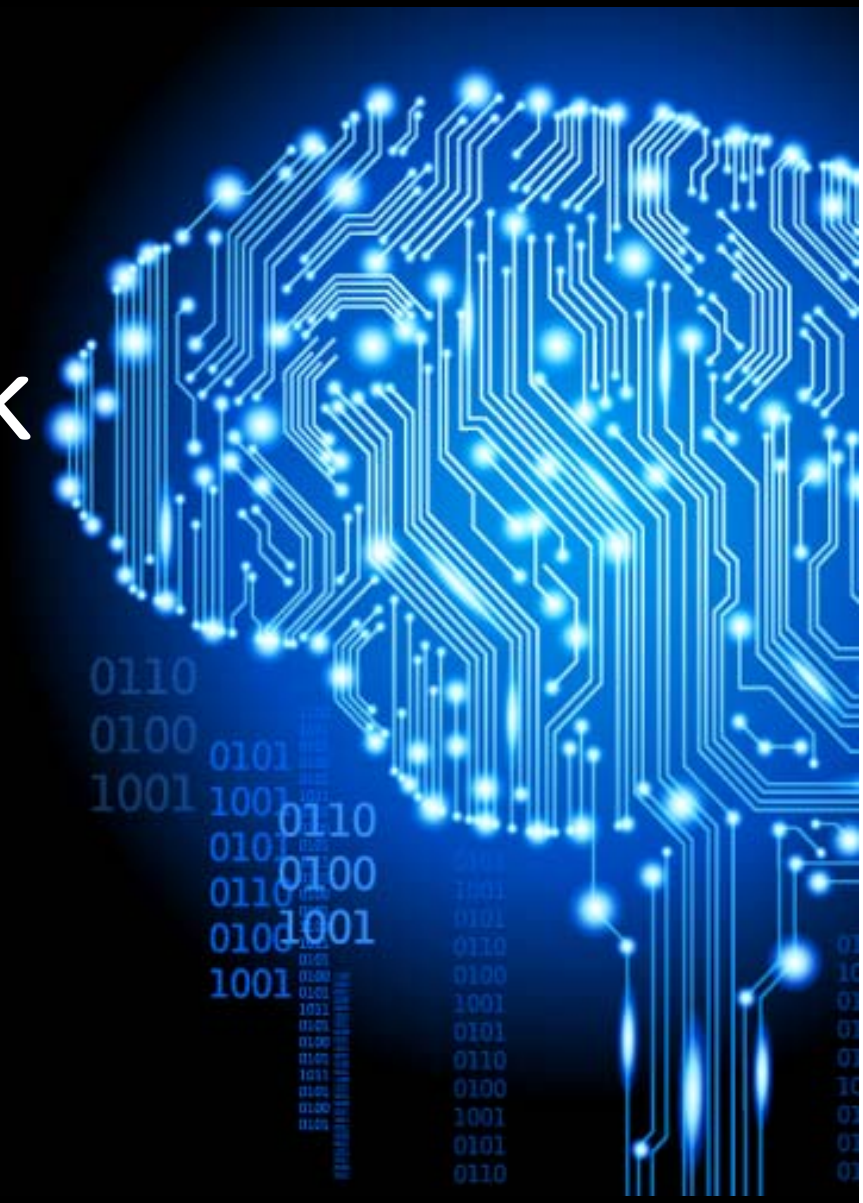
Computer System 1: A Deep Neural Network on an FPGA

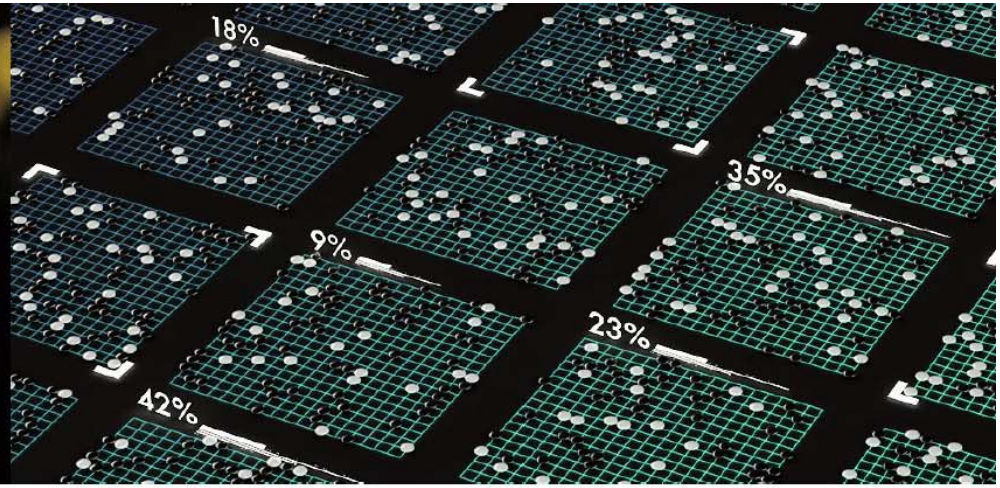
Hiroki Nakahara

Tokyo Institute of Technology, Japan

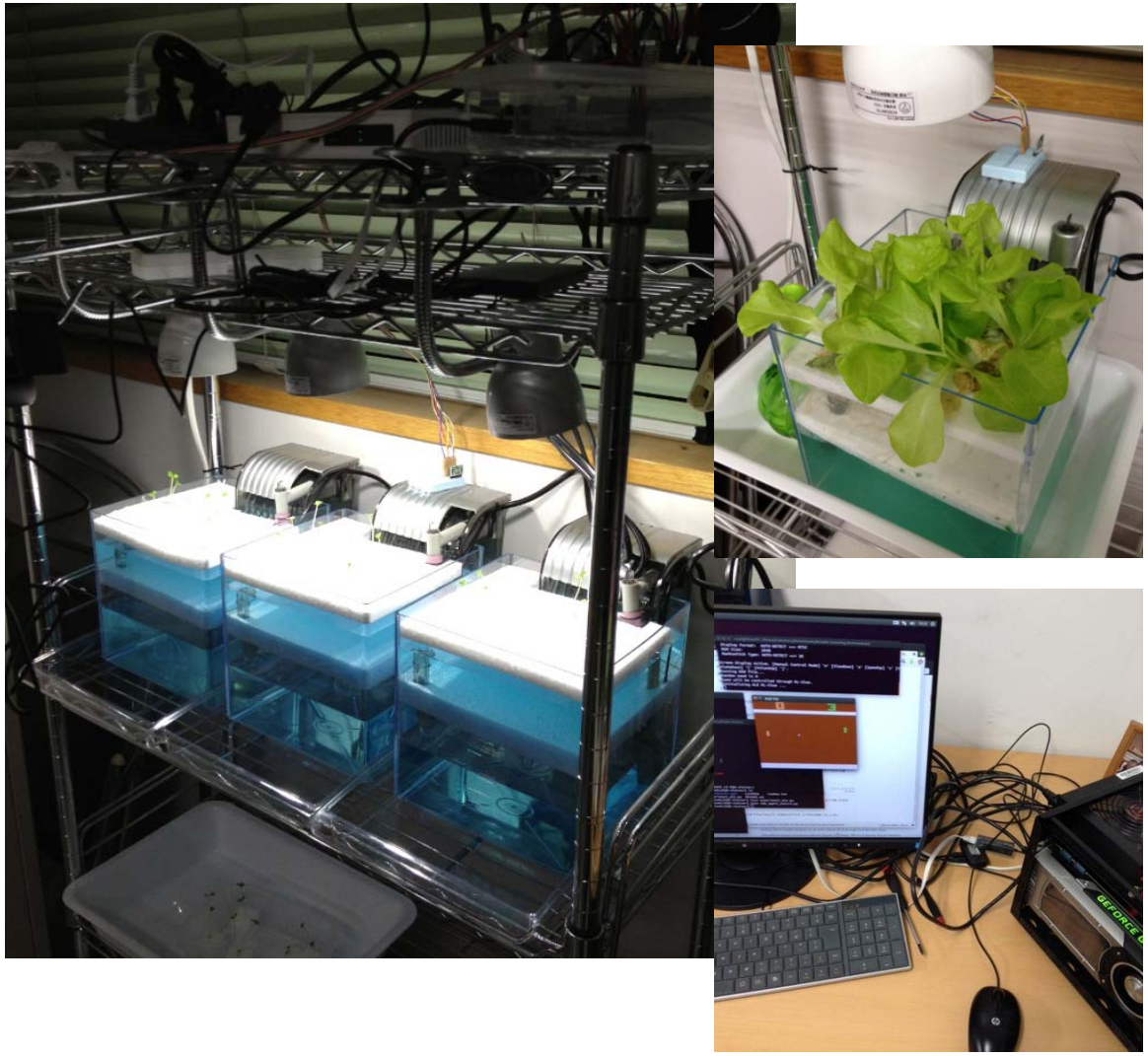


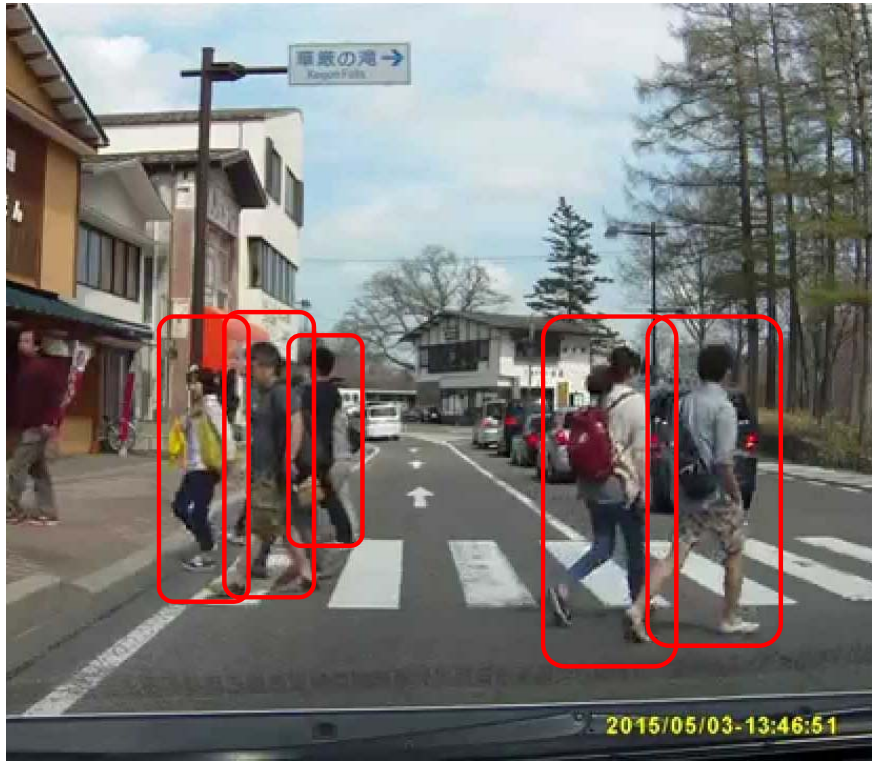
1. AI Trends

Artificial Intelligence is everywhere

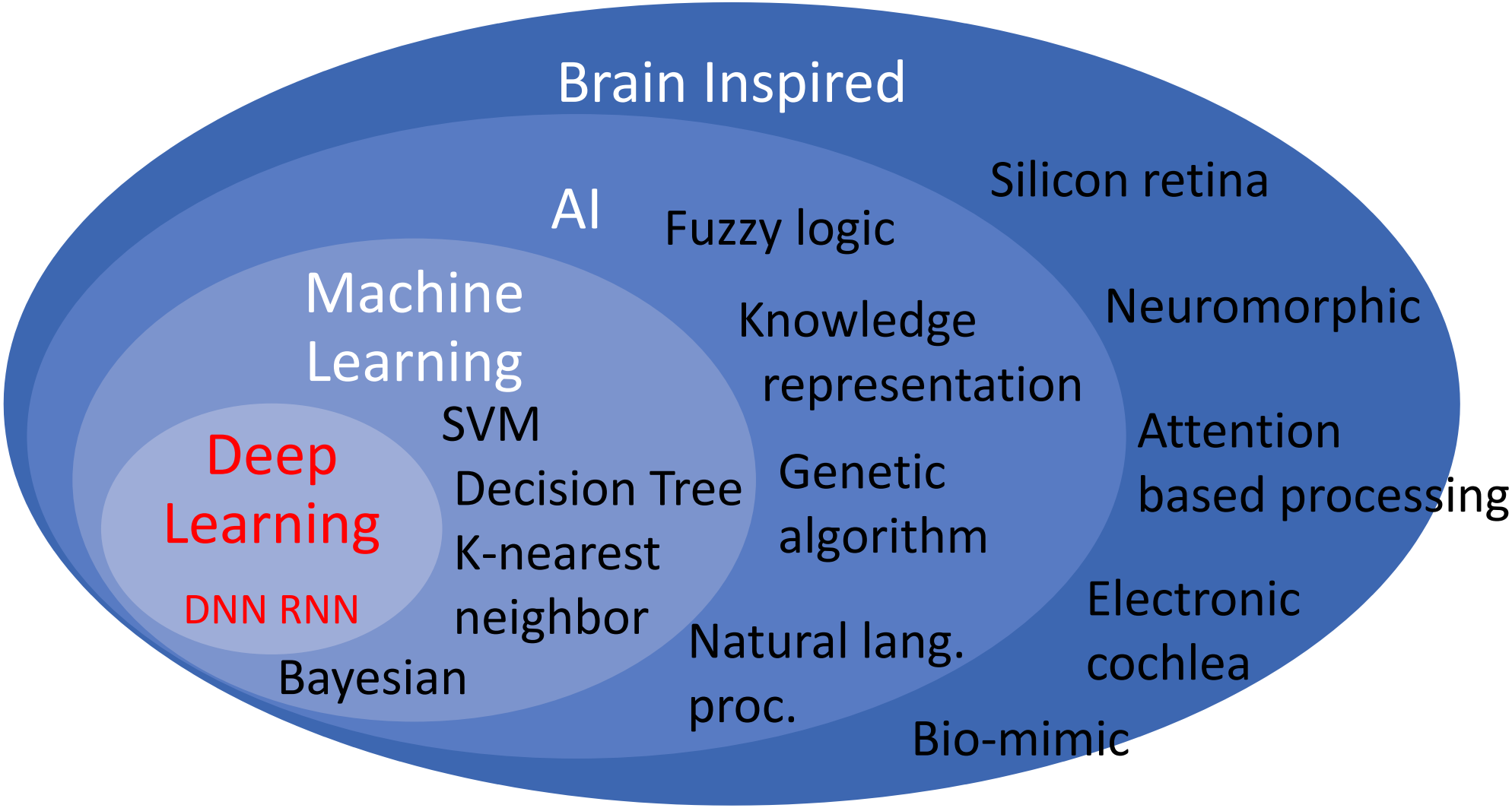


Reinforcement Learning with Deep Neural Network



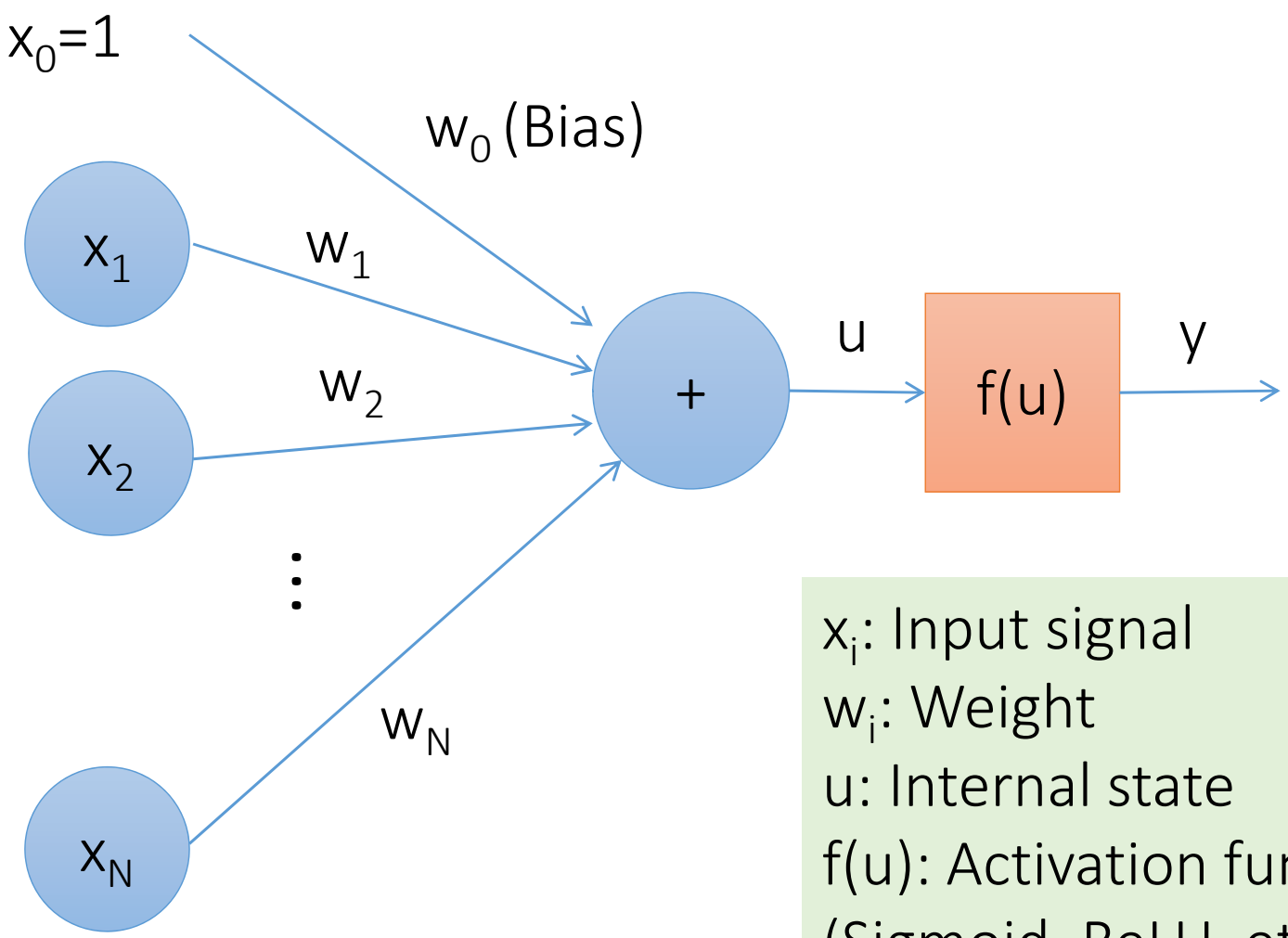


Intelligence and Deep Learning



J. Park, "Deep Neural Network SoC: Bringing deep learning to mobile devices," Deep Neural Network SoC Workshop, 2016.

Artificial Neuron (AN)

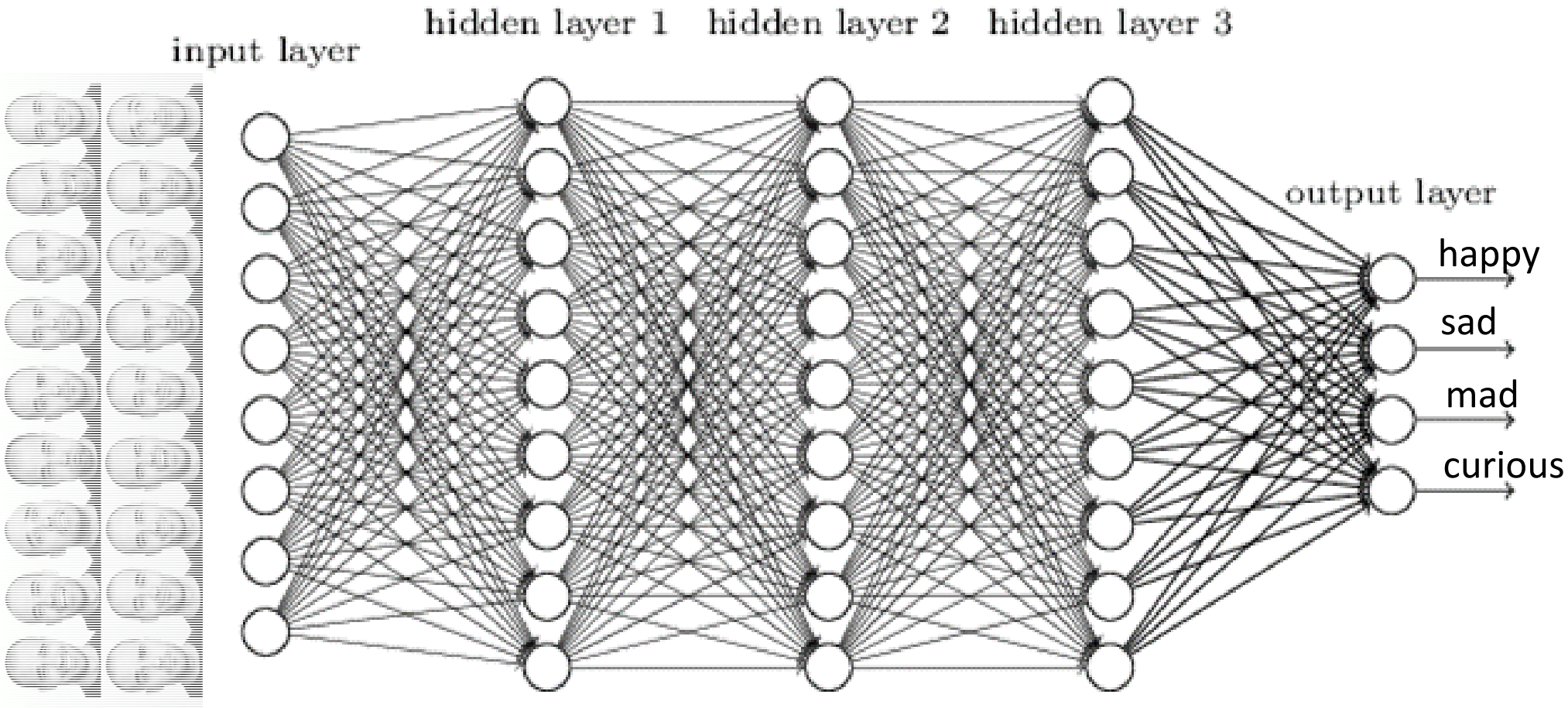


$$y = f(u)$$

$$u = \sum_{i=0}^{N} w_i x_i$$

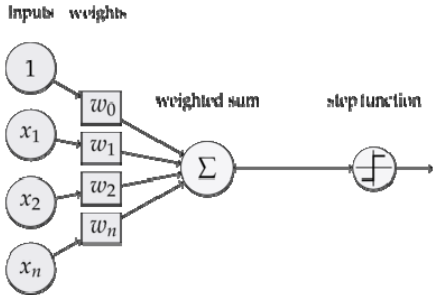
x_i: Input signal
w_i: Weight
u: Internal state
f(u): Activation function
(Sigmoid, ReLU, etc.)
y: Output signal

Deep Neural Network (DNN)



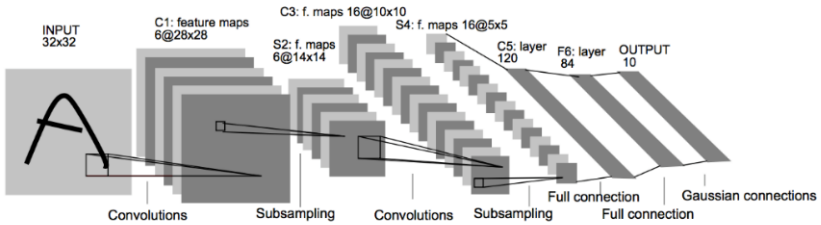
出典: imotionsglobal.com

Brief History: DNNs



Perceptron

Back-propagation



Convolutional Neural Network

Google Brain Project

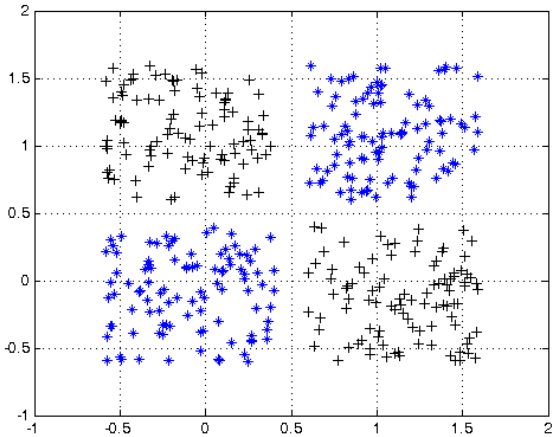
58 69 74 95 98 06 12

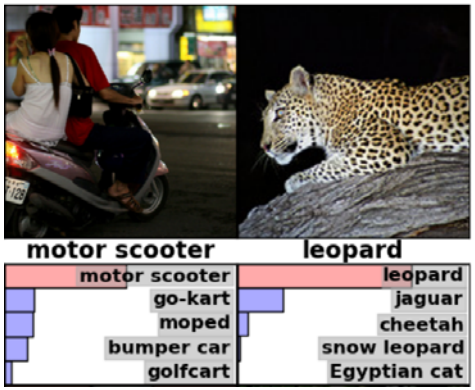
XOR Problem

SVM

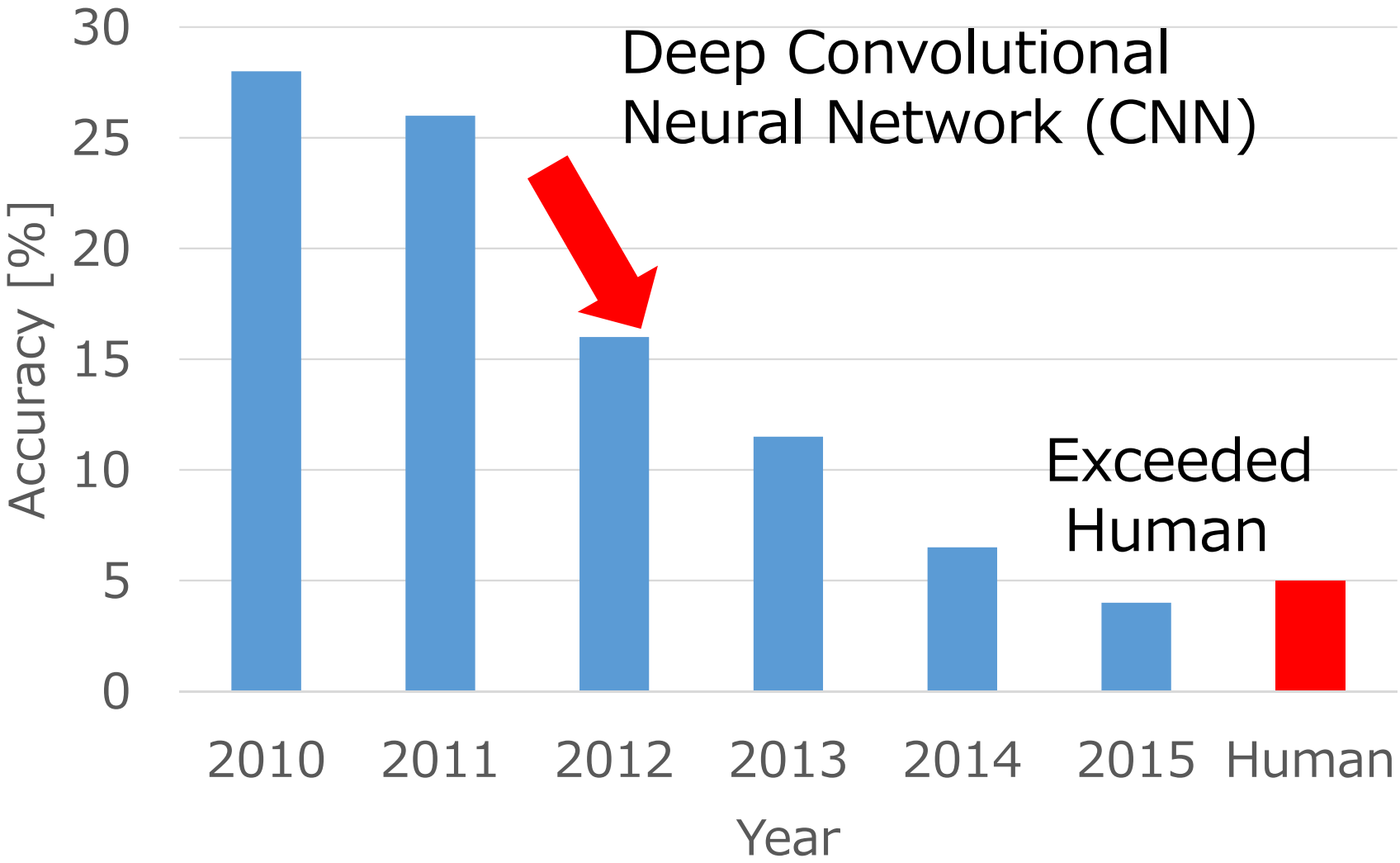
Restricted Boltzmann Machine

AlexNet Wins





Accuracy of a DNN



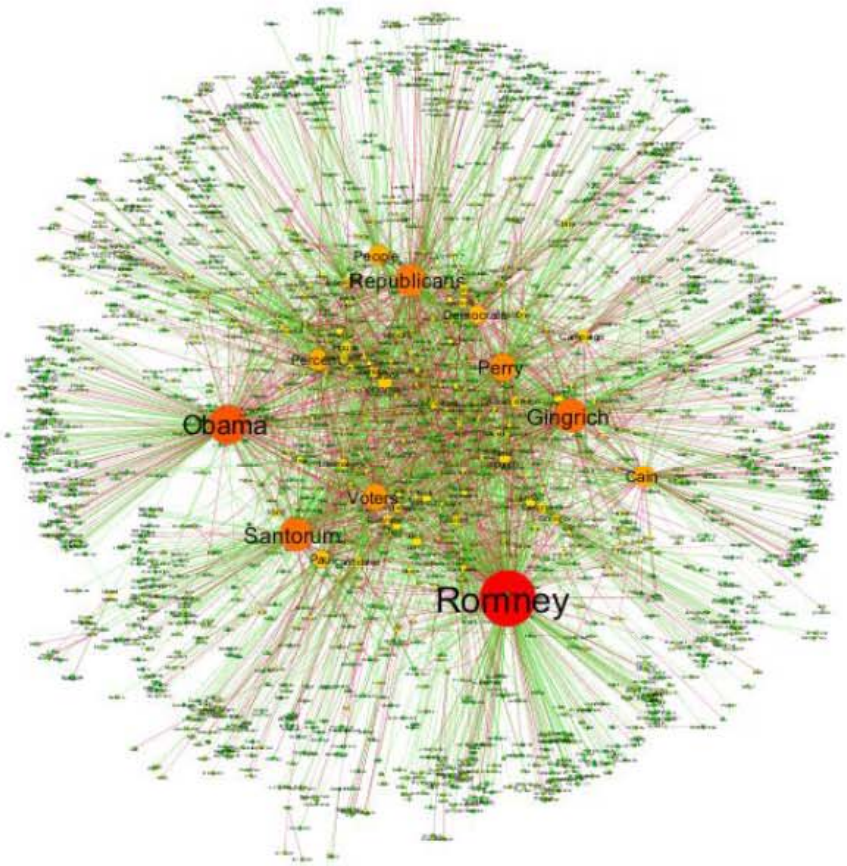
O. Russakovsky et al. "ImageNet Top 5 Classification Error (%)," IJCV 2015.

Technological singularity

- The **technological singularity** (also, simply, the **singularity**)[1] is the hypothesis that the invention of artificial superintelligence will abruptly trigger runaway technological growth, resulting in unfathomable changes to human civilization[3]
- Ray Kurzweil predicts the singularity to occur around 2045[7]

[1] M. John, "When Is the Singularity? Probably Not in Your Lifetime." The New York Times. The New York Times Company, 2016.
[2] Singularity hypotheses: A Scientific and Philosophical Assessment. Dordrecht: Springer. 2012. pp. 1–2.ISBN 9783642325601.
[3] R. Kurzweil, “The Singularity is Near”, pp. 135–136. Penguin Group, 2005.

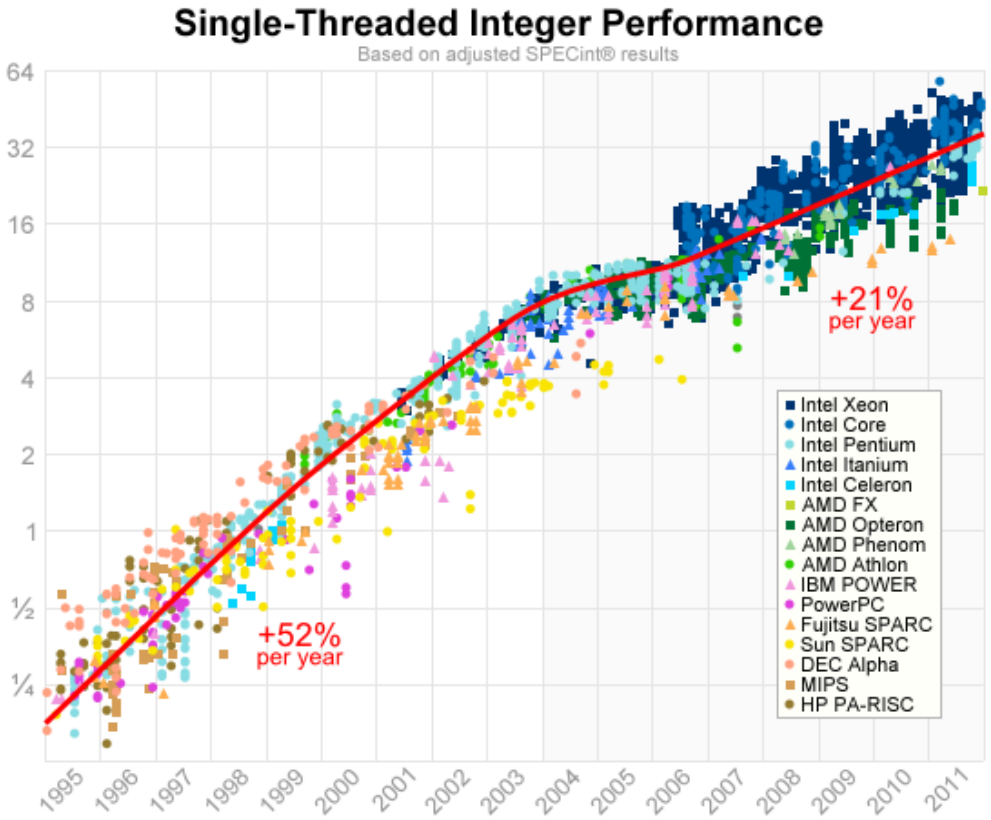
Why Deep Neural Networks?

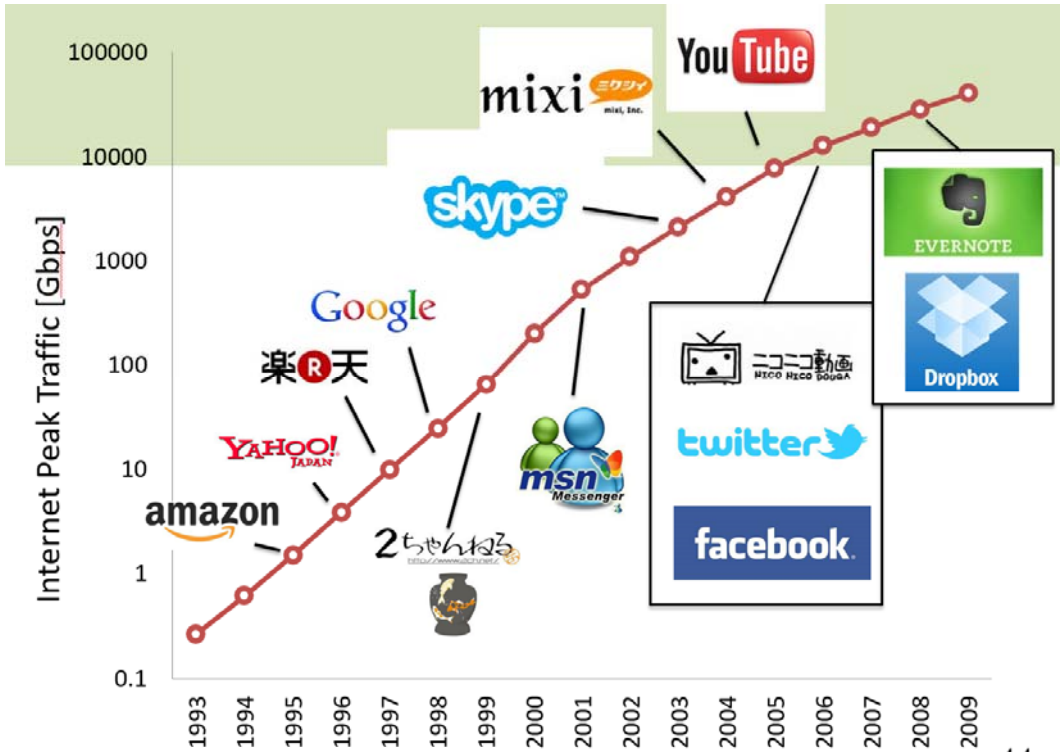


Big Data

Computational Power

Computational Power and Big Data





High performance computation,
big data, and a progress of Algorithms

(Left): "Single-Threaded Integer Performance," 2016

(Right): Nakahara, "インターネットにおける検索エンジンの技術動向(In Japanese)," 2014

DNN Frameworks

Academic…CAFFE, Industry…Tensorflow

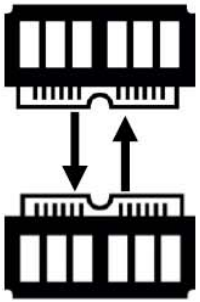
Source by nVidia Corp.

2. High performance deep neural network

Requirements for DNNs

Performance

Teraflops



Memory Bandwidth

100s of GB/s

Storage

10s of GBs

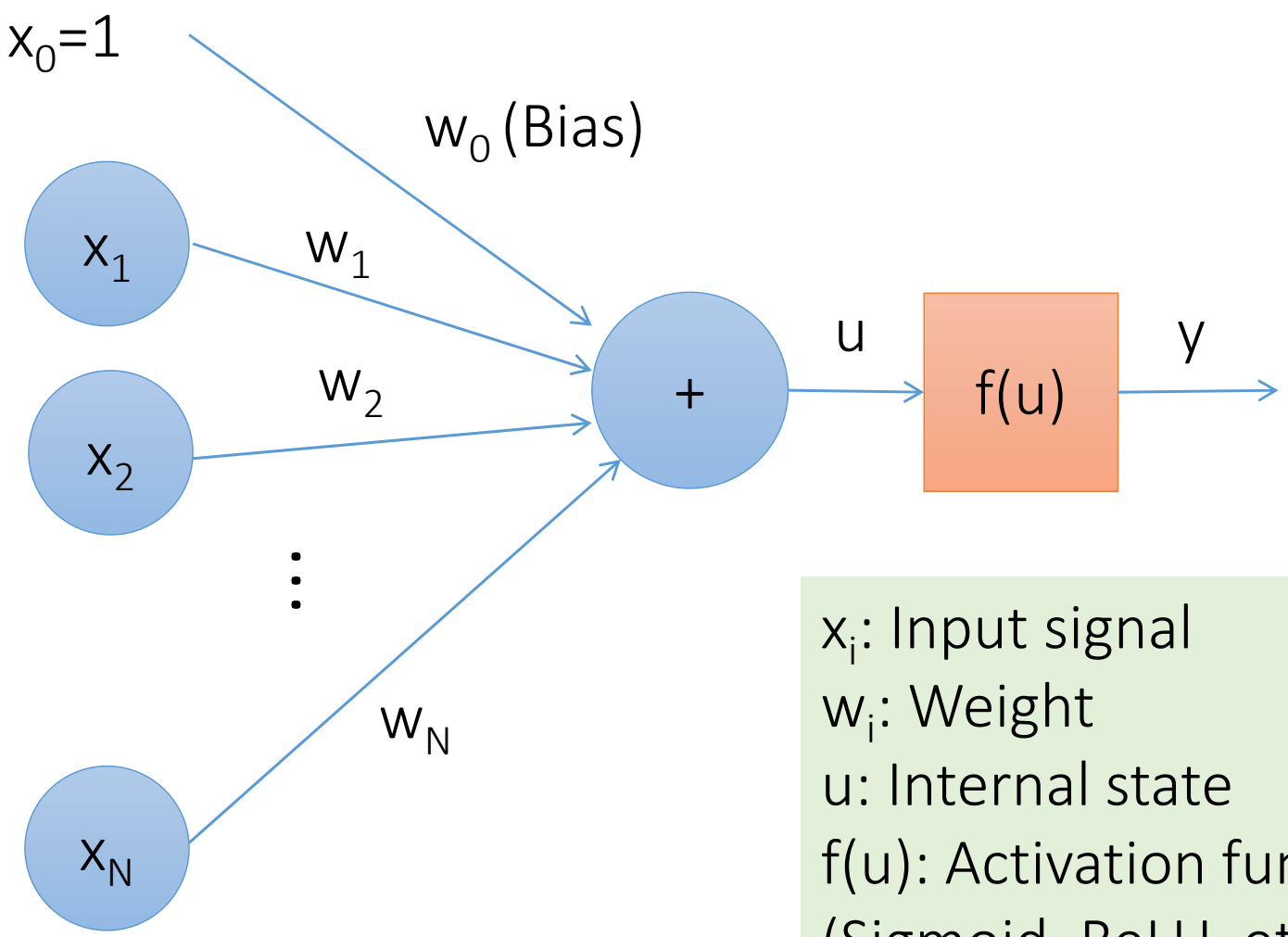
Power

100s of Watts

- 20 Billion MACs (Multiply ACcumulation operation)/image

J. Park, "Deep Neural Network SoC: Bringing deep learning to mobile devices," Deep Neural Network SoC Workshop, 2016.
J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," *Artificial Neural Networks and Machine Learning (ICANN2014)*, 2014, pp. 281-290.

Artificial Neuron (AN)



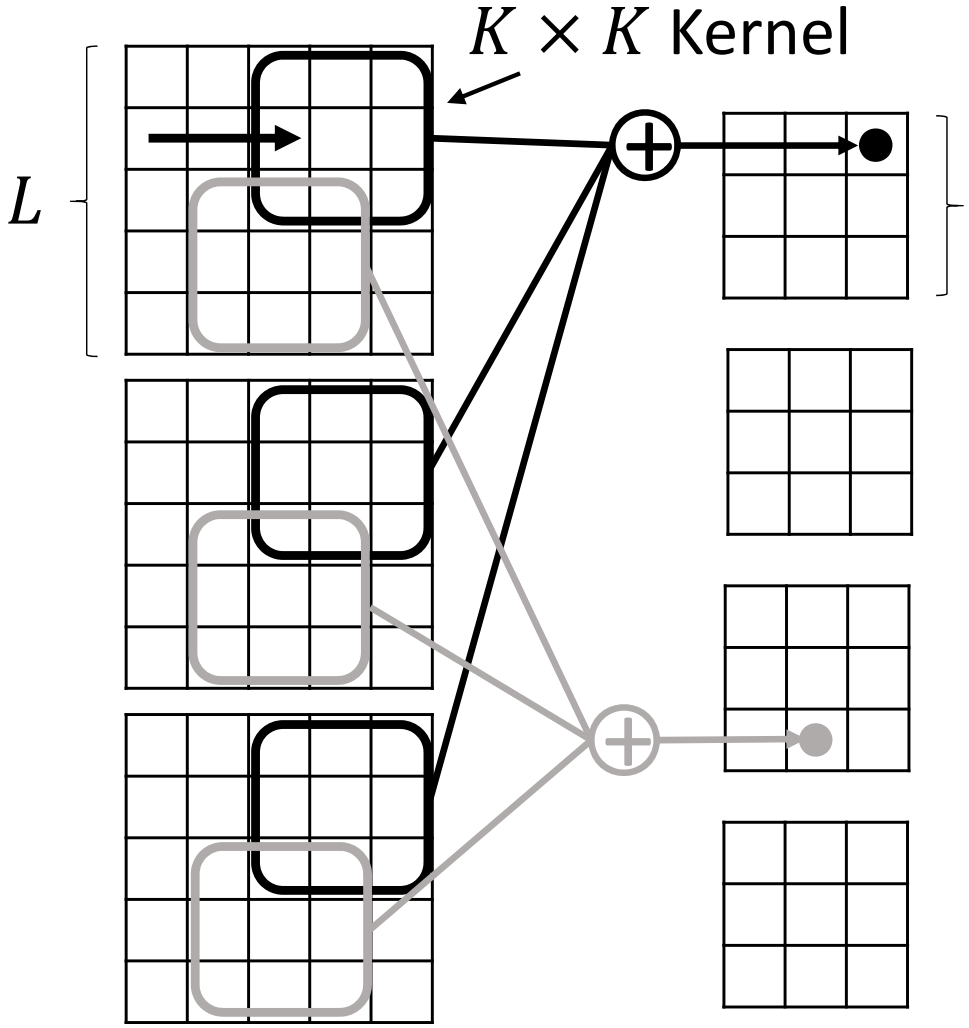
$$y = f(u)$$

$$u = \sum_{i=0}^{N} w_i x_i$$

x_i: Input signal
w_i: Weight
u: Internal state
f(u): Activation function
(Sigmoid, ReLU, etc.)
y: Output signal

2D Convolutional Layer

- Feature Map: Allocates neurons as 2D array
- Applied MAC operations for each Kernel on a feature map



$$U_{i+1,x,y} = \sum_{k=0}^{N_{i+1}} \left(Y_{k,i,j} + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} X_{k,i+m,j+n} W_{k,m,n} \right)$$

$$Z_{i+1,x,y} = f(U_{i+1,x,y})$$

X: Input signal
Y : Bias, W: Weight
K: Kernel size
N: # of Layers
f(U): Activation function
Z: Output signal

Layer i (N_i F. maps) Layer $i+1$ (N_{i+1} F. maps)

Pooling Layer

- Inserting the non-linear and low-imaging operations
  - Obtain the movement invariance
  - Reduce # of computations for the 2D convolutional layer
- Realized by a simple circuit
  - Max-pooling operation can be realized by a comparator

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

Max pooling

20	30
112	37

Average pooling

13	8
79	20

20 Billions MAC/image

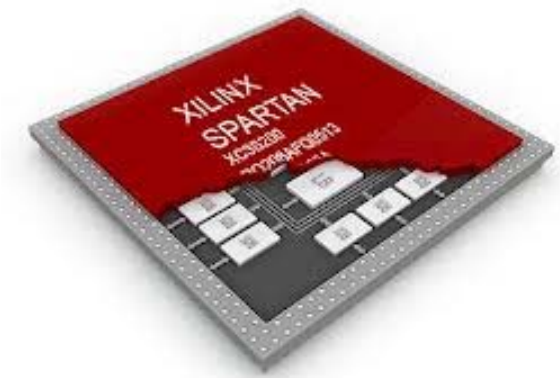
- 20B MACs=20×10^9
- Assume that 100 parallel DSP blocks@100MHz on FPGA

↓

20×10^7 MACs@100MHz

(10ns operation, thus 10^8 executions)

↓

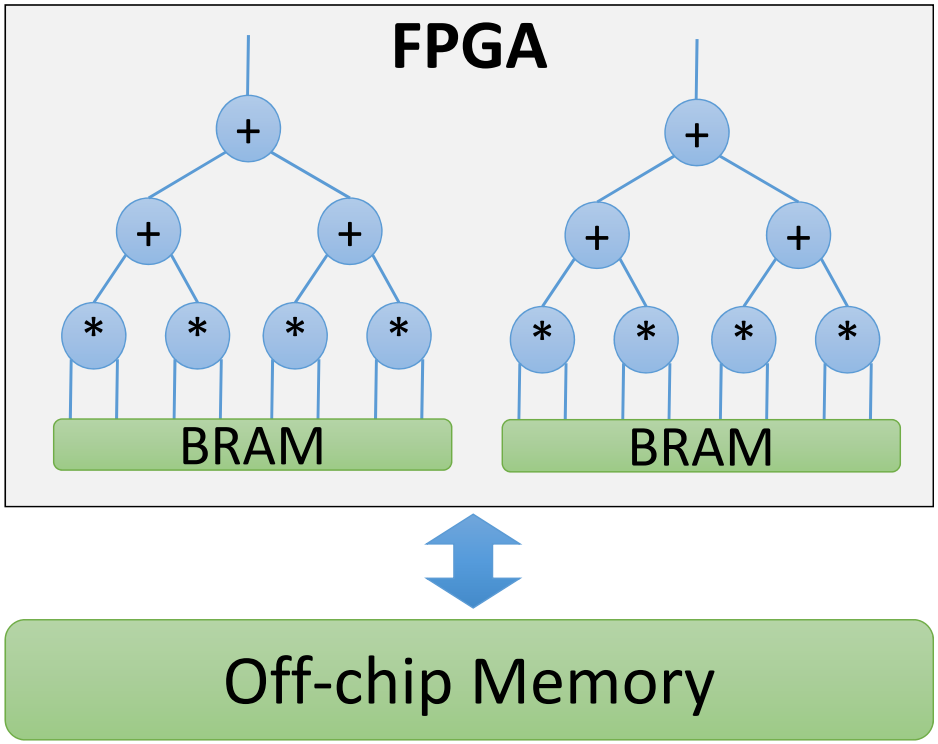


0.5 Recognizes per second

< 30 Frames per second

Conventional Realization of 2D Convolutional Layer

- Retains many layers and weights by off-chip memory
  - I/O band width, power consumption
- Massive MAC operations: Consumes area and power

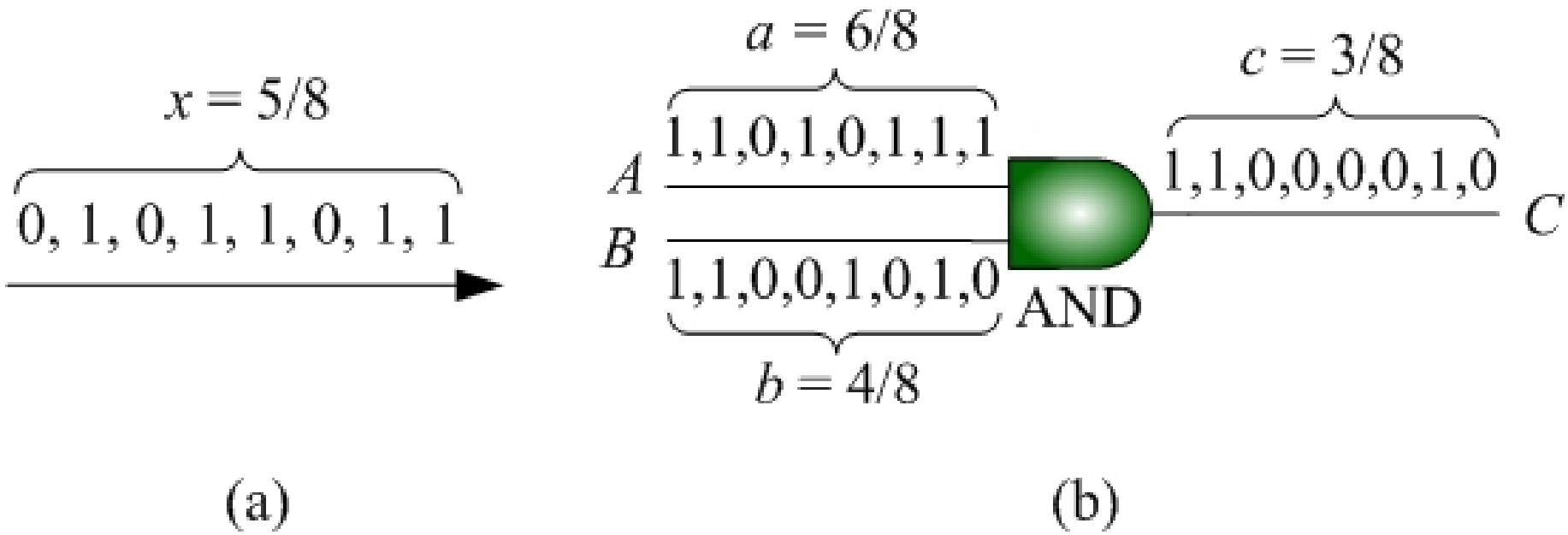


Reduction # of DSP Blocks

- High precision (Integer)
  - Residue Number System (RNS)
- Low precision (Binary)
  - Stochastic
  - Deterministic

Stochastic Multiplication

- Realized by an AND gate
  - Addition is realized by an multiplexer
- High precision, but slow



Deterministic Multiplication

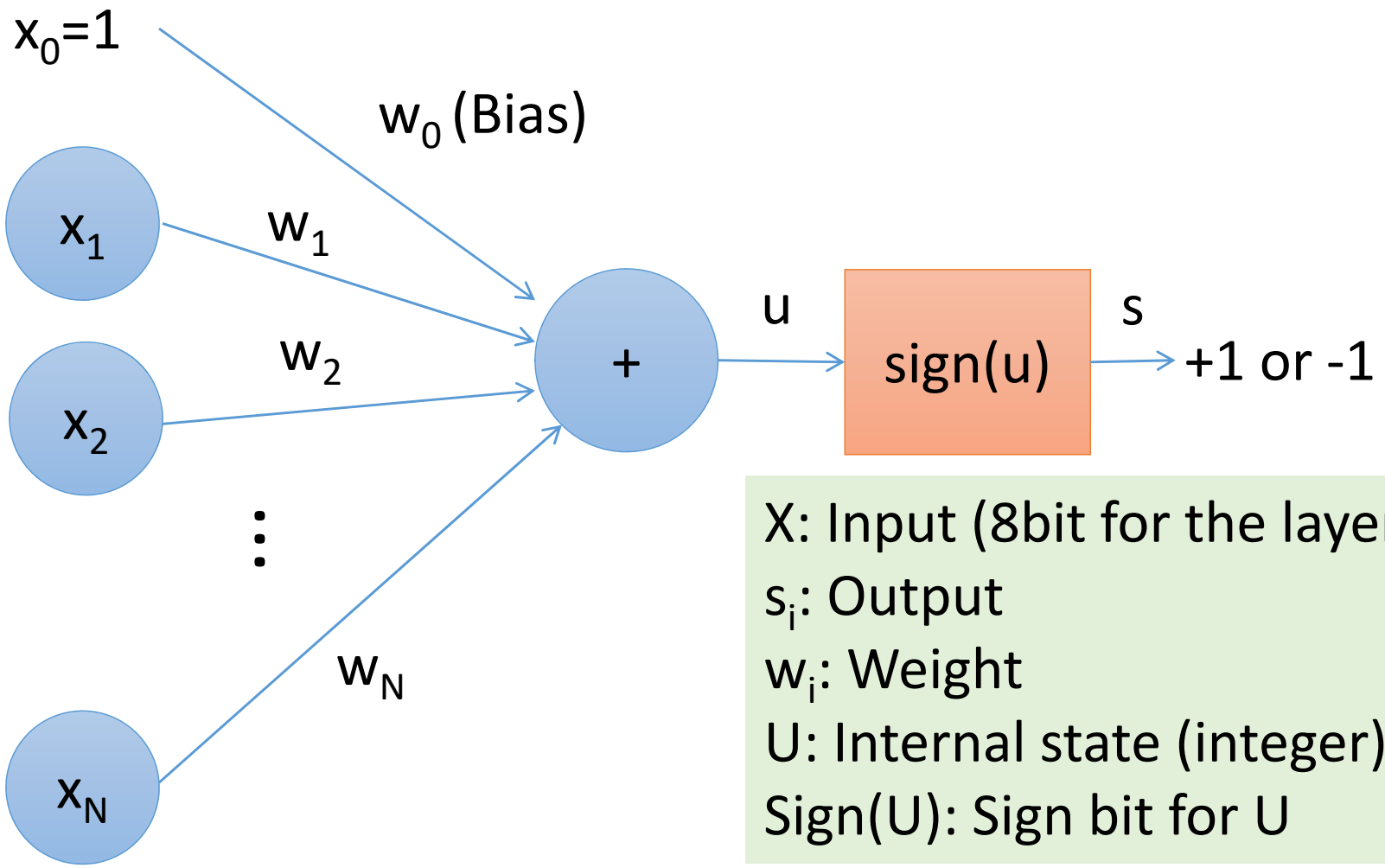
- Realized by an XNOR gate
- Fast, but low precision

x1	x2	Y
-1	-1	1
-1	+1	-1
+1	-1	-1
+1	+1	1

x1	x2	Y
0	0	1
0	1	0
1	0	0
1	1	1

Binarized DCNN

- Treats only binarized (+1/-1) values (weights and inouts)
  - Except for the first and the last layers



X: Input (8bit for the layer 1)
s_i: Output
w_i: Weight
U: Internal state (integer)
Sign(U): Sign bit for U

M. Courbariaux, I. Hubara, D. Soudry, R.E.Yaniv, Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *Computer Research Repository (CoRR)*, Mar., 2016, http://arxiv.org/pdf/1602.02830v3.pdf

2D Convolutional Layer for Binarized DCNN

- Sum of Binarized-Weights operations instead of MAC operations in integer

$$U_{i+1,x,y} = \sum_{k=0}^{N_{i+1}} \left(Y_{k,i,j} + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} X_{k,i+m,j+n} \oplus W_{k,m,n} \right)$$

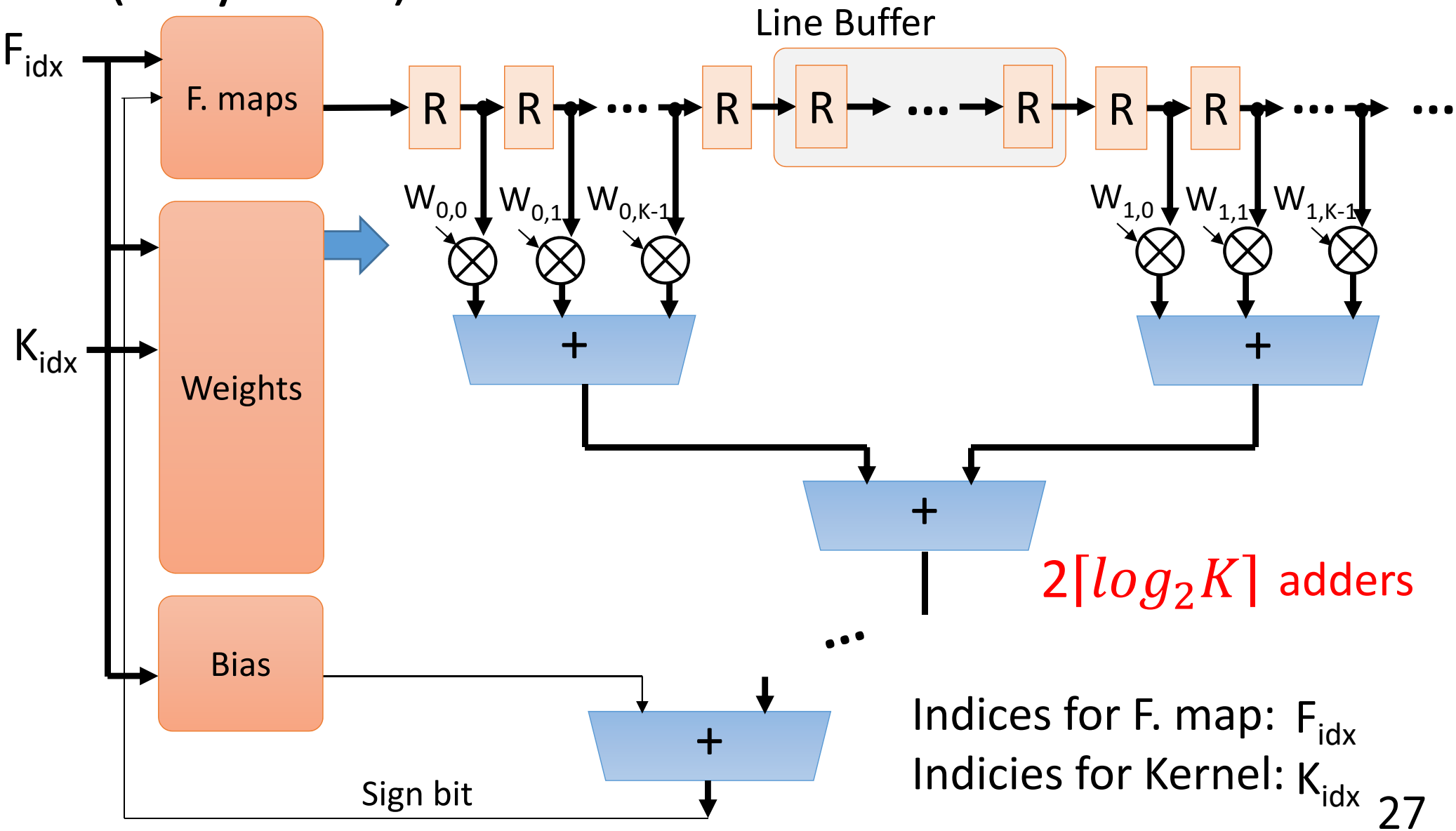
$$Z_{i+1,x,y} = Sign(U_{i+1,x,y}) = \begin{cases} +1 & if \;\; U_{i+1,x,y} \geq 0, \\ -1 & otherwise. \end{cases}$$

x: Input signal (Binary)
y : Bias (Binary)
w: Weight (Binary)
K: Kernel size
N: # of Layers
f(U): Activation function
Z: Output signal (Binary)

Example of Sum of Binarized-weights (n=5)

s_0	s_1	s_2	s_3	s_4	f(S)
0	0	0	0	0	$-w_0-w_1-w_2-w_3-w_4$
0	0	0	0	1	$-w_0-w_1-w_2-w_3+w_4$
0	0	0	1	0	$-w_0-w_1-w_2+w_3-w_4$
0	0	0	1	1	$-w_0-w_1-w_2-w_3+w_4$
0	0	1	0	0	$-w_0-w_1+w_2-w_3-w_4$
0	0	1	0	1	$-w_0-w_1+w_2-w_3+w_4$
0	0	1	1	0	$-w_0-w_1+w_2+w_3-w_4$
0	0	1	1	1	$-w_0-w_1+w_2+w_3+w_4$
0	1	0	0	0	$-w_0+w_1-w_2-w_3-w_4$
0	1	0	0	1	$-w_0+w_1-w_2-w_3+w_4$
0	1	0	1	0	$-w_0+w_1-w_2+w_3-w_4$
0	1	0	1	1	$-w_0+w_1-w_2+w_3+w_4$
		⋮			⋮
1	1	1	1	1	$+w_0+w_1+w_2+w_3+w_4$

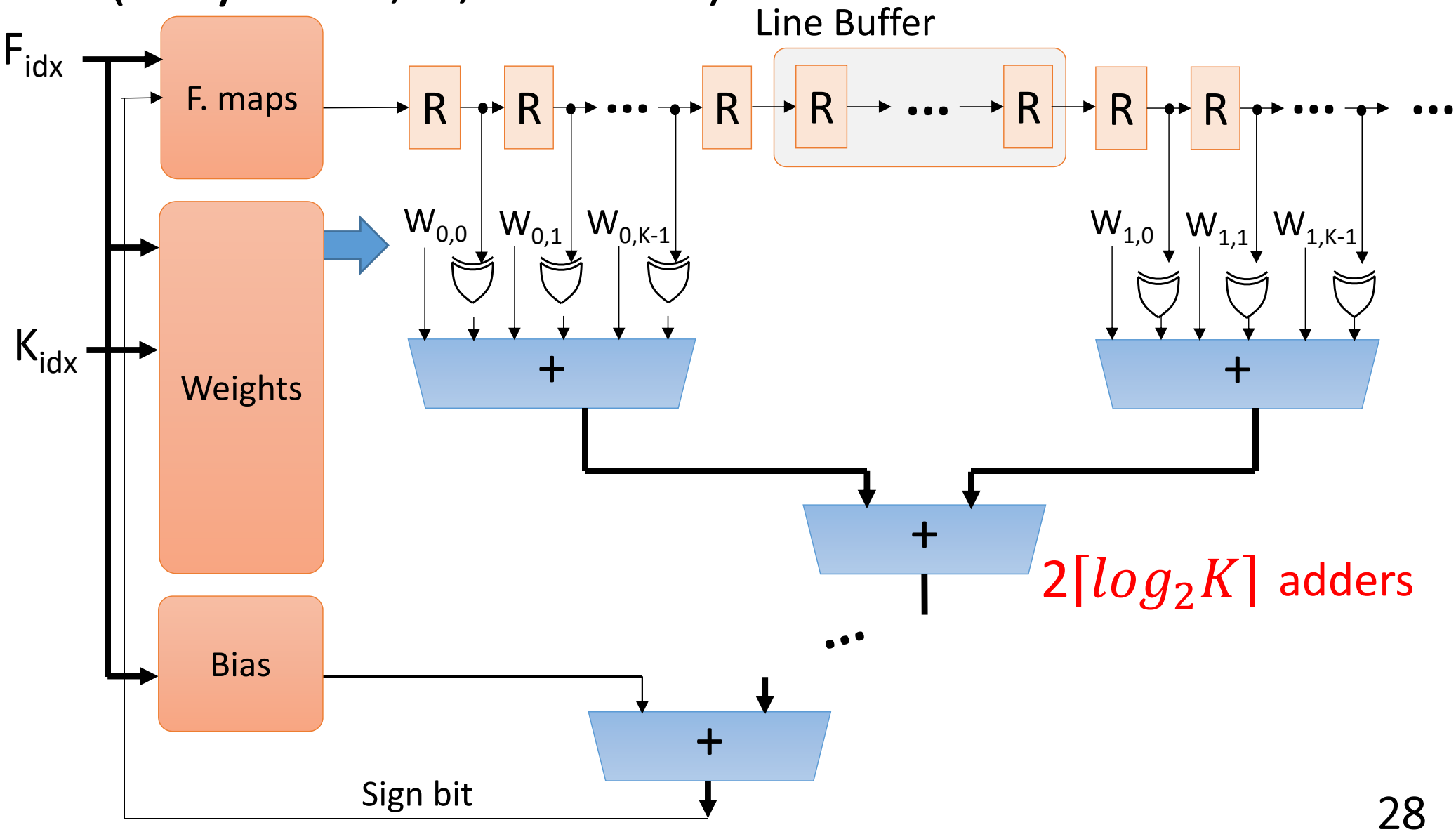
Straightforward Realization (Layer 1)



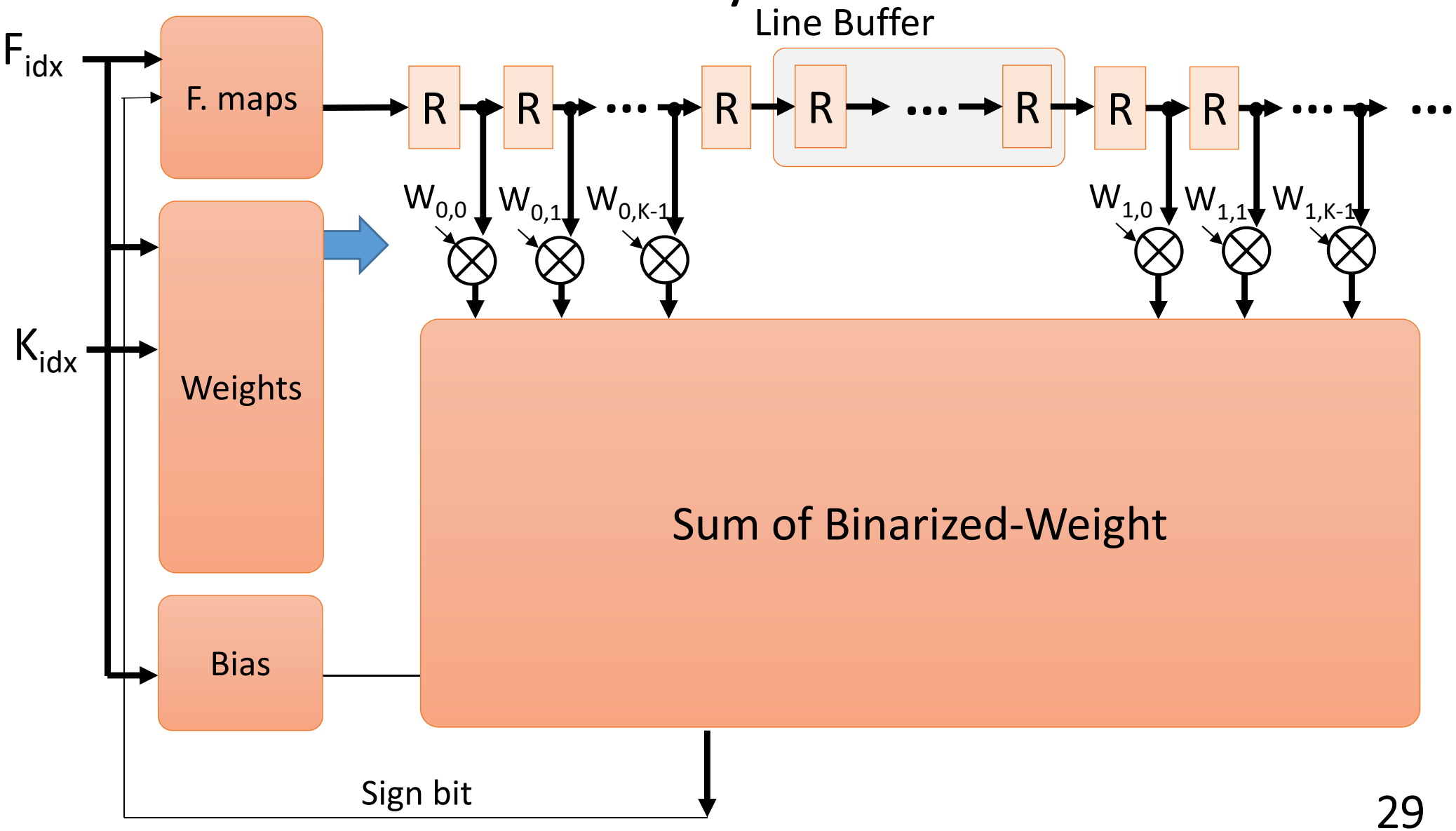
Indices for F. map: F_{idx}

Indicies for Kernel: K_{idx}

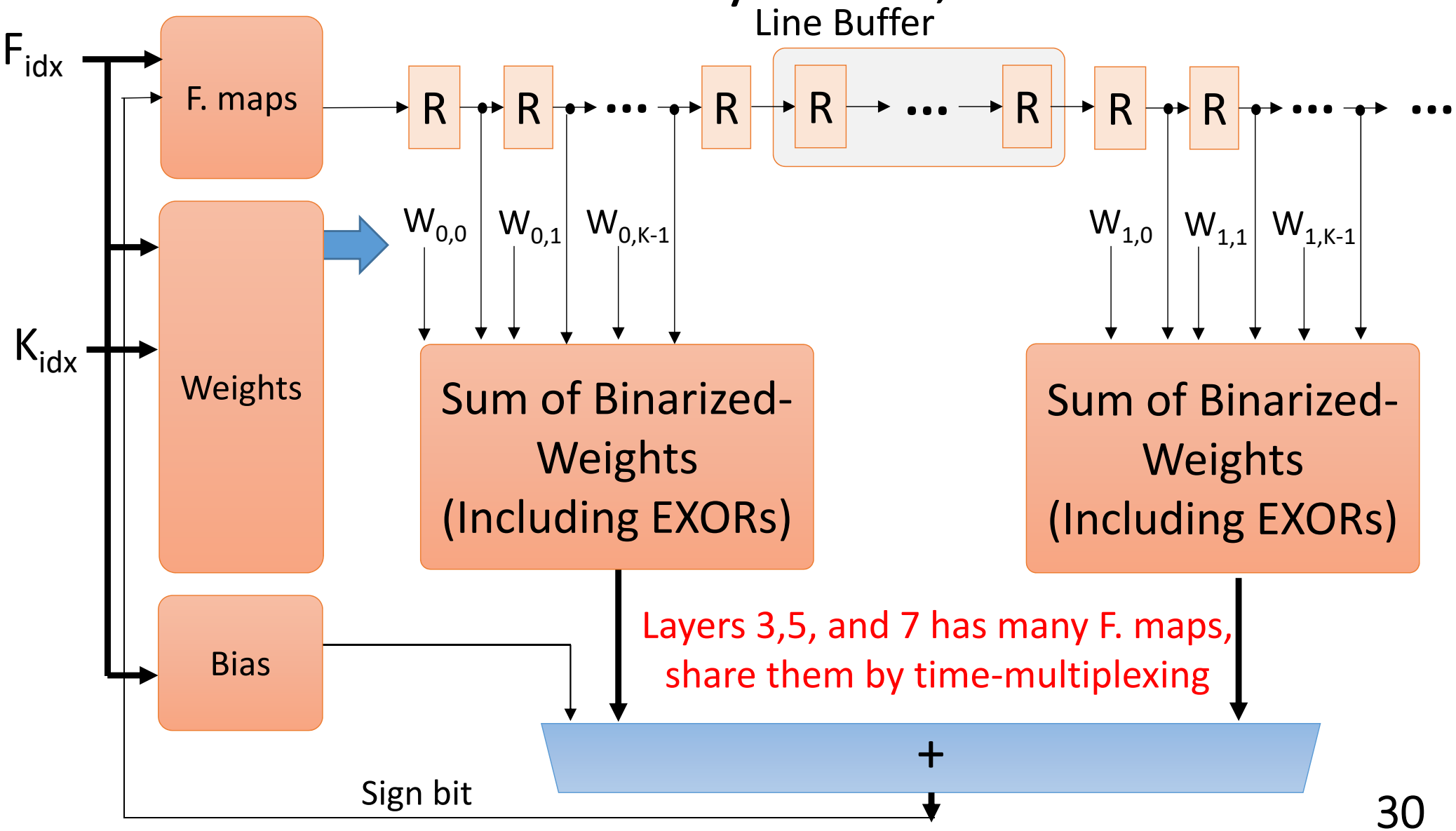
Straightforward Realization (Layer 3,5, and 7)



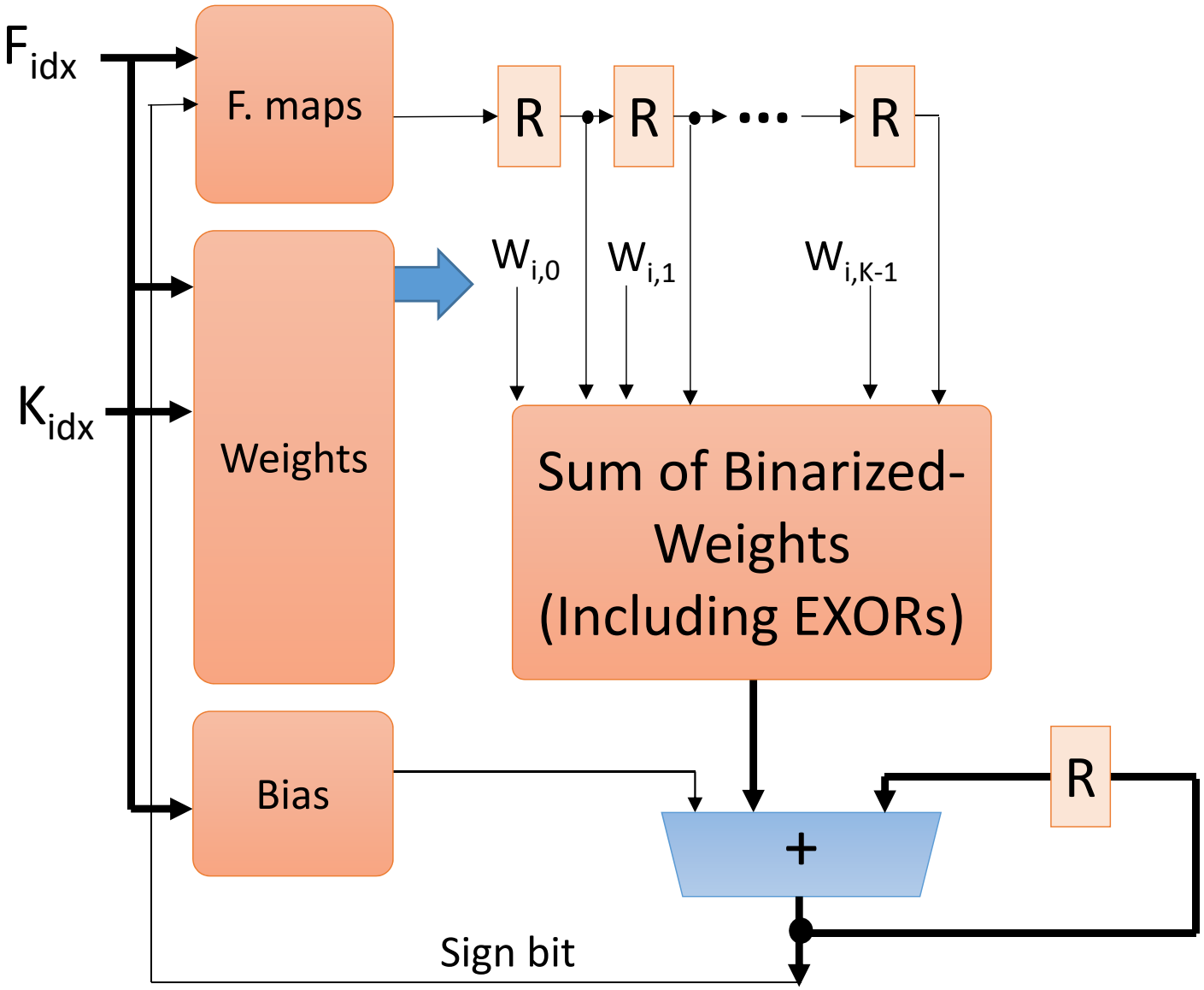
Straightforward Memory-based Realization for Layer 1



Straightforward Memory-based Realization for Layers 3, 5 and 7



Memory Reduction by Time-multiplexing



Functional Decomposition

Decomposition Chart

Bound Variables

$X_1=(x_1, x_2)$

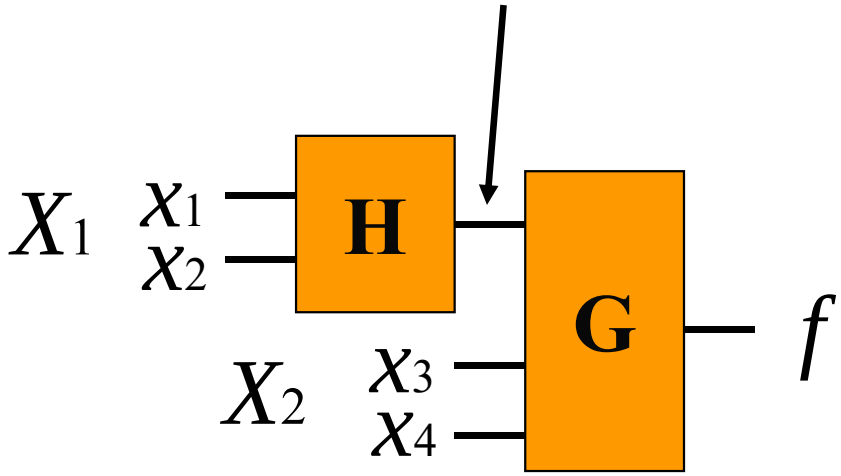
Free Variables

$X_2=(x_3, x_4)$

	00	01	10	11
00	**0**	**1**	**0**	**1**
01	**1**	**1**	**1**	**1**
10	**1**	**0**	**1**	**0**
11	**1**	**0**	**1**	**0**
$h(X_1)$	0	1	0	1

Column Multiplicity $\mu=2$

# of rails = $\lceil \log_2 \mu \rceil$

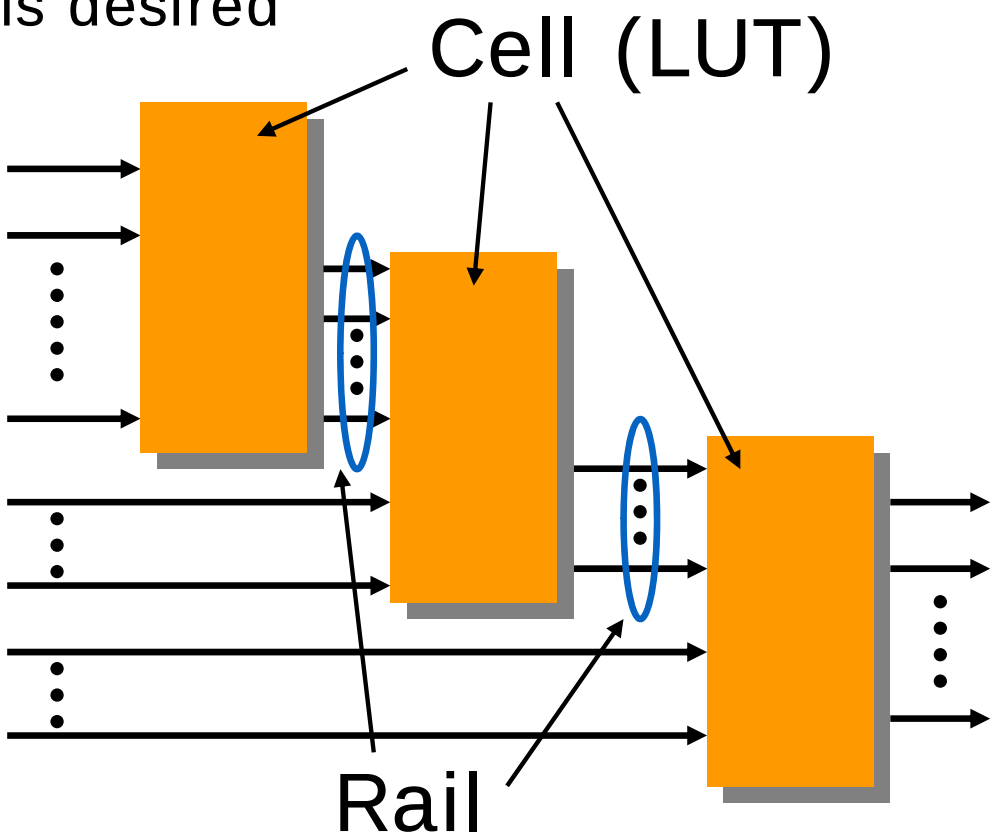


$f=g(h(X_1),X_2)$

LUT Cascade

- Applied functional decompositions recursively
- Memory size is depended on # of rails
  - Small column multiplicity is desired

Single Memory



LUT Cascade

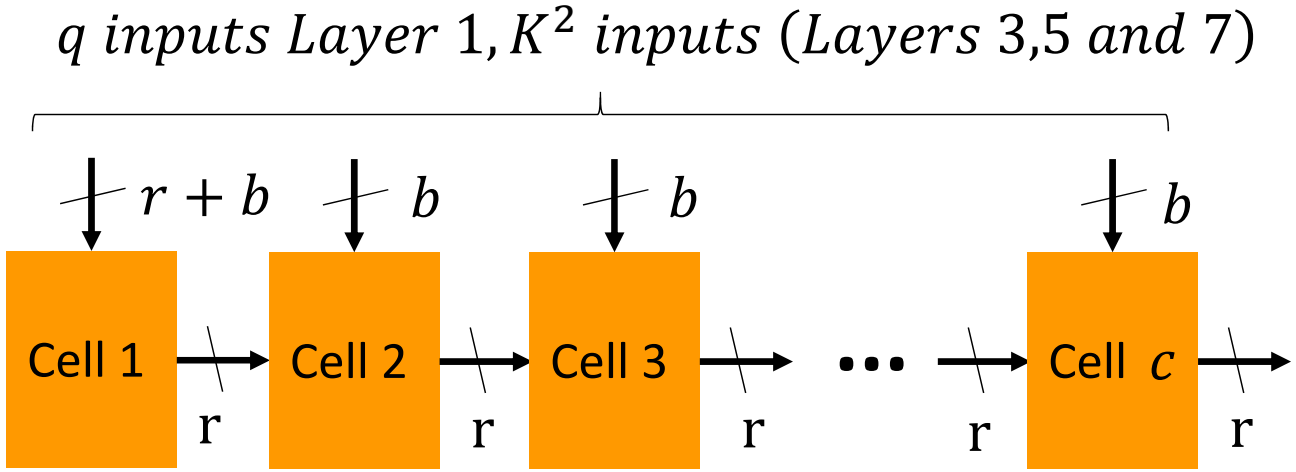
Analysis of a weight-sum function

$$u = \sum_{i=0}^{N} w_i x_i$$

- Theorem: When the output of the WS function is represented by *q*-bits, its column multiplicity is at most 2^q
- Corollary: When the output of the WS function is represented by *q*-bits, its number of rails for the LUT cascade is at most *q*

Memory size for the LUT Cascade

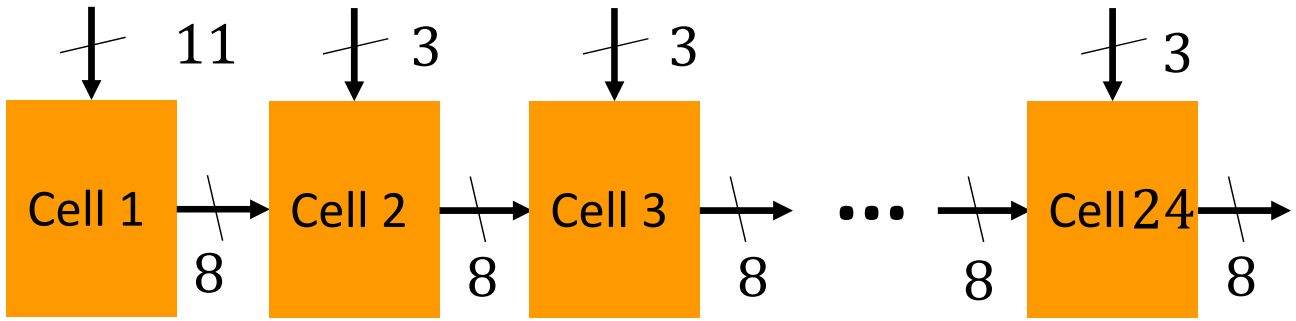
- Kernel size: K
- # of rails: $r = \lceil log_2 \mu \rceil$, μ: colum multiplicity
- External # of cell inputs: b
- # of cells: $c = \left\lceil \frac{NK^2 - r}{b} \right\rceil$, Memory size : $c2^{r+b} \cdot r$



Select adequate b for different on-chip memories

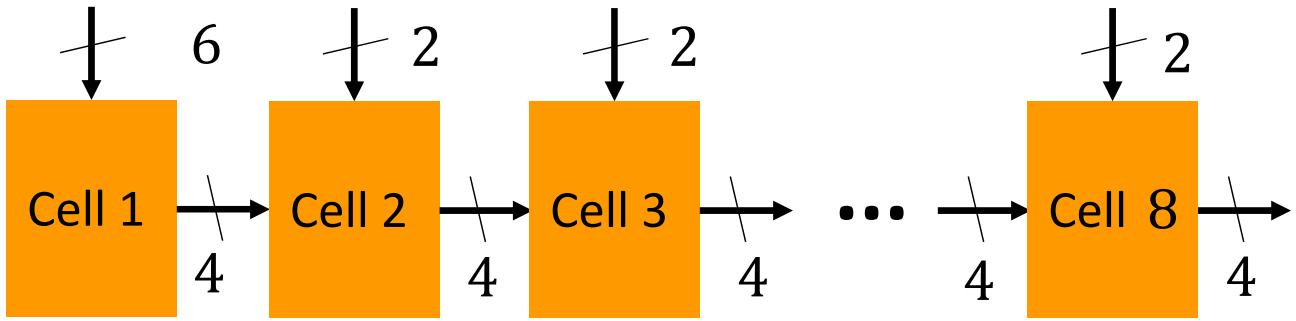
LUT Cascade for the Layer 1 by BRAMs

- Layer 1:
  - 8 bit RGB image, since K=3, # of inputs=8x3^2+8=80 ("+8" means bias value)
  - # of rails: $r = 8$
- Configure 18Kb BRAM as 11 inputs and 8 outputs
- Then, # of cells (BRAMs) = 24



LUT Cascades for Layers 3,5 and 7 by distributed RAMs

- Layers 3,5 and 7:
  - Since K=3, # of inputs=3^2x2+1=19
    ("x2" means binarized weight and input,
    "+1" means binarized bias)
  - # of rails: $r = 4$
- Configure 6-input LUTs as distributed RAMs
- Then, # of cells = 8 (32 6-input LUTs)



Prediction (Recognition) Precision

CIFAR-10 Benchmark DNN

Method	Error Rate
BinaryNet	11.40%
Binary Connect	8.27%
Gated pooling	7.62%

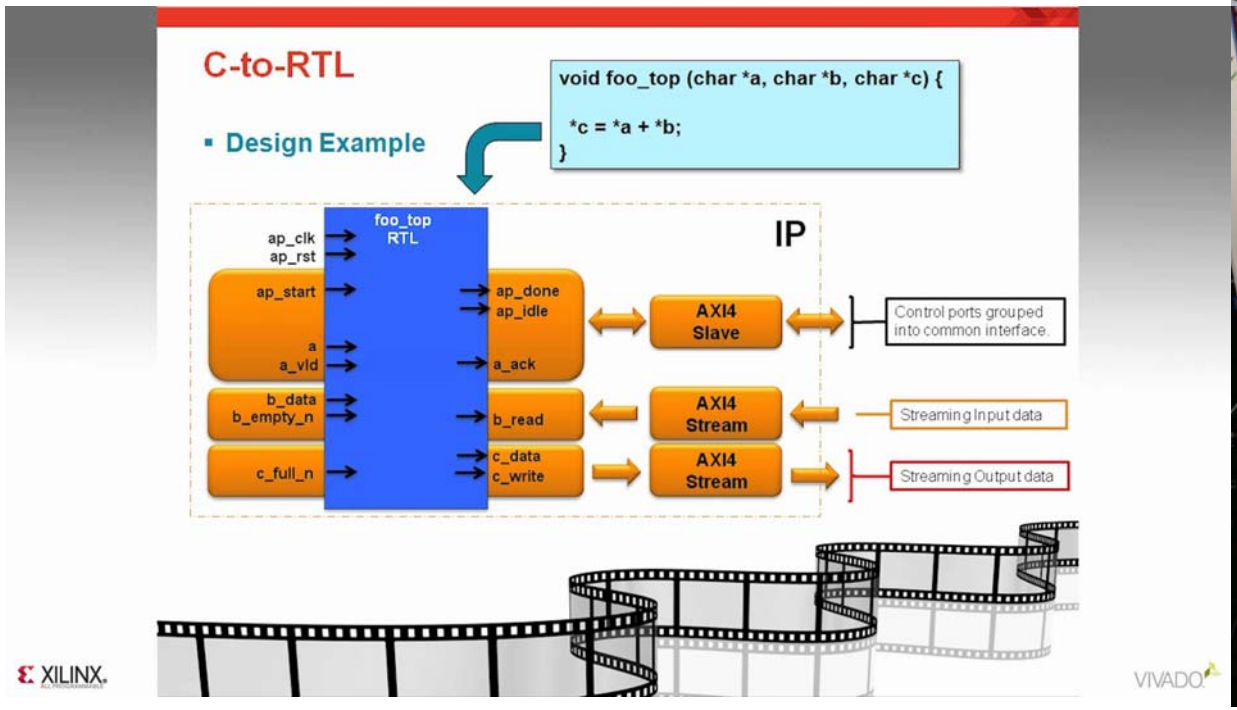
SVHN Benchmark DNN

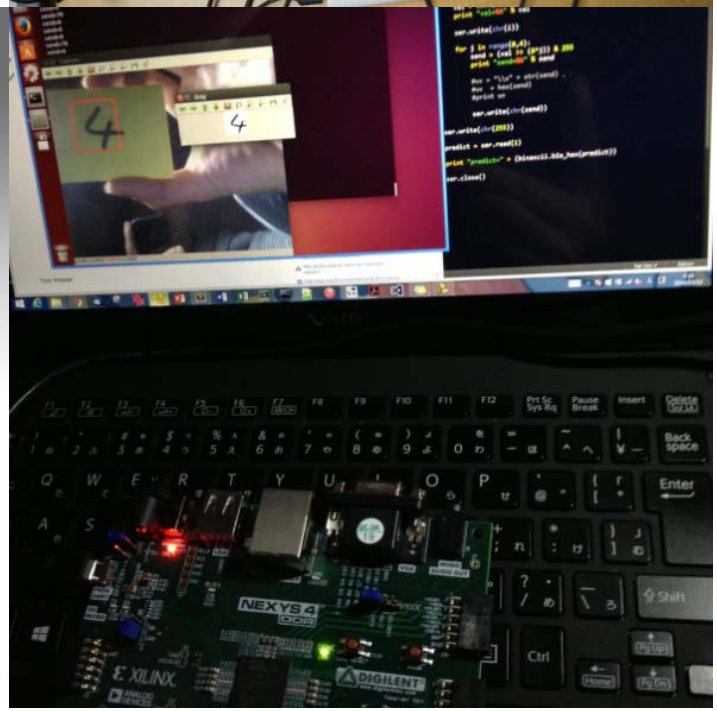
Method	Error Rate
BinaryNet	2.80%
Binary Connect	2.15%
Gated pooling	1.69%

3. Short time design using HLS for DNNs

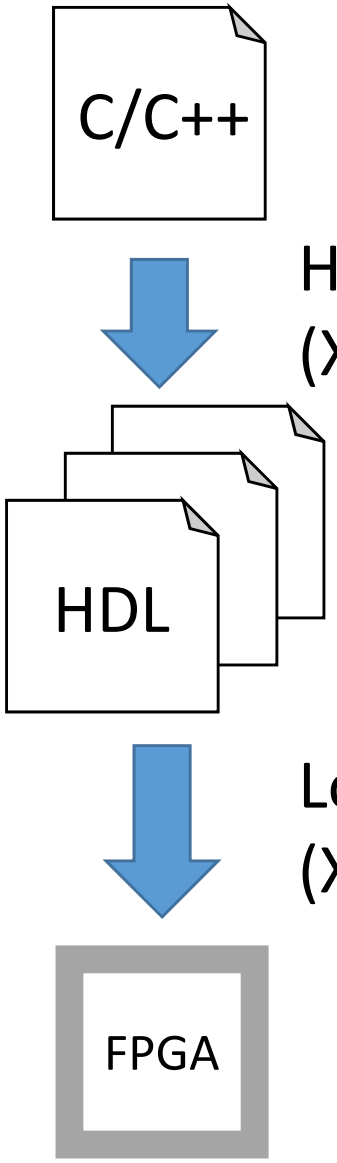
High-Level Synthesis (HLS)

- Short TAT, and high-performance
- Xilinx Inc. Vivado-HLS
- Altera Corp. Altera SDK for OpenCL





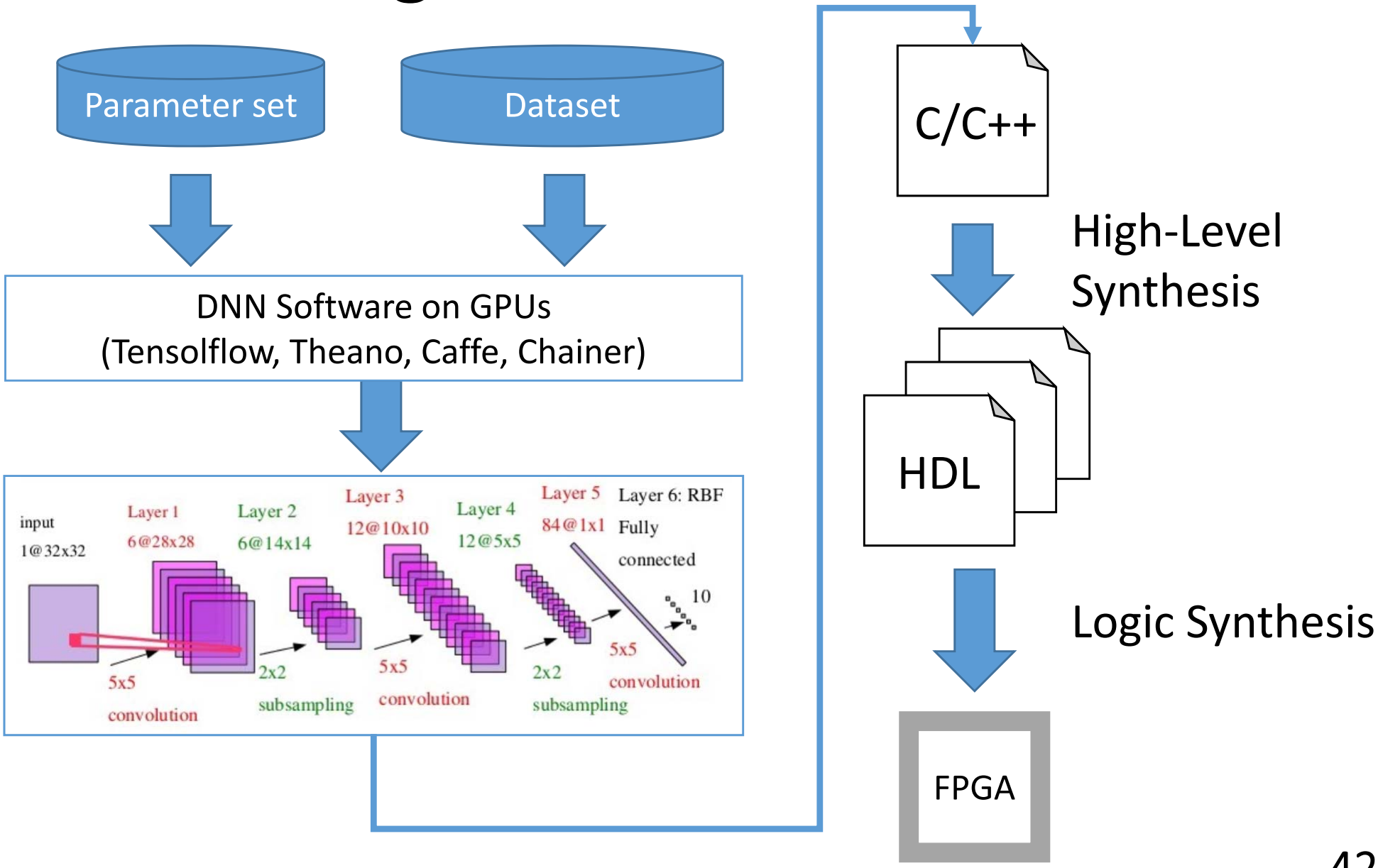
Typical FPGA Design using a HLS



High Level Synthesis
(Xilinx Vivado HLS, Altera SDK for OpenCL)

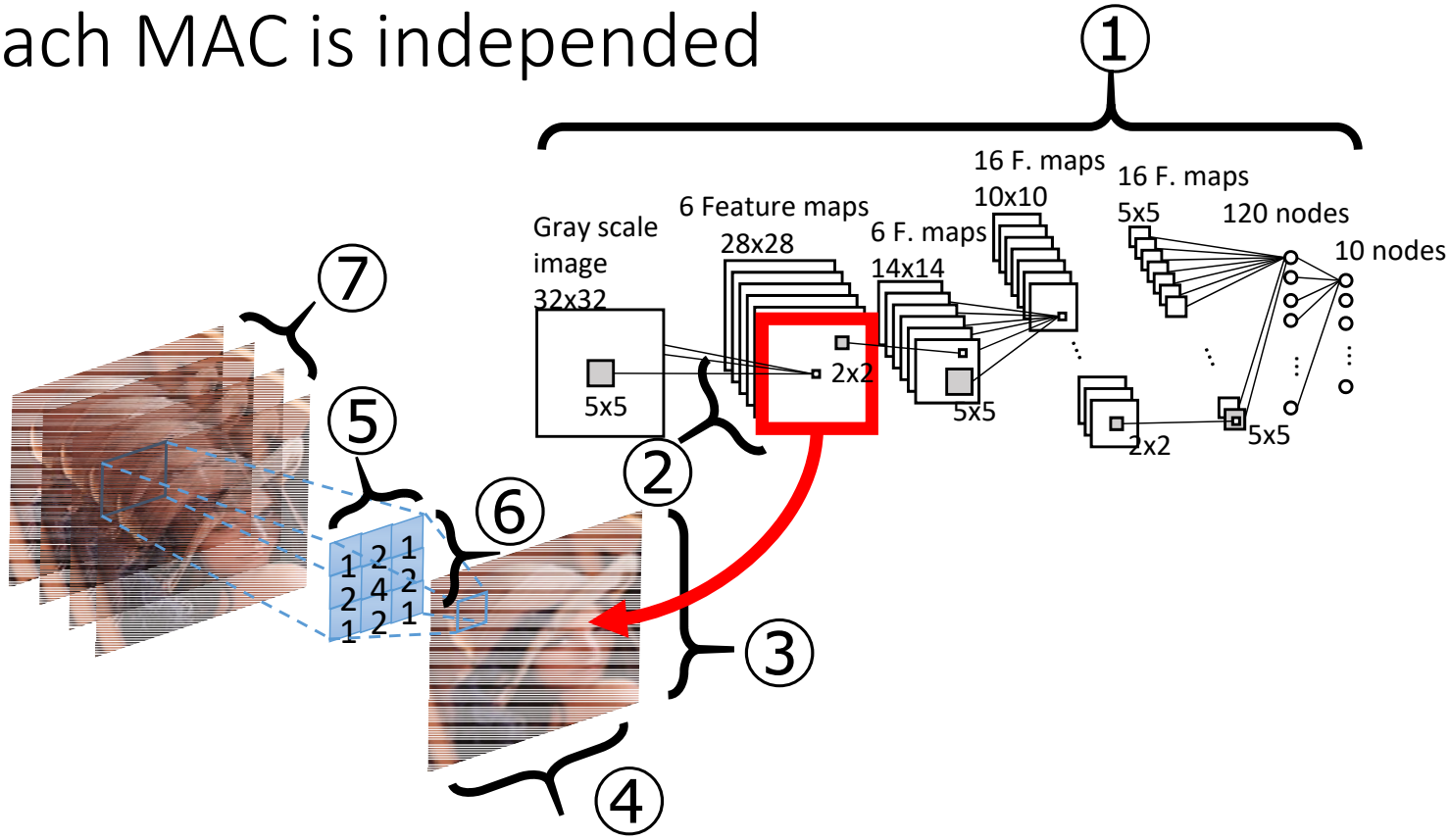
Logic Synthesis
(Xilinx Vivado, Altera Quartus II)

DNN Design for an FPGA

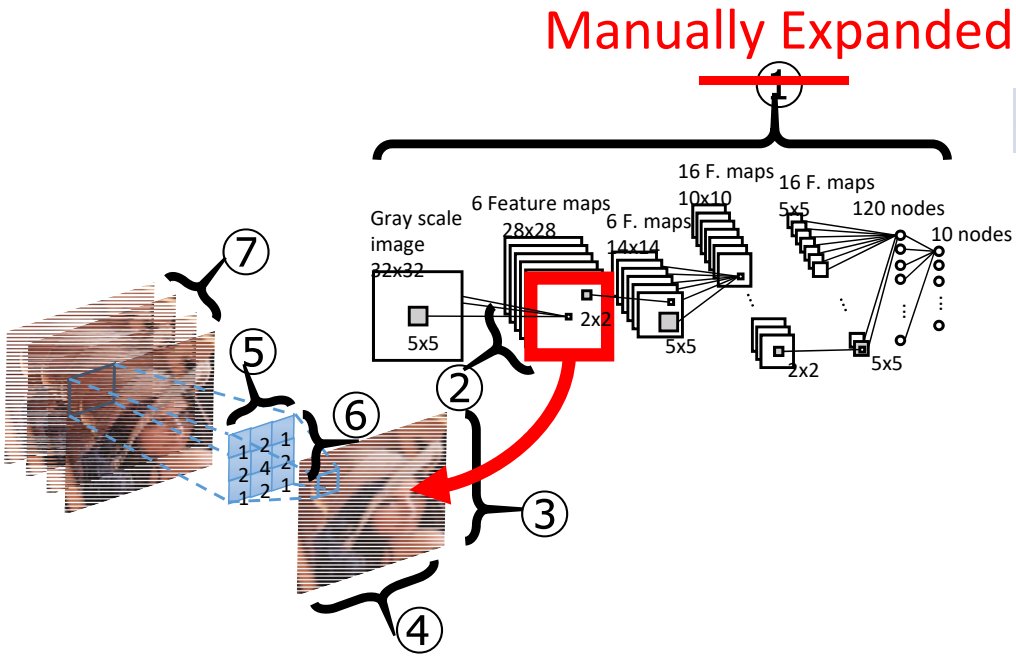


HLS Implementation for DNNs

- The typical DNN has seven “for” loops
- Each MAC is independed



Actual C/C++ code for HLS design




```
// layer 1 -------------------------------------
stride = 4;
ksiz   = 8;
n_ofeat = 32;
n_infeat = 4;
infeat_siz = 84;
//for ofeat in range(n_ofeat):
for( ofeat = 0; ofeat < n_ofeat; ofeat++){
    oy = 0;
    //for inf_y in range(0,infeat_siz - stride,stride):
    for( inf_y = 0; inf_y < infeat_siz - stride; inf_y += stride){
        ox = 0;
        //for inf_x in range(0,infeat_siz - stride,stride):
        for( inf_x = 0; inf_x < infeat_siz - stride; inf_x += stride){
            tmp = 0.0;
            //for infeat in range(n_infeat):
            for( infeat = 0; infeat < n_infeat; infeat++){
                //for ky in range(ksiz):
                for( ky = 0; ky < ksiz; ky++){
                    //for kx in range(ksiz):
                    for( kx = 0; kx < ksiz; kx++){
                        //print ("tmp=%f f=%f w=%f b=%f" % (tmp,f1[infeat,inf_y+ky,inf_x+kx],l1w[of
                        //print("feat=%d inf_y=%d ky=%d +=%d" % (infeat,inf_x,kx,inf_x+kx))
                        tmp = tmp + (f1[infeat][inf_y+ky][inf_x+kx] * l1w[ofeat][infeat][ky][kx]);
                    }
                }
            }
            //print ("tmp=%f w=%f f=%f b=%f" % (tmp,f1[infeat,inf_y+ky,inf_x+kx],l1w[ofeat,infeat,k
            //f2[ofeat,oy,ox] = tmp
            tmp = tmp  + l1b[ofeat];
            if( tmp > 0.0)
                f2[ofeat][oy][ox] = tmp;
            else
                f2[ofeat][oy][ox] = 0.0;

            ox = ox + 1;
        }
        oy = oy + 1;
    }
}
```

HLS Design using Vivado HLS

- Estimate unconstrained loop with
  #pragma HLS LOOP_TRIPCOUNT min=x max=y

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.79	1.25

Latency (clock cycles)

Summary

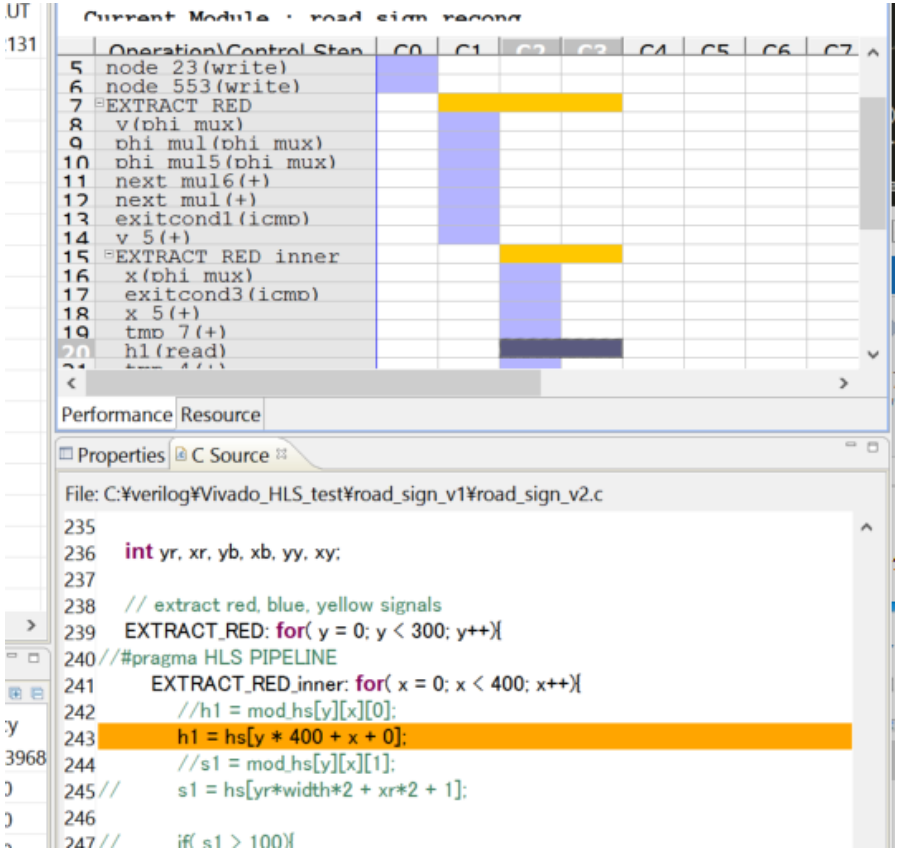
Latency		Interval		
min	max	min	max	Type
1	2483968	2	2483969	none

Detail

Instance

Loop

	Latency			Initiation Interval		
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count
- EXTRACT_RED	240600	240600	802	-	-	300
+ EXTRACT_RED_inner	800	800	2	-	-	400
- EXTRACT_BLUE	360600	360600	1202	-	-	300
+ EXTRACT_BLUE_inner	1200	1200	3	-	-	400
- EXTRACT_YELLOW	360600	360600	1202	-	-	300
+ EXTRACT_YELLOW_inner	1200	1200	3	-	-	400
- INIT_RESULT	32	32	1	-	-	32
- RECONG_RED	5776	507376	152 ~ 13352	-	-	38
+ RECONG_RED_inner	150	13350	3 ~ 267	-	-	50
++ CLR_RED	264	264	22	-	-	12



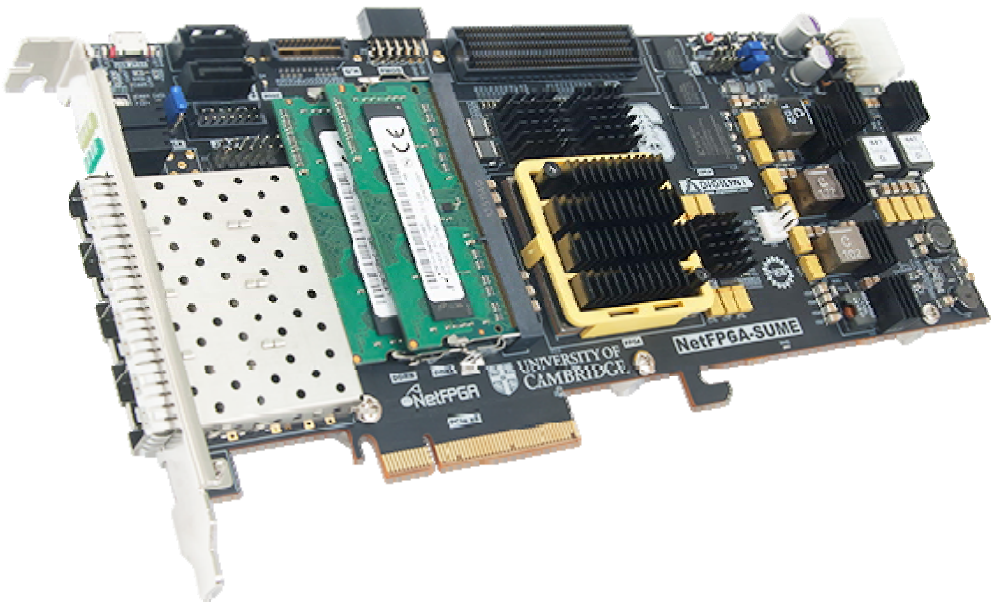
4. Implementation

Implementation of Integer DCNN

- 20bit Fixed Precision[ICLR2015]

  Pre-learned ImageNet

- FPGA board: NetFPGA Sume
  - FPGA: Virtex7 VC7V690T
    - 153,000 Slices
    - 3,600 DSP Slices
    - 2,940 18Kb BRAMs
  - 4GB DDR3SODIMMx2
- Synthesis tool: Xilinx Vivado2014.1
  - Timing constrain: 400MHz



Comparison with Other FPGA Implementations

Method	Precision	Max. Freq. [MHz]	FPGA	Performance [GOPS]	Performance per area [GOPS/ Slice x 10^{-4}]
ASAP2009	16bit fixed	115	Viretex5 LX330T	6.7	1.3
PACT2010	--- fixed	125	Viretex5 SX240T	7.0	1.9
FPL2009	48bit fixed	125	Spartax3A DSP3400	5.3	2.2
ISCA2010	48bit fixed	200	Virtex5 SX240T	16.0	4.3
ICCD2013	--- fixed	150	Virtex6 LVX240T	17.0	4.5
FPGA2015	32bit float	100	Virtex7 VX485T	61.6	8.1
FPL2015	48bit fixed	400	Virtex7 VX485T	132.2	25.2
Proposed	20bit fixed	400	Virtex7 VC7V690T	132.2	64.19

Summary

- DNN design methods for an FPGA
  - Integer DNN → Residue number system (RNS)
  - Binarized DNN → Deterministic + Batch Normalization
- With high-level synthesis design
  - Short TAT
  - Considerable HW consumption
- Comparison with other FPGA based design
- Future works
  - Acceleration for the learning
  - Detection hyper parameters → Grid search?

Exercise 3

1. (Mandatory) How do you think a “Technological singularity”? Near/Far/Never? why? and what’s happen?

Deadline is 11th, Nov., 2016

Send an E-mail to nakahara.h.ad@m.titech.ac.jp

with entitled “Exercise 3 (your name)”