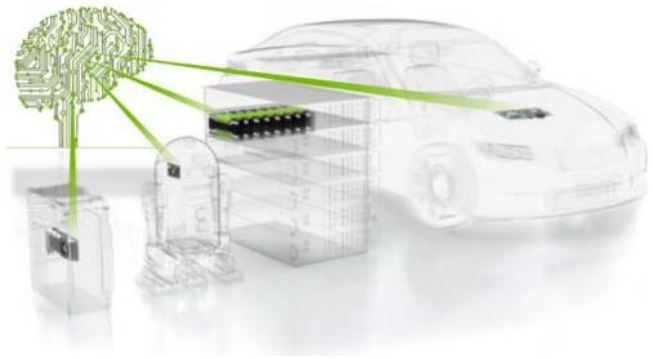
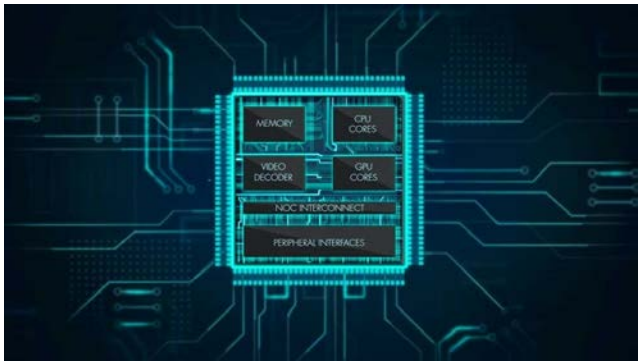


Digital system (SoC) design for Deep Neural Networks (AI accelerator design)



Author : **Hyun Kim**

Affiliation : Seoul National University of Science and Technology
Electrical and Information Engineering

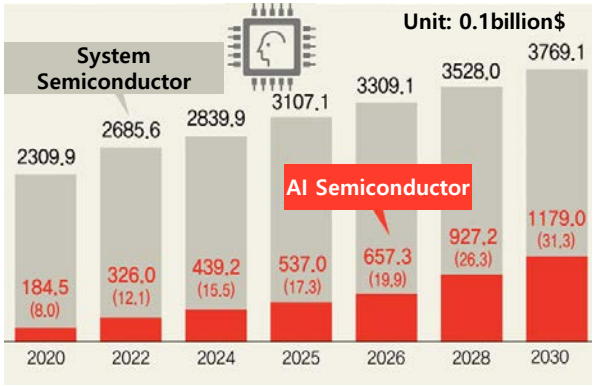
Position : Assistant Professor

Contact : hyunkim@seoultech.ac.kr / 010-9600-5427
idsl.seoultech.ac.kr

Necessity of AI Semiconductor

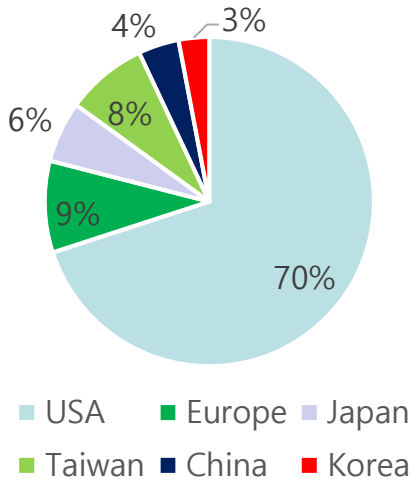
Architecture-Level

System/AI semiconductor market size



Korean government's high interest in AI

System/AI Semiconductor market share



Necessity and importance of research and manpower training for AI semiconductors

- **Prospect**: In next 5-10 years, the system semiconductor market including AI semiconductors is expected to grow up to 3 x of the memory semiconductor market, and the government has also high interest in AI semiconductors
- **Current Status**: Significantly lower market share of system semiconductors compared to memory semiconductors (about 70% vs. 3%) due to the small/medium company-oriented development structure
- **Solution**: Securing stand-alone technologies through industry-academia cooperation research + Producing experts who have a comprehensive understanding of AI algorithms and system semiconductor design technology

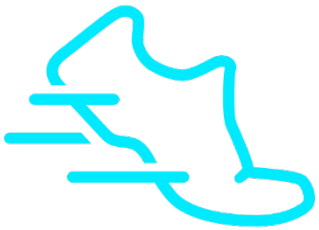
- 모바일 자가 학습 가능 재귀 뉴럴 네트워크 프로세서 기술 개발, 2020.04~2024.12, 과학기술정보통신부 지원
- 2,000 TFLOPS급 서버 인공지능 딥러닝 프로세서 및 모듈 개발, 2020.04~2027.12, 과학기술정보통신부 지원
- 인공지능 반도체 융합 전문 인력 육성 사업, 2020.04~2025.12, 과학기술정보통신부 지원

Key Issue of Mobile AI Accelerators

Architecture-Level

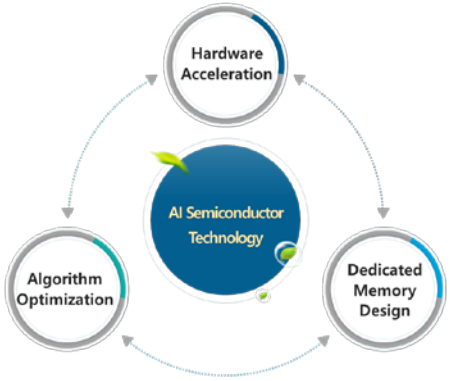
- AI accelerators based on digital system design are expected as a solution to these challenges in AI
- Three main goals of AI accelerators: High accuracy + High speed (throughput) + Low power

Accuracy ↑



Speed ↑

Power ↓



- Necessity of Architecture-level approach: Hardware acceleration of the optimized neural networks with parallelization and optimization enables fast processing with low-power consumption

- Most effective way to accelerate AI
  - GPU: Various development frameworks for AI are supported, but GPU suffers from size & cost & power problems!
  - FPGA/ASIC: Expertise in implementation is required, but this approach has advantages of small size, high cost-efficiency, high power-efficiency, and is easy to apply techniques to increase hardware utilization

- Results of processing the same network on FPGAs & GPUs

YOLOv2	GPU (GTX Titan X)	FPGA (Virtex-7 VC707)
Power	170W	18.3W

IDSL INTELLIGENT DIGITAL SYSTEMS DESIGN LAB

SEOULTECH

DrivePX2 and FPGA

Various Platforms for Autonomous Driving

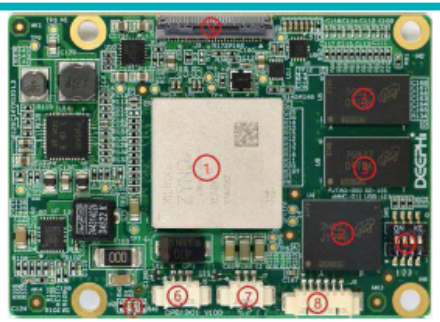
- Requires a low-power device because it must operate from a limited resource of vehicles
- Nvidia launches autonomous embedded board → still high power consumption
- Recently, FPGA is attracting attention due to high efficiency in terms of the trade-off between power and performance

	Titan XP	Drive PX2 (Tesla)	Jetson TX2 (Embedded)	AGX Xavier (Embedded)	FPGA (Virtex UltraScale+VCU 1525)
Manufacturer	Nvidia	Nvidia	Nvidia	Nvidia	Xilinx
Performance	12.15 TFLOPs	8-10 TFLOPs	1.5 TFLOPs	11 TFLOPs	**21 TOPs** (16 INT)
Power	250W	125W	< 15W	< 30W	**< 25W**
Processor	1x CPU(Xeon) + 1x GPU(1080Ti)	2x CPU + 1x GPU	1x CPU (TegraX2) + 1x GPU (Pascal)	1x CPU (Xavier) + 1x GPU (Volta) + 1x TPU	FPGA
Price	$1,500 (GPU only)	$15,000	$599	$1,299	$5,995

FPGA Implementation Results - DEEPHi

Benchmark comparison of networks

- **Platform** ÆMPSoC ZU2EG
- **Size:** 50*70 mm
- **DPU** ÆB1152
- **Peak perf.:** 576GOPS (500MHz)



	GOP	200MHz		250MHz		300MHz		350MHz		400MHz	
		FPS	Power	FPS	Power	FPS	Power	FPS	Power	FPS	Power
ResNet-50	7.7	15.3	5.3	18.38	5.55	22.3	5.9	24.87	6.32	27	6.6
ResNet-50[1)]	3.8	23.6	5.29	28.24	5.54	33.5	5.89	37.8	6.2	39.6	6.52
GoogLeNet	3.2	36.5	5.26	45.90	5.54	53.8	5.97	61.7	6.43	68.2	6.74
GoogLeNet[1)]	1.6	62	5.24	77.11	5.59	93	6.03	109.4	6.41	116	6.72
SSD	117	1.63	5.4	2.03	5.71	2.44	6.22	2.835	6.52	3.23	7.05
SSD [1)]	11.6	13.2	5.47	16.35	5.77	19.34	6.18	22.43	6.65	25.3	7.05

1) deep compression technique is applied on these networks

Reference : Deephi

Recent Trend of FPGA & GPU

Fig. NVIDIA (GPU) stock price trend in the early stages of AI semiconductor commercialization

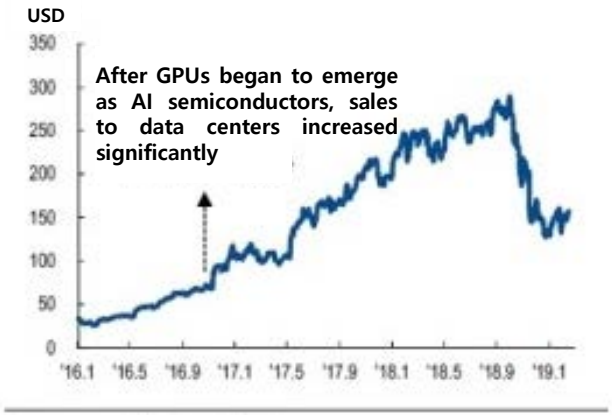
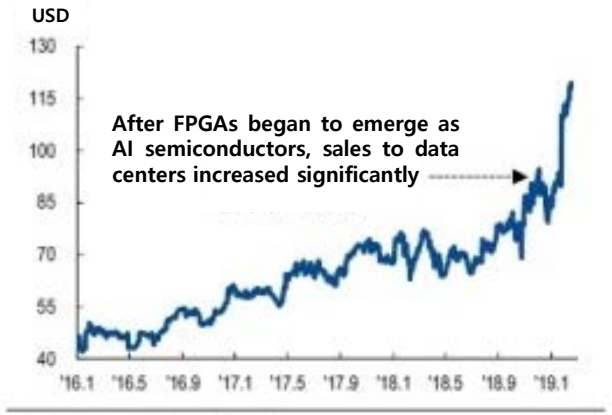
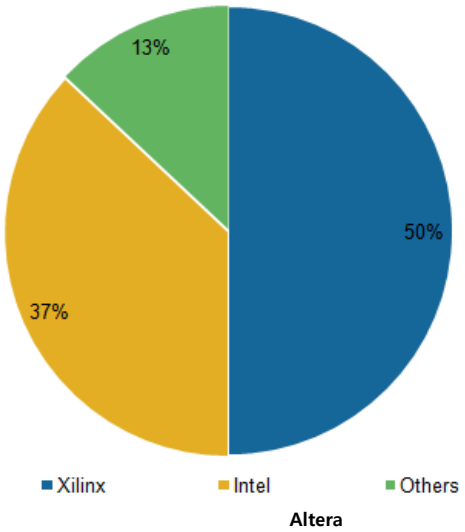


Fig. XILINX (FPGA) stock price trend in the early stages of AI semiconductor commercialization

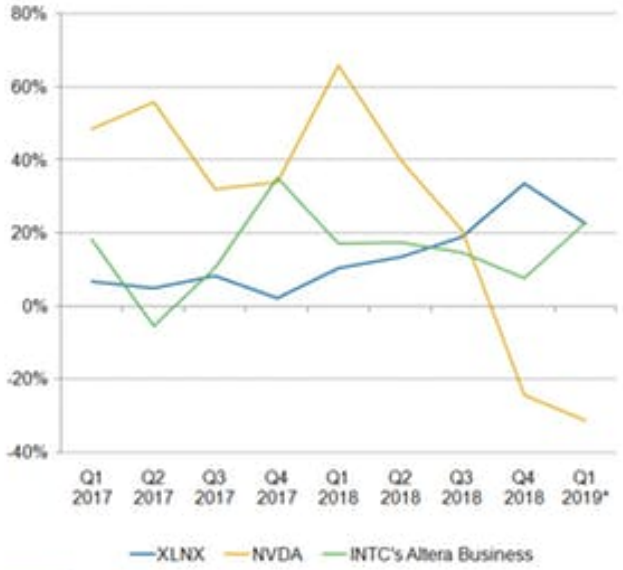


Programmable Logic Devices' Vendors by Revenue in Calendar 2015



Source: IHS

YoY Revenue Growth Rate of Xilinx and Peers from Calendar Q1 2017 to Q1 2019*



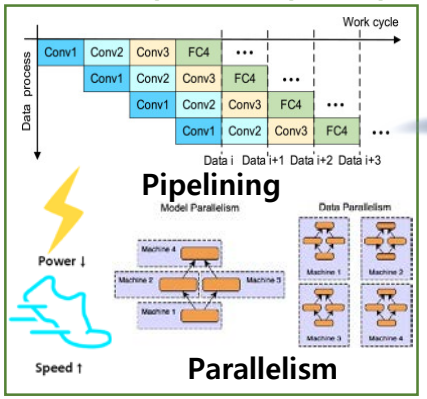
Overview of AI Accelerators

Architecture-Level

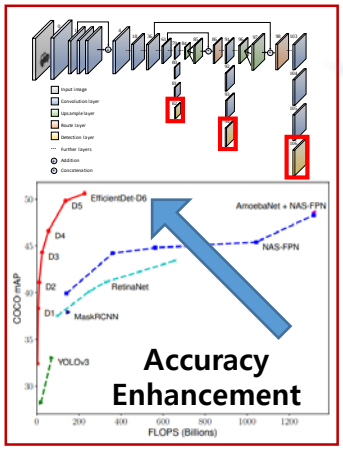
Mobile Characteristic

- Mostly Inference
- Embedded GPU → FPGA/ASIC
- Low-Precision: 2b/4b/8b (INT)
- Sparse network
- Application-specific accelerator design

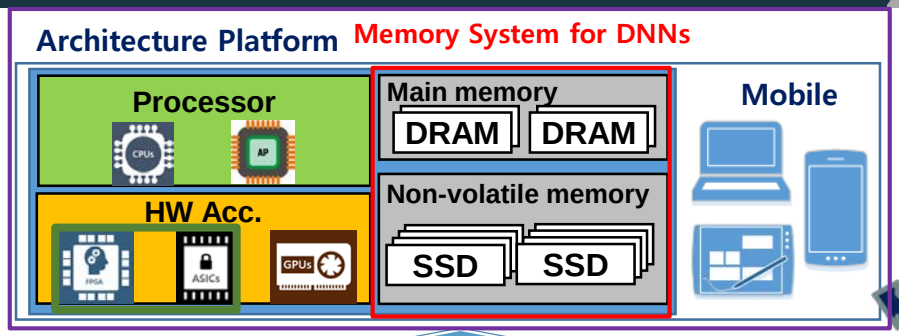
HW-based low complexity schemes for low-power & speed-up



Performance enhancement schemes

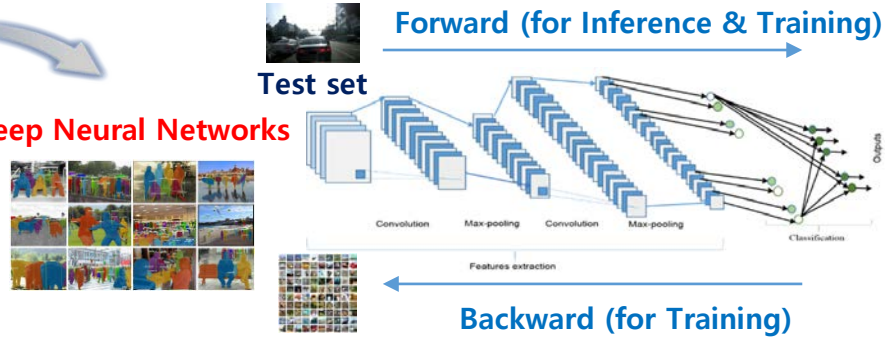


Architecture Platform — Memory System for DNNs

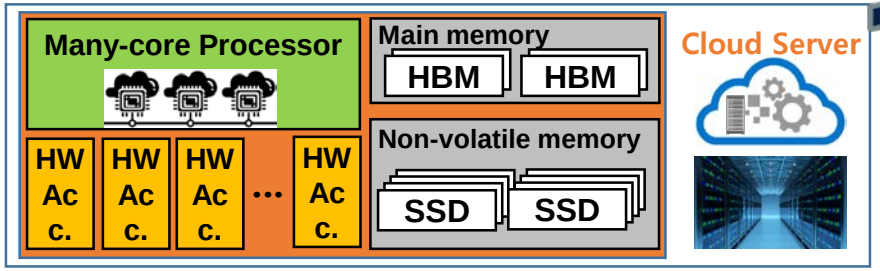


Focus on designing the application-specific AI accelerators for mobile devices

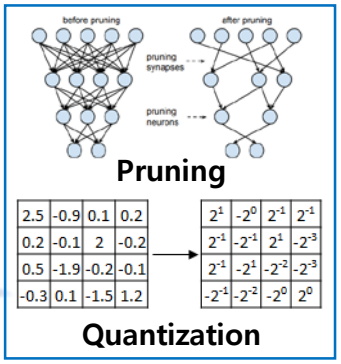
Deep Neural Networks



Architecture Platform



SW-based low complexity schemes for low-power & speed-up



Weight Update

Autonomous Driving

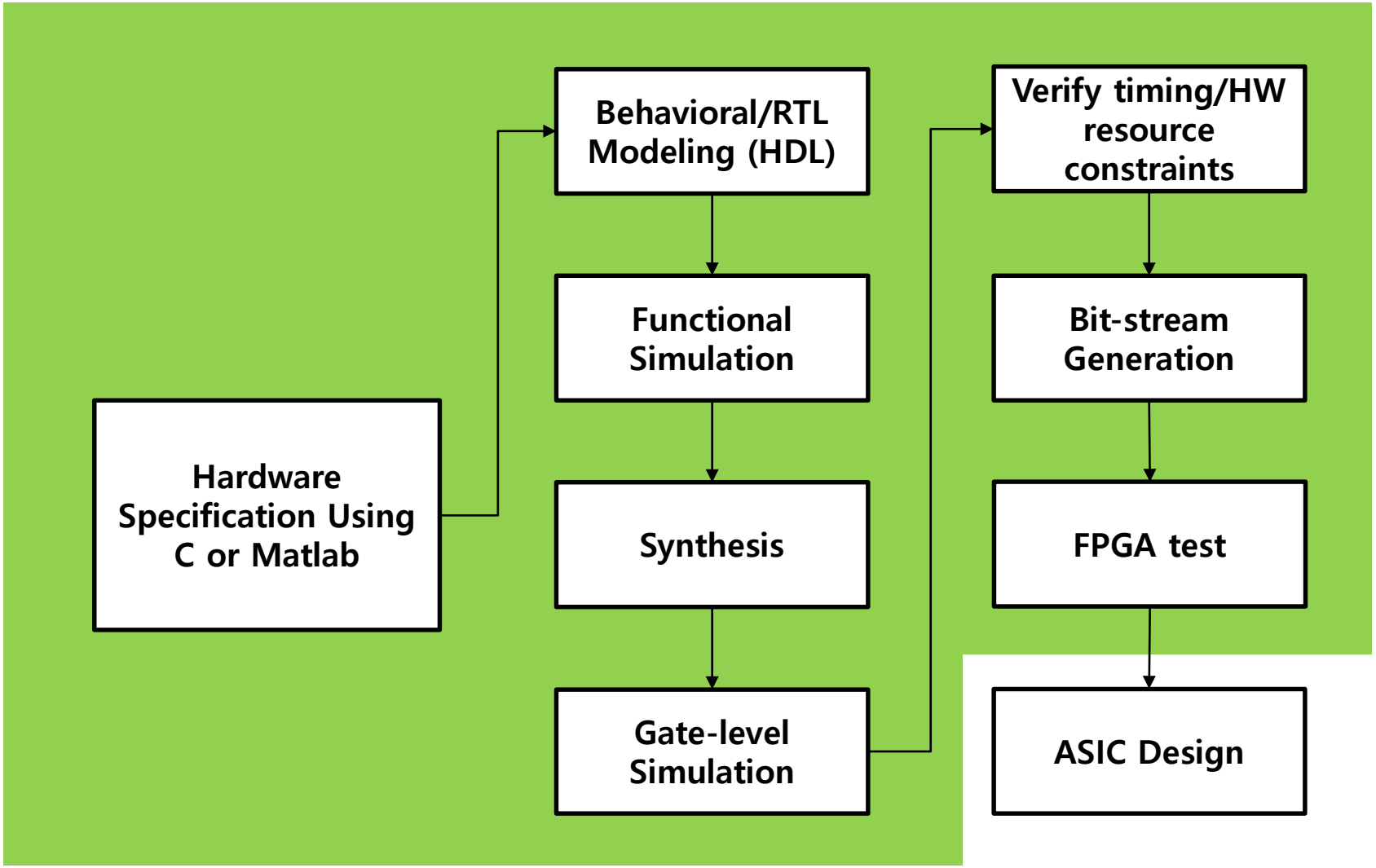
Apply to target applications

Video Surveillance as a Service (VSaaS)

Server Characteristic

- Training is dominant
- Still, GPU is dominant
- Precision: 8b/16b/32b (FP)
- Dense network
- General-purpose accelerator design

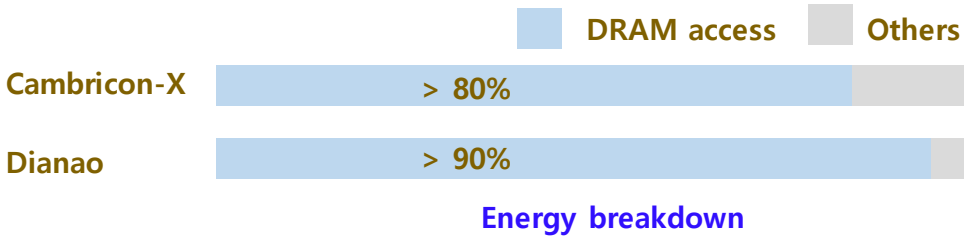
Front-End Verification



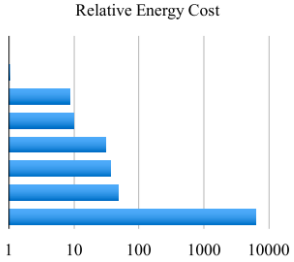
Challenges Designing CNN HW Accelerator

Motivation

- CNNs require **numerous computations** and **external memory accesses**
- **Why should we minimize off-chip memory access?**
  - Energy consumed by off-chip access is much larger than on-chip access/arithmetic ops
  - SOTA accelerators such as Dianao [1], Cambricon-X [2]: DRAM energy accounts for > 80% of total
- **Why should we reduce buffer size?**
  - In ASIC design, the large buffer size leads to the large chip size → increase hardware cost



Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**



General Solution

- Quantize networks with low precision to minimize computations and on-chip SRAM size
- Data-path optimization to minimize the off-chip access
- Reuse the modules and apply pipelining for increasing HW utilization

[1] "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ASPLOS* 2014.
[2] "Cambricon-x: An accelerator for sparse neural networks," *MICRO* 2016.

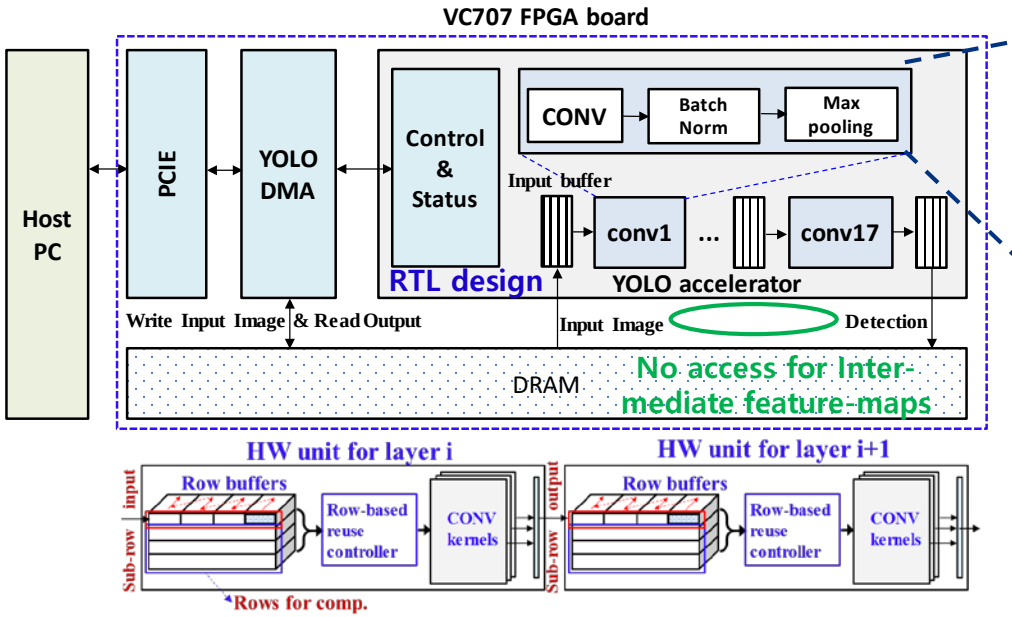
FPGA Design for YOLO Object Detectors (1)

Architecture-Level

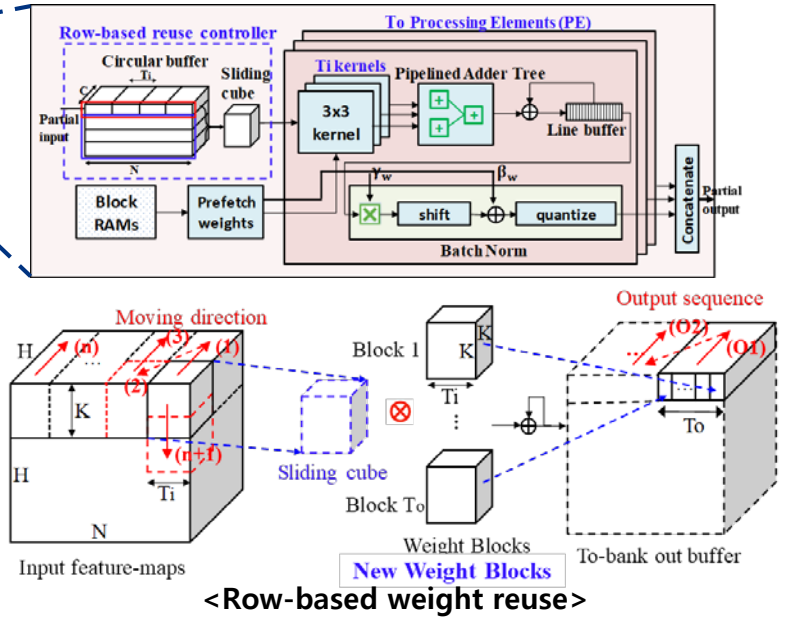
- **Goal**: Implement **high throughput** and **low-power (low-cost) HW accelerators**
- **Motivation**: CNNs require significant **external/internal memory accesses and capacity** as well as large amounts of computation
  - Energy and latency caused by off-chip accesses are much more critical than on-chip access/arithmetic operation
  - Large buffer size leads to an increase in hardware cost, and internal memory allowed in FPGAs is also limited
- **Solution/Contribution**

W: weight, A: activation

  - **HW-Centric Quantization**: Binary W + Flexible low-bit A(3b to 6b) to store the entire parameter in **on-chip BRAMs**
  - **Streaming HW architecture** with **data-path optimization**: By using row-based weight reuse, intermediate data are reused without off-chip access → **Completely remove off-chip access** for intermediate feature-maps
  - Enhancing **hardware utilization**: Pipelined computation & Pipelined batch processing



<Pipelined structure between each layer>

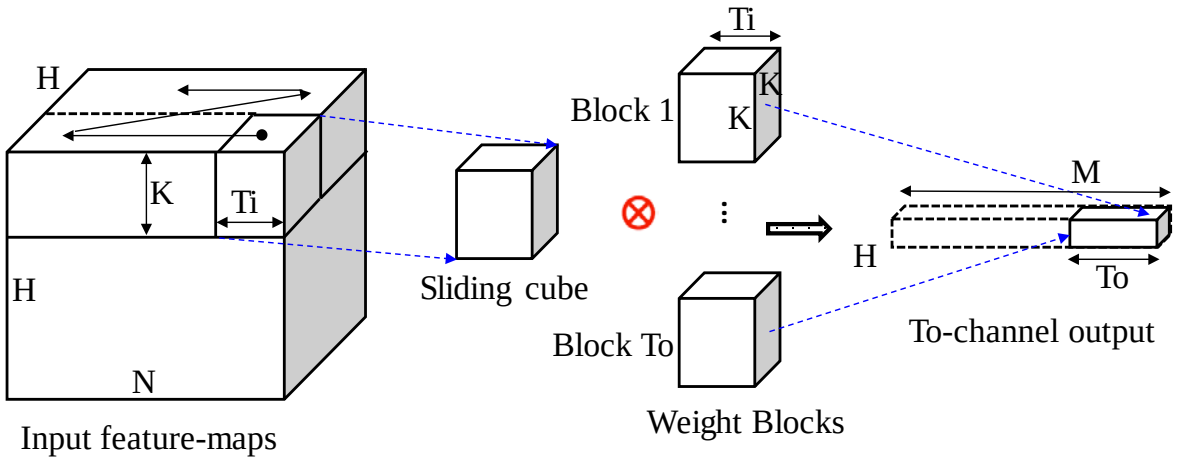


<Row-based weight reuse>

- **Related Papers**
  - **"A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection," IEEE Trans. VLSI Syst., Aug. 2019 (Google Scholar Citation: 101)**

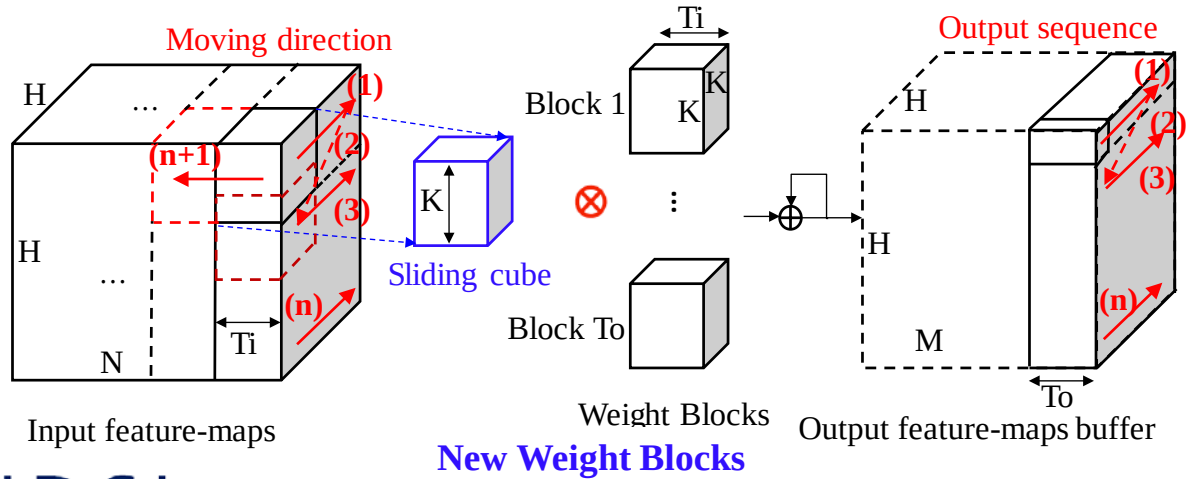
FPGA Design for YOLO Object Detectors (2)

- Data reuse
  - Basically, block-based computations are used to achieve the trade-off between HW resource and performance
  - Previous Design (1): No weight reuse



- The input sliding cube moves from the beginning toward the end of the channel dimension → Each sliding cube is convolved with new weight blocks (No reuse)
- This has the best locality of the partial sum → It does not require a temporary buffer for the accumulation

  - Previous Design (2): Frame-based weight reuse



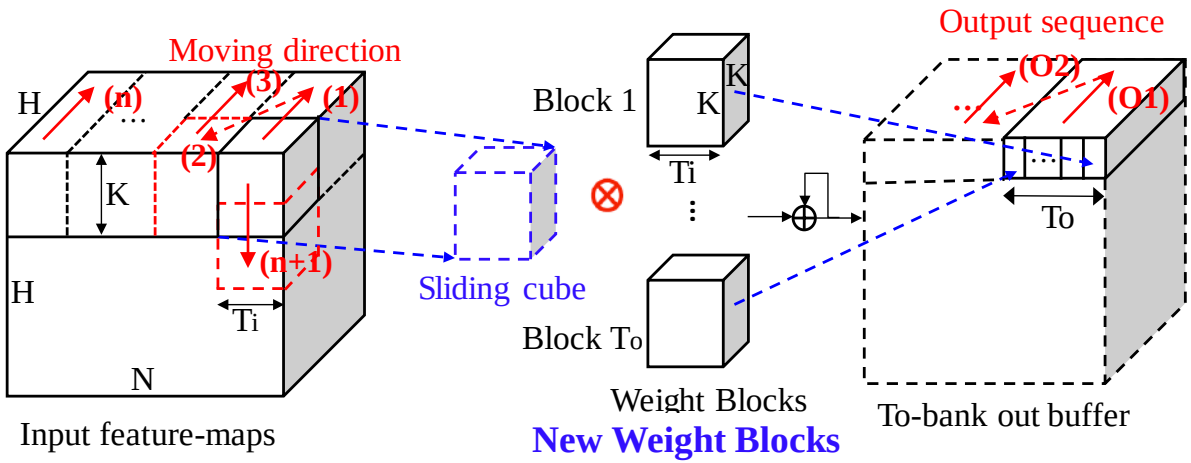
- Weight reuse scheme in [1], [2]
- It maximizes the weight reuse
- Drawbacks:
  - Inputs/outputs are accessed multiple times
  - To reduce off-chip access → Large buffer size to store inputs/outputs

[1] "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," DATE 2018.
[2] "DNA: Deep Convolutional Neural Network Architecture with Reconfigurable Computation Patterns," TVLSI 2018.

FPGA Design for YOLO Object Detectors (3)

Data reuse

Proposed Design: Row-based weight reuse



✓Advantages
- ✓Inputs are loaded once from off-chip memory in a row-by-row manner
  - Input load time is hided by overlapping with computation
  - Reused for a row size
- ✓Small buffer for inputs/outputs

✓Disadvantages
- ✓Weights are read H times
- ✓Requires large weight buffer

✓Efficient for layers with large feature maps and small weights

- ✓ How sliding cube moves:
  - ✓ (1) slide through a row pass (weight blocks are reused for a row pass)
  - ✓ (2) shift next Ti input channels
  - ✓ (3) next row pass (new weight blocks reuse)
  - ✓ (4) last row pass
  - ✓ Repeat (1)-(4) for next sub-row output (O2) until last sub-row output
  - ✓ (5) move down one row & go to (1)
- ✓ Outputs sequence is same as that of inputs

Features	No weight reuse	Frame-based weight reuse	Row-based weight reuse
Input buffer size	$(K+1)\times N\times H\times Q_A$	$H^2\times N\times Q_A$	$(K+1)\times N\times H\times Q_A$
Output buffer size	0	$T_o\times H^2\times Q_S$	$T_o\times H\times Q_S$
Weight read (times)	H^2	1	H
Weight reuse (times)	1	H^2	H
Efficient for	-	Deep layers	Shallow layers

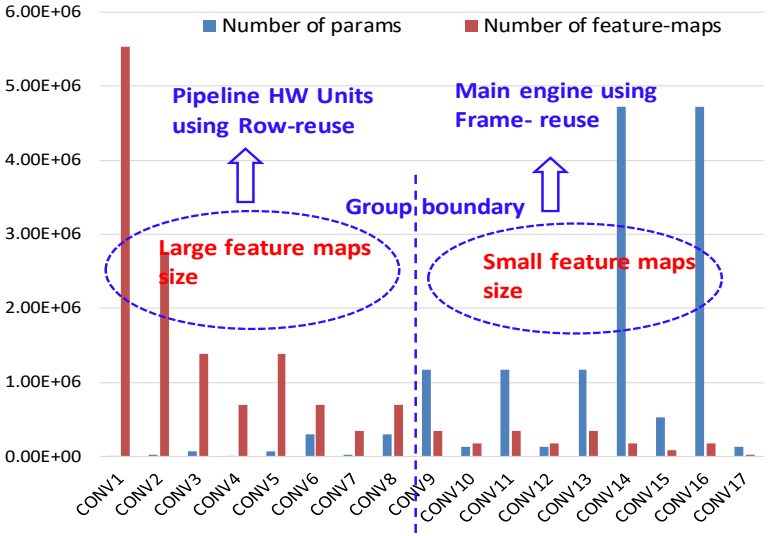
QA: input bit-width, QS: Psum bit-width

[1] "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," DATE 2018.
[2] "DNA: Deep Convolutional Neural Network Architecture with Reconfigurable Computation Patterns," TVLSI 2018.

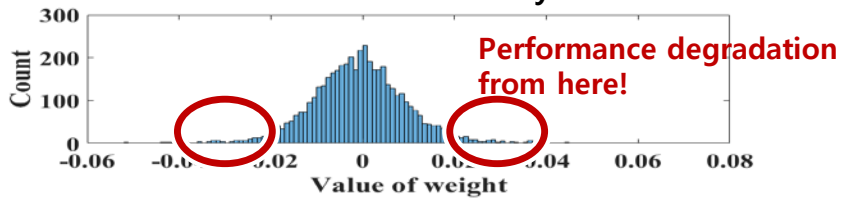
FPGA Design for YOLO Object Detectors (4)

Architecture-Level

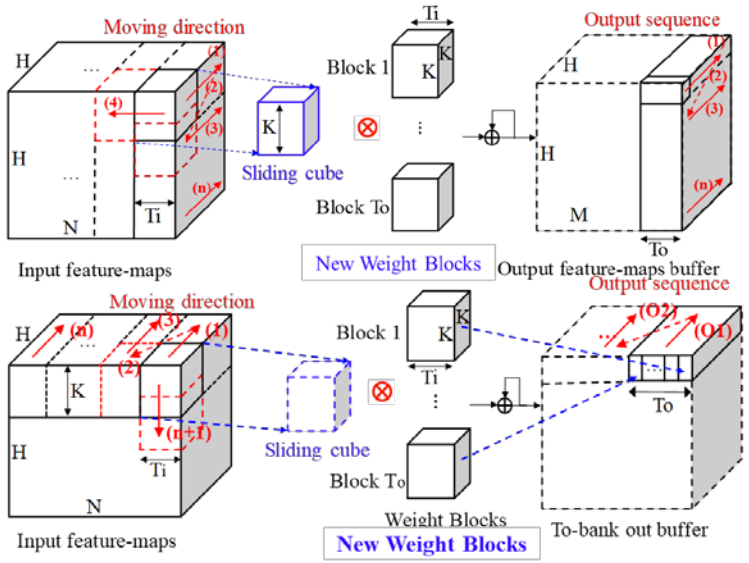
- **Goal**: Achieve **optimal hardware design** by **reflecting the characteristics of the layers** and **compensating for the loss of accuracy**
- **Motivation**: Previous design uses a common hardware organization scheme for all the layers (=**not layer-specific**), and causes a **significant accuracy degradation** due to binary W quantization
  - 1st: Characteristics of each layer are different, and the size of required buffers also varies according to the weight reuse method
  - 2nd: Most weights have small absolute values, whereas a few weights have large absolute values



<Characteristics of each layer>



<Weight distribution of YOLO>



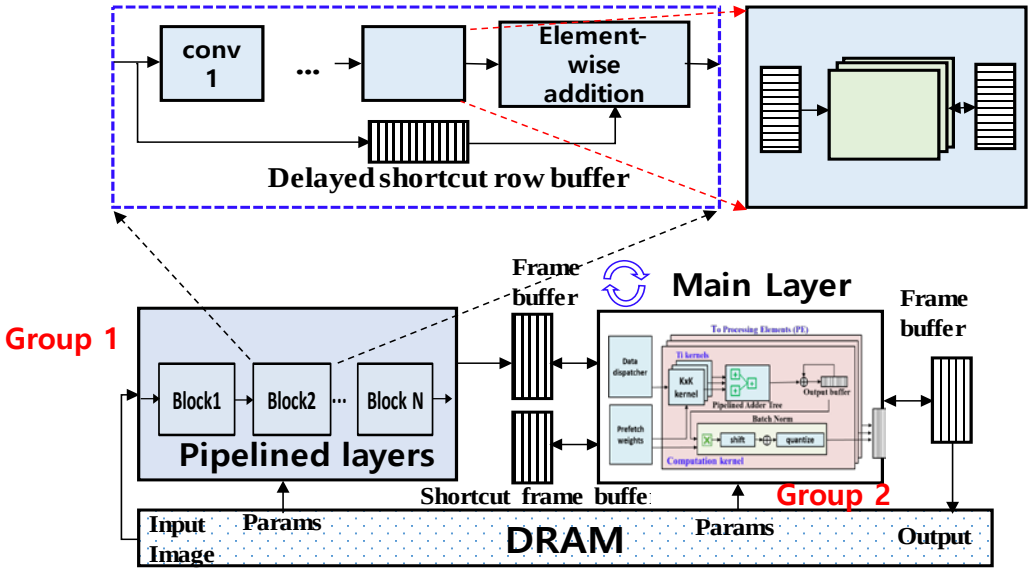
Features	Frame-based weight reuse	Row-based weight reuse
Input buffer size	$H^2 \times N \times Q_A$	$(K+1) \times N \times H \times Q_A$
Output buffer size	$T_o \times H^2 \times Q_S$	$T_o \times H \times Q_S$
Weight buffer size	$T_i \times T_o \times K^2$	$M \times N \times K^2$
Data read (times)	1	1
Efficient for	Deep layers	Front layers

- **Related Papers**
  - **"Layer-specific Optimization for Mixed Data Flow with Mixed Precision in FPGA Design for CNN-based Object Detectors," IEEE Trans. CSVT, Jun. 2021**

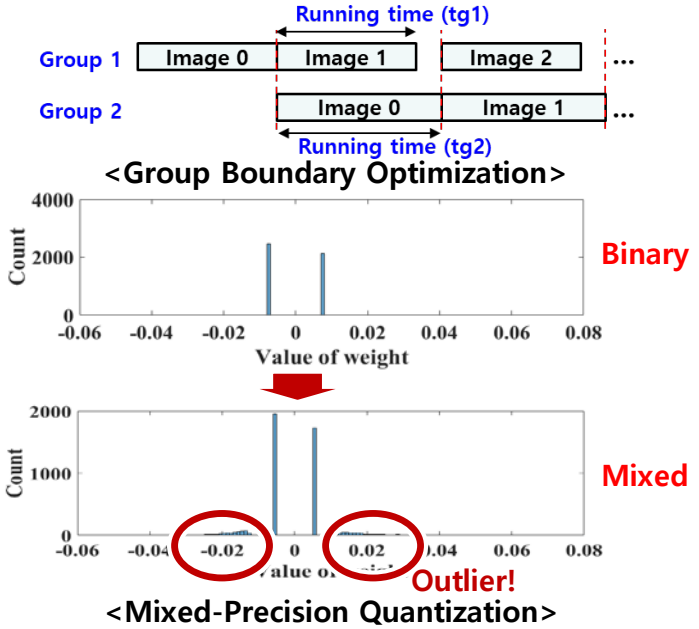
FPGA Design for YOLO Object Detectors (5)

Architecture-Level

- **Goal**: Achieve **optimal hardware design** by **reflecting the characteristics of the layers** and **compensating for the loss of accuracy**
- **Motivation**: Previous design uses a common hardware organization scheme for all the layers (=**not layer-specific**), and causes a **significant accuracy degradation** due to binary W quantization
- **Solution/Contribution**
  - **Mixed data flow**: Row-based and frame-based W reuse schemes are applied to front and deep layers, respectively
    - Front layers (Group 1): Processed in a **pipelined** HW structure using row-based W reuse
    - Deep layers (Group 2): **Sequentially** operated in a single processing module using frame-based W reuse
    - Optimization: Determine the **group boundary** to maximize HW utilization → |tg1–tg2| should be minimized
  - **Mixed (Outlier-aware) precision quantization**: Dense 1-bit for small weights + Sparse 8-bit for large weights



<Mixed-data flow HW structure>



<Mixed-Precision Quantization>

- Related Papers
  - "Layer-specific Optimization for Mixed Data Flow with Mixed Precision in FPGA Design for CNN-based Object Detectors," IEEE Trans. CSVT, Jun. 2021

FPGA Design for YOLO Object Detectors (6)

Architecture-Level

	Sim-YOLO v2 on GPU [1]	Tincy YOLO [2]	Lightweight YOLO v2 [3]	Proposed (Sim-YOLO v2)	Proposed (Layer Opt.)
Platform	GTX Titan X (16nm)	Zynq Ultrascale+ (16 nm)	Zynq Ultrascale+ (16 nm)	Virtex-7 VC707 (28 nm)	Virtex-7 VC707 (28 nm)
Frequency	1 GHz	N/A	300 MHz	200 MHz	200 MHz
BRAMs (18 Kb)	N/A	N/A	1706	**1144**	**1245**
DSPs	N/A	N/A	377	272	829
LUTs - FFs	N/A	N/A	135K – 370K	155K – 115K	245K – 117K
CNN Size (GOP)	22.73	4.5	14.97	17.18	17.18
Precision (W, A)(**)	(32, 32)	(1, 3)	(1-32, 1-32)	(1, 3-6)	(Mixed 1-8, 3-6)
Image Size	416×416	416×416	224×224	416×416	416×416
Frame rate	88	16	40.81	109.3	109.3
Accuracy (mAP) (%)	72.08	48.5	67.6	64.16	**71.13**
Throughput (GOPS)	1512	72	610.9	**1877**	**1877**
Efficiency (GOPS/kLUT)	N/A	N/A	4.52	**12.11**	**7.66**
Power (W)	170	6	N/A	18.29	N/A
Power efficiency (GOP/s/W)	8.89	12	N/A	**102.62**	N/A

- The proposed design (Sim-YOLOv2) is **1.2x faster & 11.5x more power efficient** than GPU
- Compared to lightweight YOLO-v2, 3.1x faster (w/ higher resolution) at 1.5x slower frequency
- Layer Opt. shows **significant improvement in mAP** despite similar HW resources with Sim-YOLOv2

[1] J. Redmon, A. Farhadi, "YOLO9000: Better, Faster, Stronger," [Online]. Available: arxiv.org/abs/1512.03385. arxiv.org/abs/1612.08242.
[2] T. B. PreuBer et al. "Inference of Quantized Neural Networks on Heterogeneous All-Programmable Devices," in Proc. IEEE DATE Conf., 2018.
[3] H. Nakahara et al. "A lightweight YOLO v2: A Binaized CNN with Parallel Support Vector Regression for an FPGA," in Proc. ACM Symp. FPGA, 2018.

FPGA Design for YOLO Object Detectors (7)

- **Live Demo (FPGA Evaluation) of YOLO Accelerator for streaming inputs**

FPGA Design for YOLO Object Detectors (8)

- FPGA Evaluation of YOLO Accelerator for ADAS (Blackbox HD inputs)

Self-learnable AI Accelerator for Mobile Devices (1)

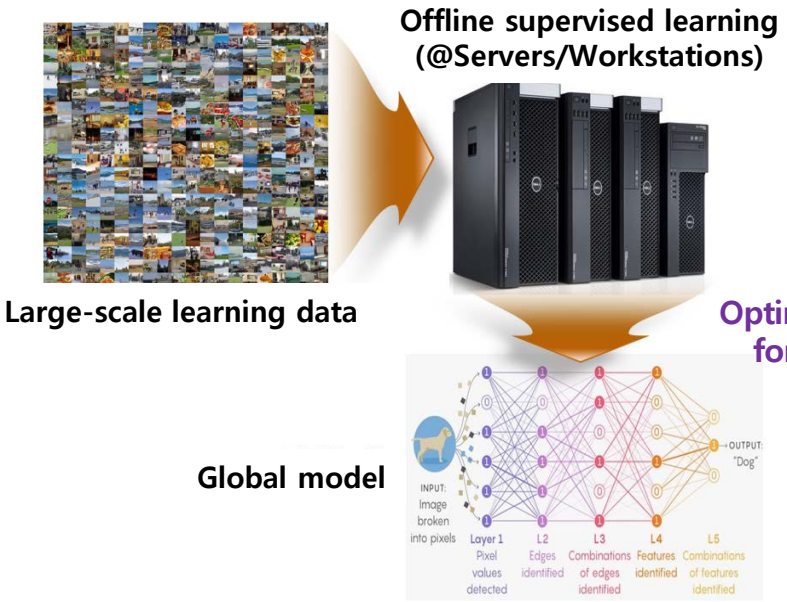
◈ **Motivation:** Increasing demand for '**Mobile Self-learning**' for optimal AI models in user's individual environments

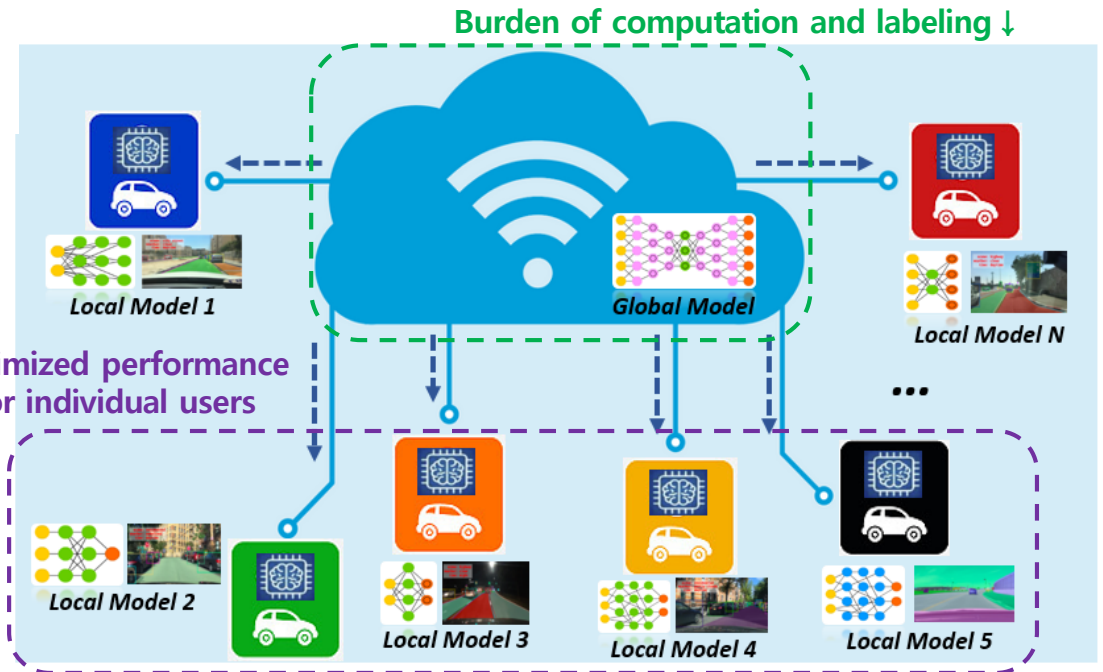
- It is difficult to cope with **different environments for each user** only with the global model created by the cloud
- Re-training of AI models for each user in the cloud requires **large-scale computing/memory resources** in the cloud and may raise **personal privacy issues** and **labeling burdens**

→ **Mobile self-learning shares the burden of computation and labeling** of the cloud server and enables the achievement of **optimized performance** for individual users **without privacy issues**

◈ **Challenging point**

- HW: Difficulty in **implementing RTL of backpropagation** + Difficulty in **applying lightweight techniques**
- SW: Difficulty in **selecting unlabeled data to be used for self-learning** + Difficulty in **training only additional data** based on pre-trained weights





< Example of on-server training of DNN model>

<On-device training of DNN model (smart mobility)>

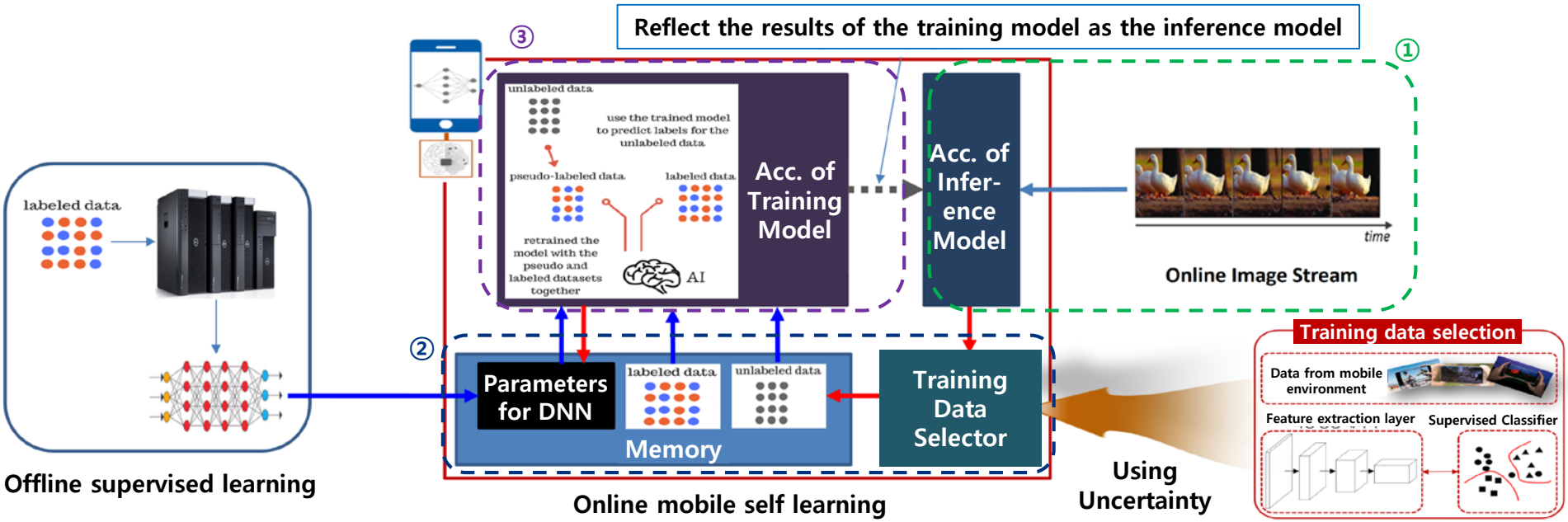
Self-learnable AI Accelerator for Mobile Devices (2)

◈ **Solution:** Implementation of **self-learnable AI accelerator platforms using inference results from mobile devices**

- ①DNN Inference of online inputs on **'Inference Accelerator'** using pre-trained weights → ②Based on the inference results, **'Training Data Selector'** distinguishes the training data to be used for self-learning later and stores these data in **'Memory'** → ③Selected data in the memory are trained as weak-labeled data in **'Training Accelerator'** and newly trained weights suitable for the current environment are updated to **'Inference Accelerator'** intermittently

◈ **Contribution**

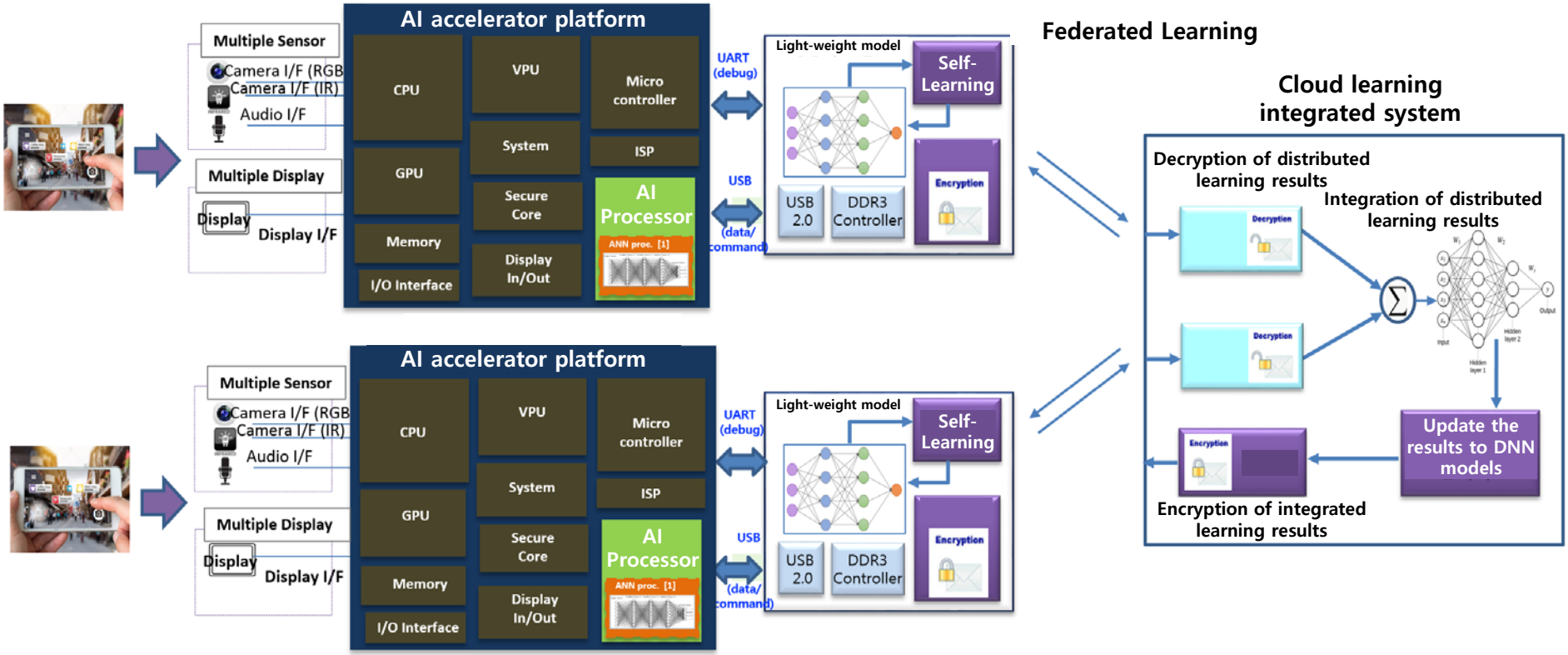
- HW: ①**RTL design of DNN backpropagation** (Approximated structure for gradient calculation, Up-sampling/scaling and weight update structure), ②**Light-weight schemes for training HW structure** (8/16bits Mixed precision, Kernel pruning)
- SW: ① **Uncertainty**-based **training data selection scheme** for maximizing performance, ② **Uncertainty**-aware **training acceleration scheme** for minimizing iterations required for self-learning



<Self-learnable mobile AI platform based on weakly-supervised learning and training data management scheme>

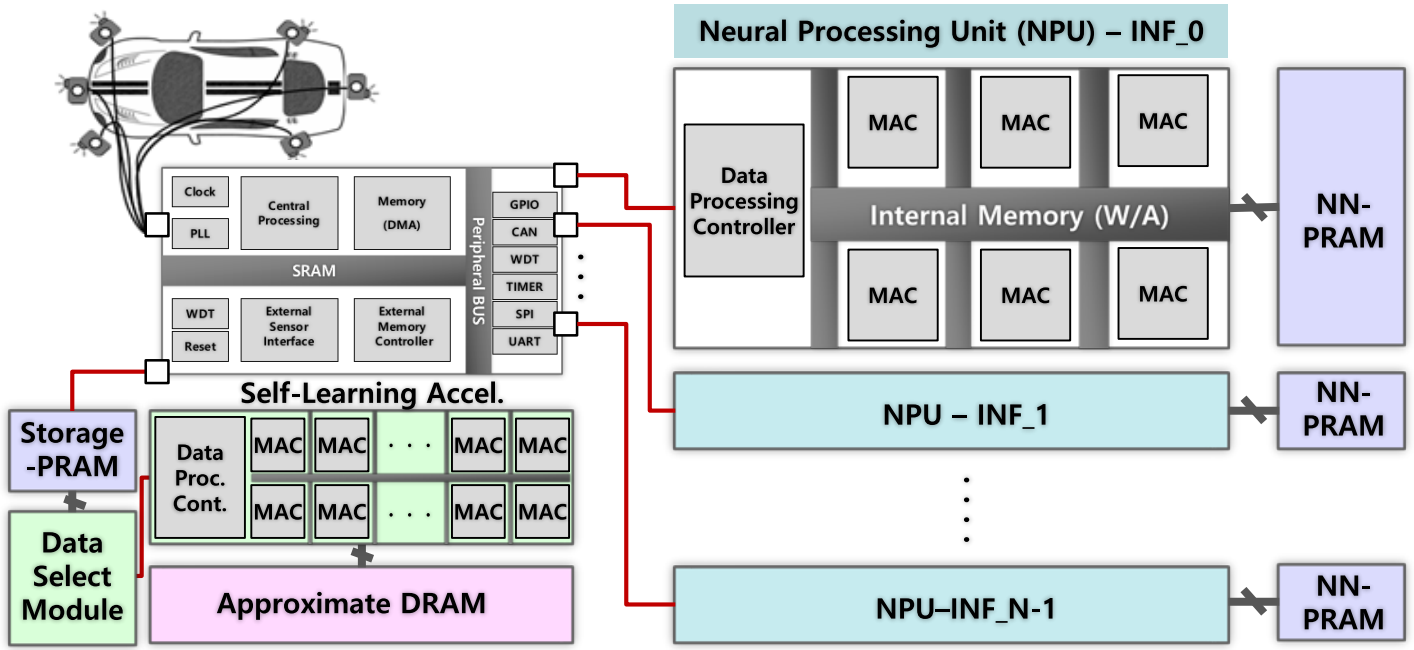
Integrated Platform for Self-learnable HW IP

- System-on-Chip design incorporating self-learnable HW IP



Integrated SoC platform for autonomous driving

Architecture-Level



Motivation

- **Computing power** required for level 4 autonomous driving (AD) is predicted to be at the level of hundreds of TOPS, and **large-capacity/high-speed storage** technology is also required for data storage and processing
- Currently, most of the platform implementations for AD are developing into a centralized architecture with a single HPC platform, but they suffer from excessively **high computational requirements and memory capacity/BW**

Solution: **Optimal SoC platform for AD** integrating a number of **scalable inference accelerators (from previous research)**, **self-learning accelerator (from short-term plan)**, **and memory dedicated to DNN (from short-term plan & previous research)**

Contribution

- **Parallel processing** by distributing multiple inputs to **multiple inference accelerators** (using **PRAM-based LP MEM**)
- Support DNN inference **optimized for the current status** using a **self-learning accelerator** (using **Approx. DRAM**)

→ Secure AI accelerator platforms that satisfy all essential elements of AD (accuracy, real-time, and low power)