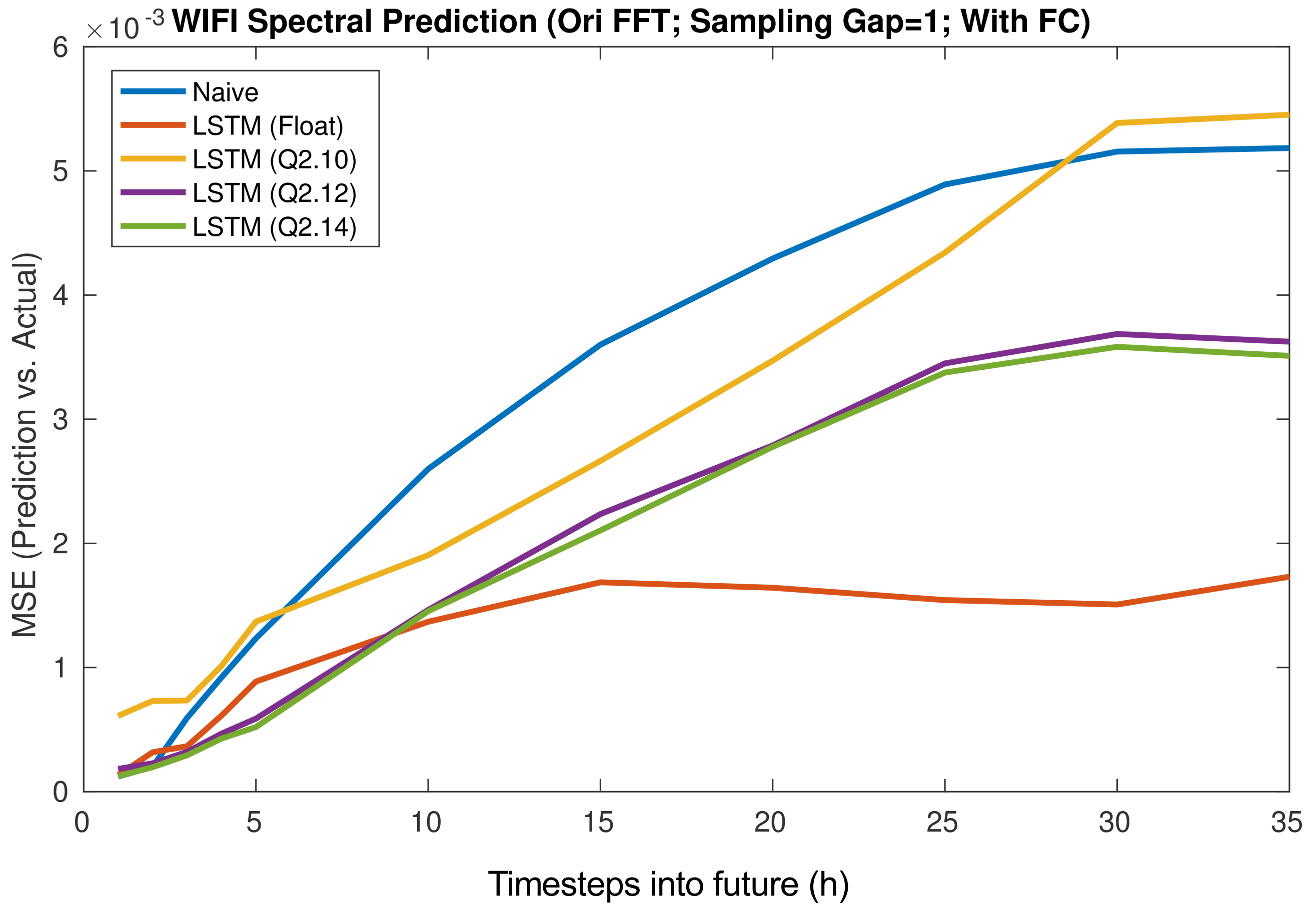
Floating-Point Accuracy



Timesteps into future (h)

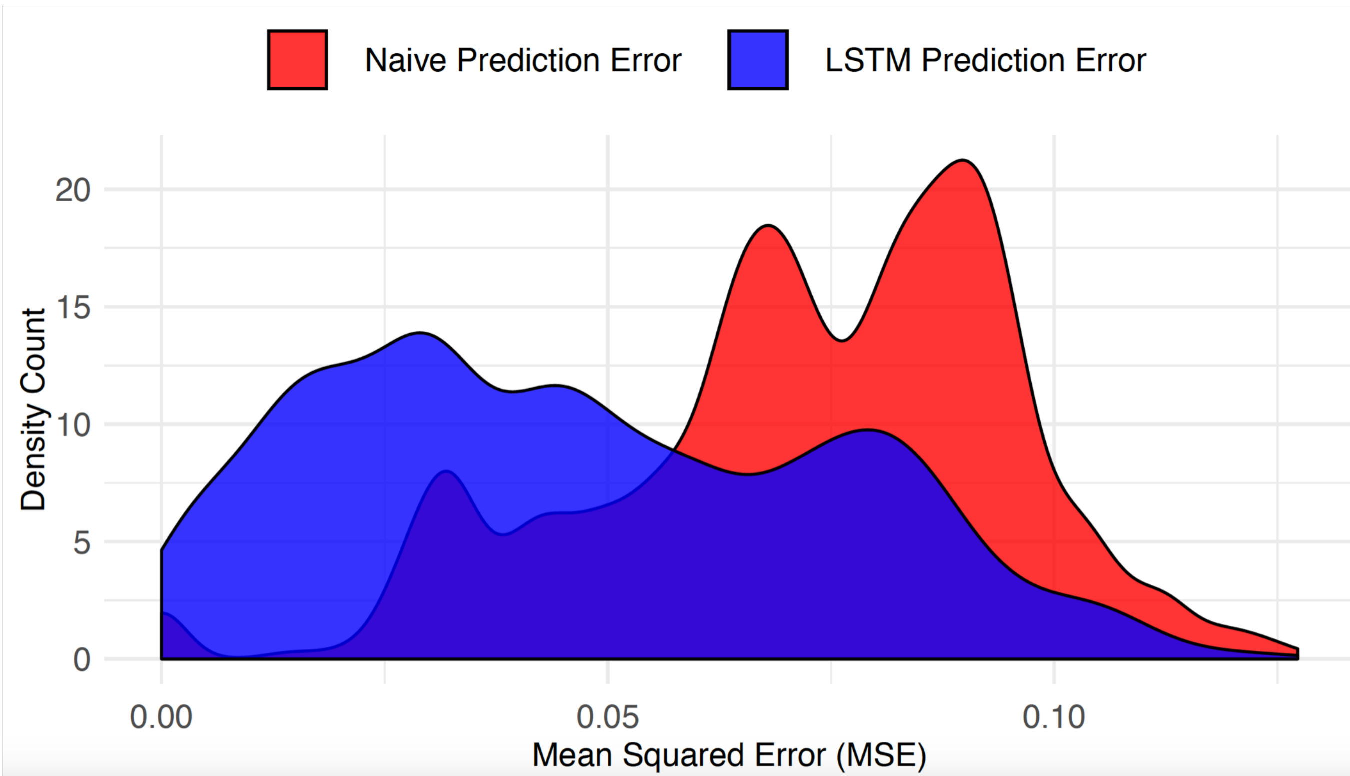
Fixed-point Implementation

- Fixed-point implementations have lower latency
- Q2.12 needed to preserve numerical accuracy



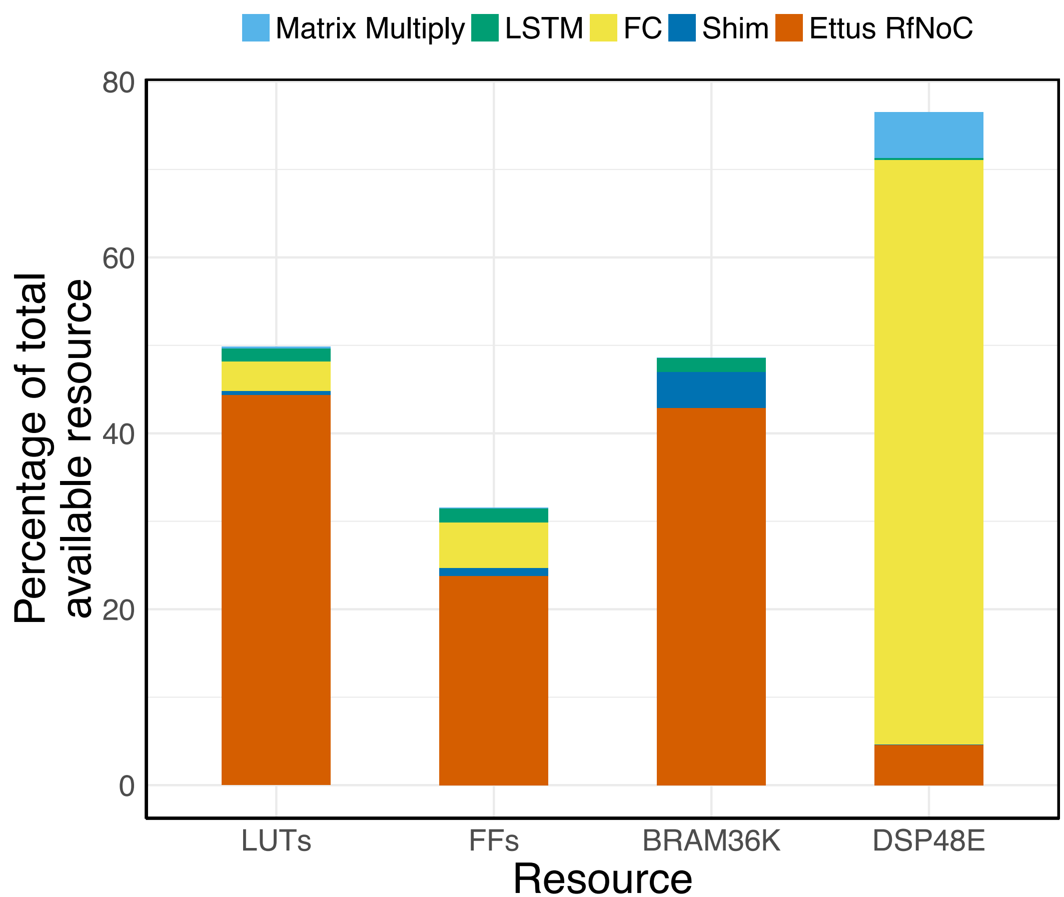
Prediction Accuracy

- N=32 history, h=4 prediction horizon
- Accuracy measured as the mean-squared error loss from true value
- LSTM gives better predictions than conventional approaches



Resource utilization

- C-code synthesised to Kintex-7 XC7K410T FPGA for Ettus X310
  - Achieves 4.3 μs latency (32 inputs and outputs)
- Limited by DSPs (~80% of 1540 available)
  - FC layer is fully unrolled to reduce prediction latency
- Most logic resources and on-chip memory used by RFNoC framework
  - Could customize design to reduce footprint and allow larger/deeper networks
  - Kintex Ultrascale with 2x more DSPs are already available



Conclusion

- Described an LSTM module generator
  - Compatible with Tensorflow
  - Generates C programs of arbitrary size, topology and precision
  - Testable and synthesisable to efficient FPGA implementation
- Low-precision fixed point LSTM can achieve better spectral prediction accuracy than conventional approaches such as Naïve or ARIMA
- Real-time LSTM-based spectral prediction feasible
  - Input/output lengths of 32; Q2.12 implementation fits easily on Ettus X310 and achieves latency of 4.3 us
- Our future research will explore how such predictions can be used to improve tactical/cognitive radios

Summary

- Presented three ideas for improving neural network performance
  - SYQ – apply symmetry to the quantisation of a CNN
  - TS multiplier – use special cases in distribution to reduce critical path (helps for relatively large wordlength)
  - LSTM – integrate all parts of a system to minimise latency
- The three ideas can be combined for greater gains in efficiency

References

Available from http://phwl.org/papers/

- 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)
- Duncan J.M. Moss, David Boland, and Philip H.W. Leong. A two-speed, radix-4, serial-parallel multiplier. *IEEE Transactions on VLSI Systems*, 2018. accepted 3 Nov, 2018.
- 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.