Carnegie Mellon University

DLAGen:
An Open-Source Model-to-Layout Generator for Deep Learning Accelerators

Siyuan Chen and Ken Mai
Department of Electrical and Computer Engineering,
Carnegie Mellon University

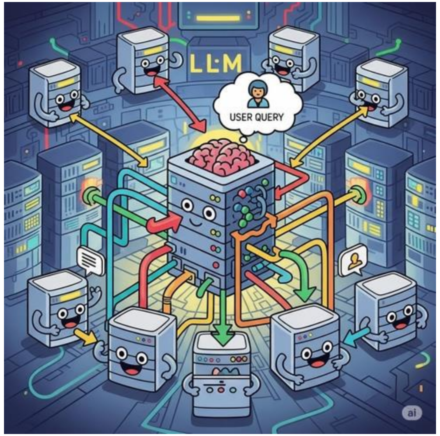
Open-Source Computer Architecture Research
June 21, 2025

Deep Learning: Diverse Applications

- Deep learning inference required at different scale for diverse applications
- Strict energy and area efficiency requirements
  - Deep learning accelerators (DLAs)

Meta-RayBan smart glasses

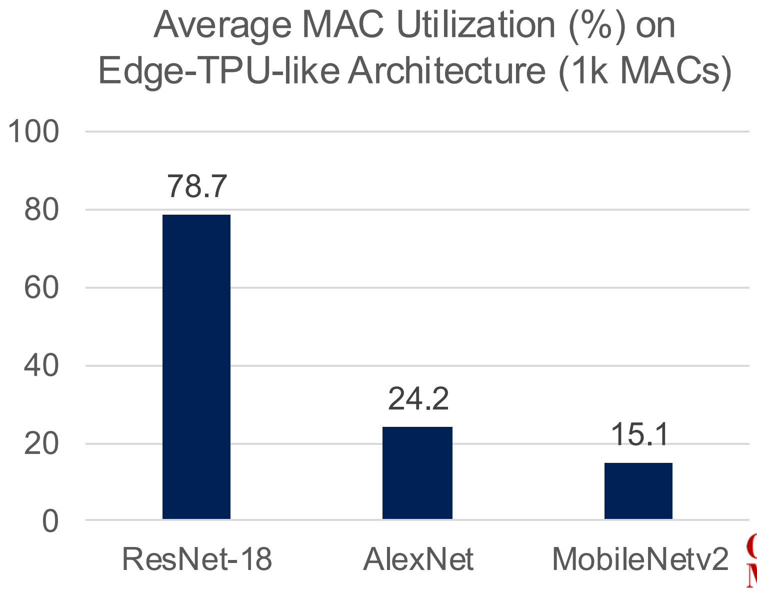
Waymo autonomous taxi



LLMs in datacenter

Current Approach: IP Reuse

- Reuse the same DLA architecture across applications/scales
- Open-source IPs: Gemmini (Berkeley) [1], NVDLA (Nvidia) [2]
- Problem: different application + scale combinations have different requirements on
  - Compute (e.g., # MACs)
  - Memory (e.g., on-chip buffer size)
  - Precision and Accuracy



Customized architecture for each application and scale?
Design cost too high!

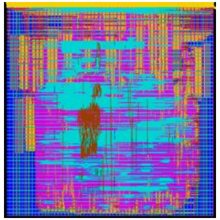
[1] H. Genc et al., DAC'21 [2] nvdla.org

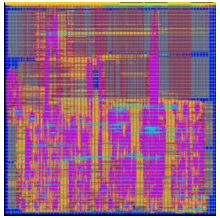
DLAGen Overview

- Automated hardware generator for deep learning accelerators
- **Search** for “optimal” architecture of given workload and resource constraint
- Reduce design cost by automatic **generation** of DLA block-level VLSI database

ResNet-18, $1mm^2$, 50 fps

Depth Estimation, $1mm^2$, 10 fps





Need a hardware abstraction to guide **search** and **generate** stages!

Loop Abstraction: GEMM Kernels as Nested Loops

- GEMM-based DL kernels represented by nested loops

```
Conv2D:
for k=[0:K):
  for ox=[0:OX):
    for oy=[0:OY):
      for fy=[0:FY):
        for c=[0:C):
          for fx=[0:FX):
            O[K][OX][OY] += W[K][C][FX][FY] * I[C][IX][IY]
```

K: output channel, **C**: input channel, **OX/OY**: output spatial, **FX/FY**: weight spatial, **IX/IY**: input spatial (derived)

Loop-based Hardware Abstraction: Spatial

Algorithm

```
Conv2D:
for k=[0:K):
  for ox=[0:OX):
    for oy=[0:OY):
      for fy=[0:FY):
        for c=[0:C):
          for fx=[0:FX):
            MAC
```

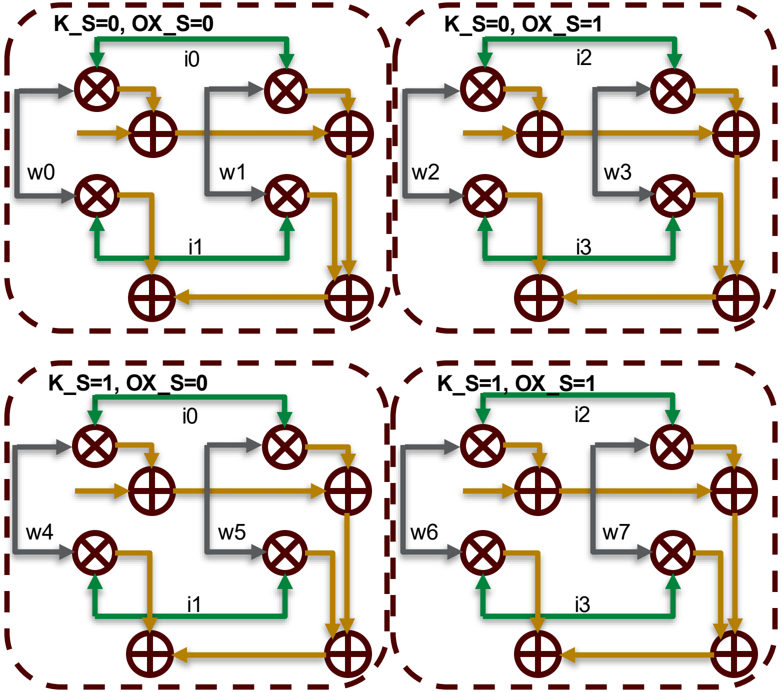
→

Hardware Description

```
for k_t=[0:K_T):
  for ox_t=[0:OX_T):
    for oy_t=[0:OY_T):
      for fy_t=[0:FY_T):
        for c_t=[0:C_T):
          for fx_t=[0:FX_T):
            parallel.for k_s=[0:K_S):
              parallel.for c_s=[0:C_S):
                parallel.for ox_s=[0:OX_S):
                  parallel.for fx_s=[0:FX_S):
                    MAC
```

- Spatial loops
  - parallelism and spatial locality of MAC array

Loop-based Abstraction Example: Spatial



K_S, C_S, OX_S, FX_S = 2

```
for k_t=[0:K_T):
  for ox_t=[0:OX_T):
    for oy_t=[0:OY_T):
      for fy_t=[0:FY_T):
        for c_t=[0:C_T):
          for fx_t=[0:FX_T):
            parallel.for k_s=[0:K_S):
              parallel.for c_s=[0:C_S):
                parallel.for ox_s=[0:OX_S):
                  parallel.for fx_s=[0:FX_S):
                    MAC
```

- Produce 4 partial sums in parallel
- Multi-cast of weights and inputs (spatial locality)

Loop-based Hardware Abstraction: Temporal

Algorithm

```
Conv2D:
for k=[0:K):
  for ox=[0:OX):
    for oy=[0:OY):
      for fy=[0:FY):
        for c=[0:C):
          for fx=[0:FX):
            MAC
```

→

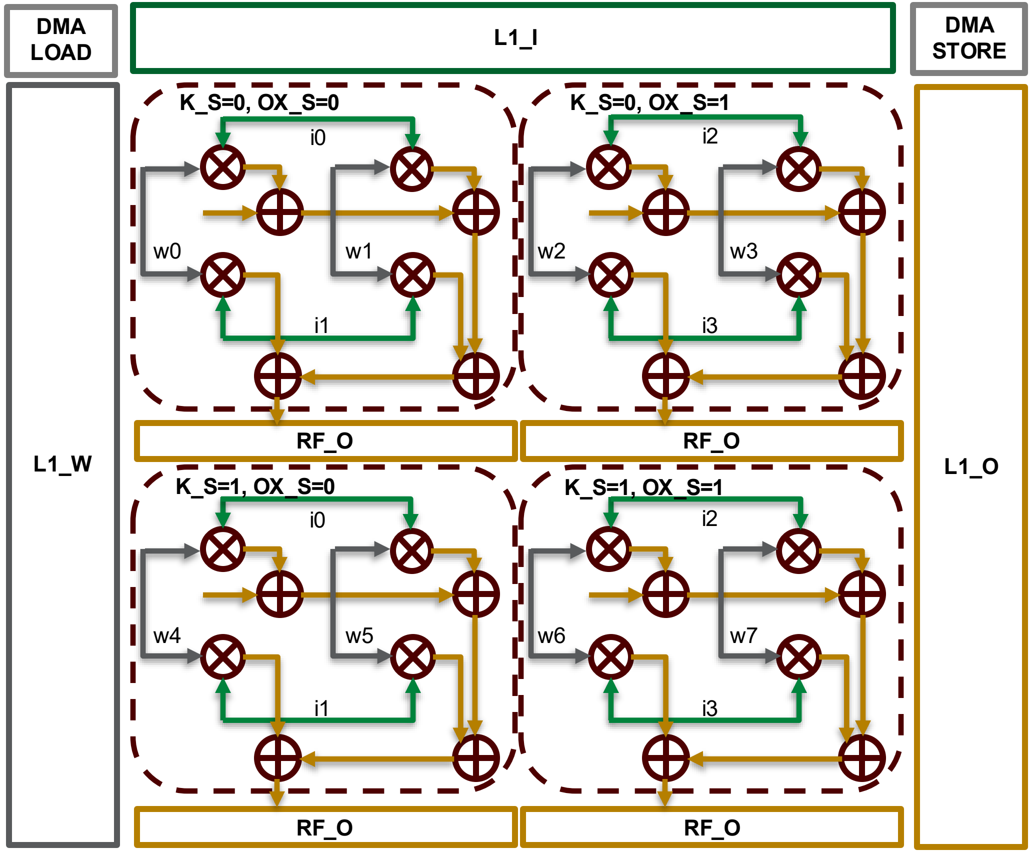
Hardware Description

```
for k_t=[0:K_T):
  dram.read(L1_W);
  dram.write(L1_O);
  for oy_t=[0:OY_T):
    dram.read(L1_I);
    for ox_t=[0:OX_T):
      L1_O.write(RF_O);
      for fy_t=[0:FY_T):
        for c_t=[0:C_T):
          for fx_t=[0:FX_T):
            L1_I.read(); L1_W.read();
            parallel.for k_s=[0:K_S):
              parallel.for c_s=[0:C_S):
                parallel.for ox_s=[0:OX_S):
                  parallel.for fx_s=[0:FX_S):
                    MAC
                    RF_O.write();
```

- Output stationary
- Temporal locality of weights and inputs (L1)

- Spatial loops
  - parallelism and spatial locality of MAC array
- Temporal loops (ordering and tiling)
  - temporal locality of memory hierarchy

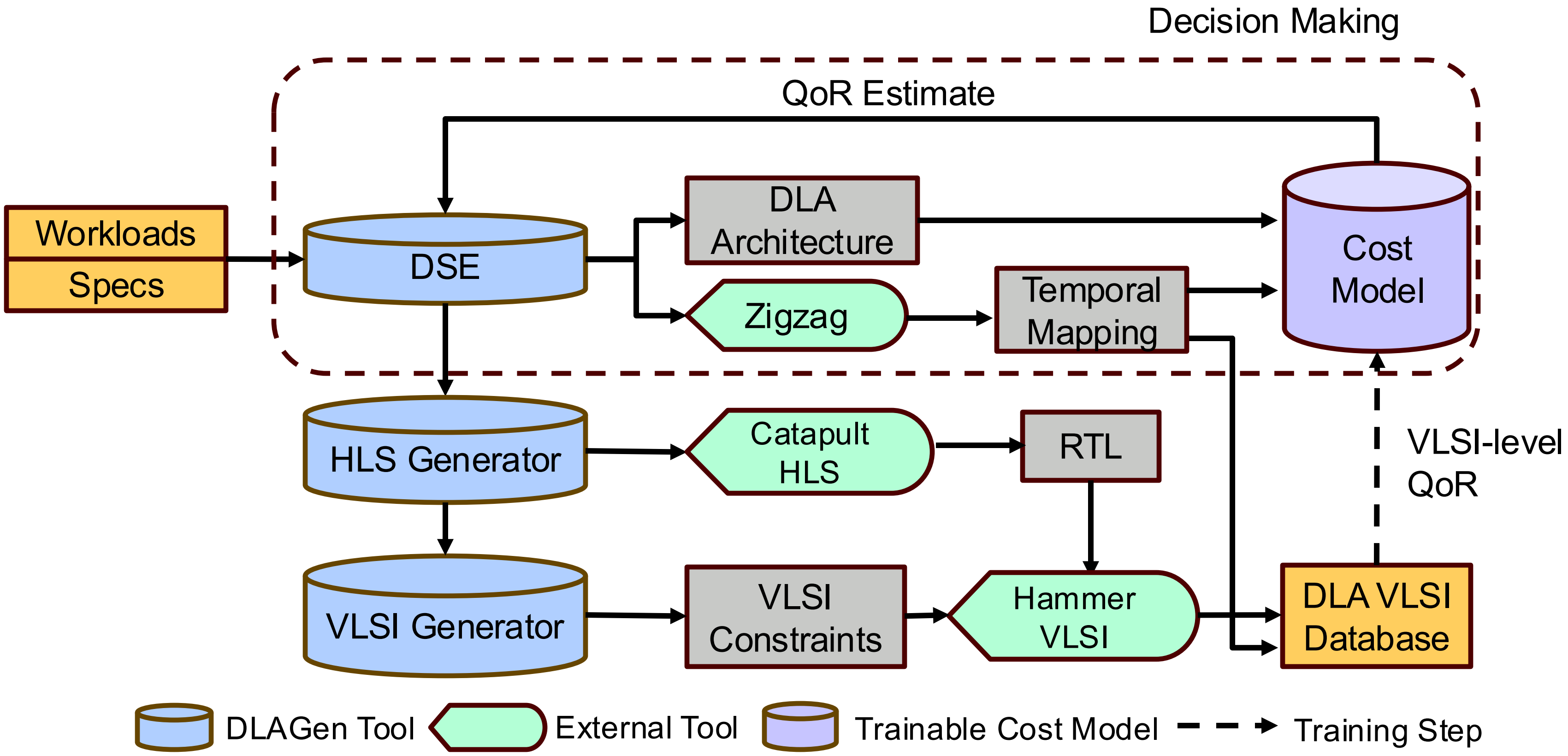
Loop-based Abstraction Example: Temporal




```
for k_t=[0:K_T):
  dram.read(L1_W);
  dram.write(L1_O);
  for oy_t=[0:OY_T):
    dram.read(L1_I);
    for ox_t=[0:OX_T):
      L1_O.write(RF_O);
      for fy_t=[0:FY_T):
        for c_t=[0:C_T):
          for fx_t=[0:FX_T):
            L1_I.read(); L1_W.read();
            parallel.for k_s=[0:K_S):
              parallel.for c_s=[0:C_S):
                parallel.for ox_s=[0:OX_S):
                  parallel.for fx_s=[0:FX_S):
                    MAC
                    RF_O.write();
```

This also looks like HLS C code!

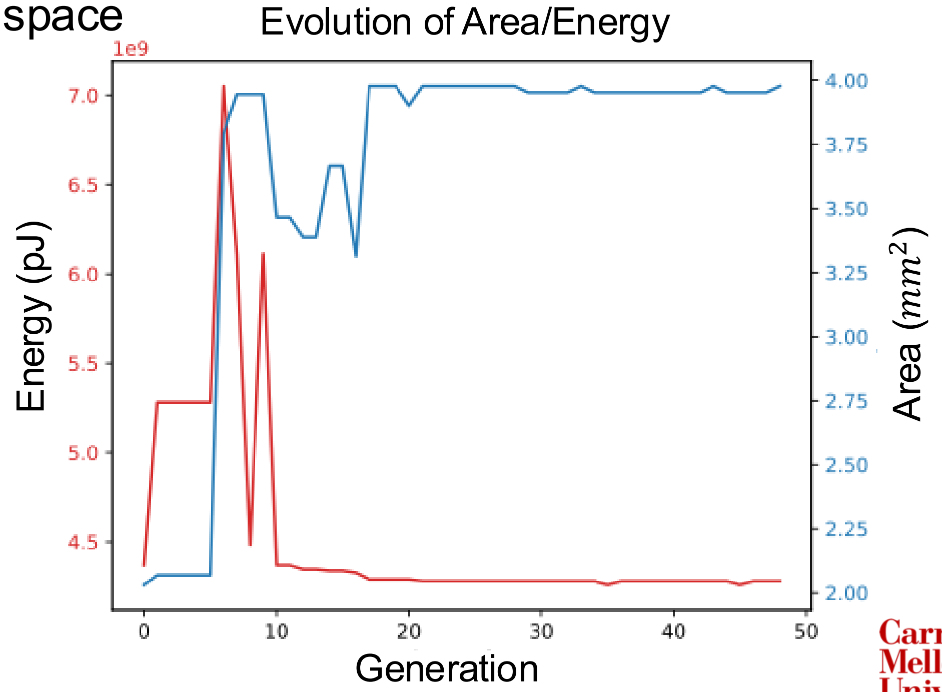
DLAGen Flow



[1] Zigzag, L. Mei et al., TC'21. [2] Hammer, H. Liew et al., DAC'22

Design Space Exploration

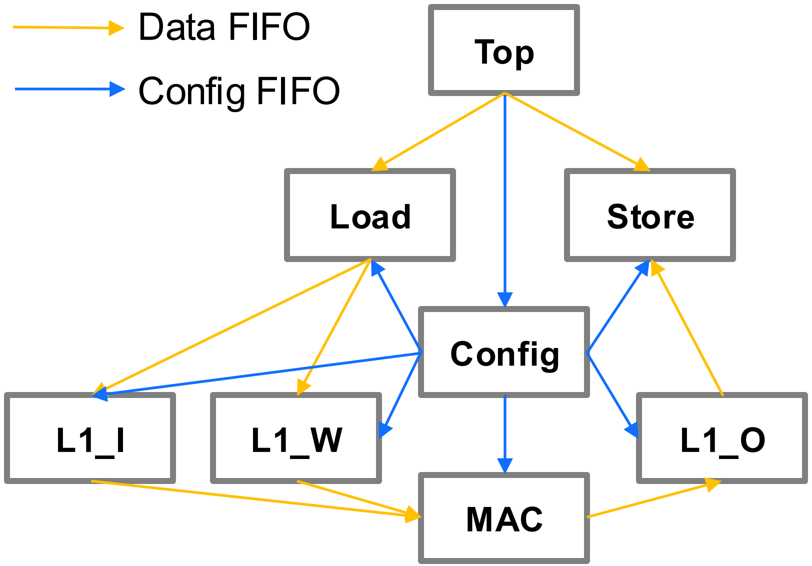
- A vector defining an individual $\vec{I}$ = {spatial loops, temporal loops, SRAM shapes}
- A cost function: $f(\vec{I})$ = (area, utilization, energy, throughput)
- Genetic Algorithm explores the design space
  - Parallelizable
  - Handles black-box functions



Carnegie Mellon University

HLS Generator

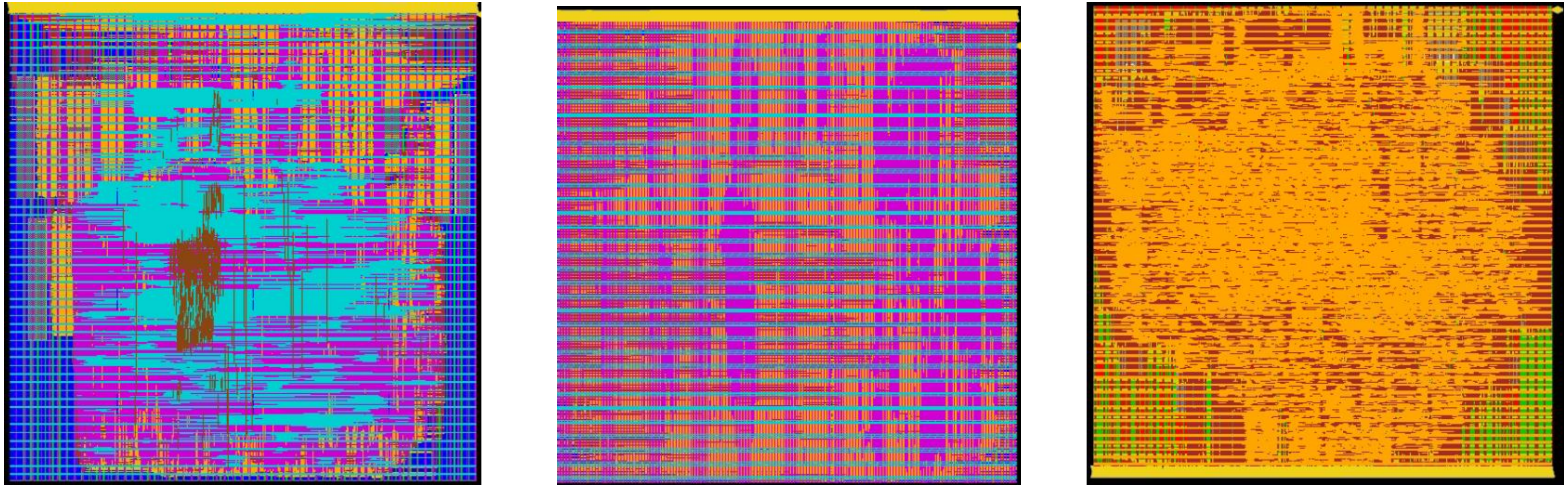
- HLS generator converts loop abstraction into hierarchical HLS-synthesizable C++
- Object-oriented construction with templatized subblocks for extensibility
- Accelerator block abstracted as a directed graph:
  - Nodes: subblocks (C++ classes)
  - Edges: FIFOs (C++ arrays/structs)



VLSI Generator

- VLSI generator outputs constraints for EDA tools
  - Clock frequency at PVT corners
  - SRAM macros: mapping and placement
- Hammer flow manager: automation and code reuse for process migration
- Final output is an Interface Logic Model ready for chip-level integration
- ***Key to DRC/LVS/timing-clean design is accurate estimation of constraints***

Portability Across Process Nodes



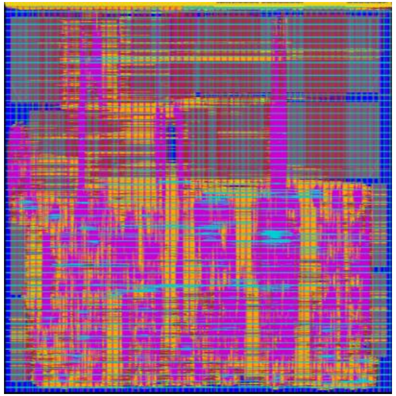
Technology	TSMC 16 nm	TSMC 28 nm	SKY130
Target FPS	ResNet-18, 50	ResNet-18, 50	ResNet-18, 50
Area	0.94 mm^2	2.18 mm^2	18.7 mm^2
Frequency (TTTT corner)	500 MHz	250 MHz	100 MHz

Dependencies and Tooling

- Current implementation requires access to Siemens and Cadence commercial tools
- Open-source “HLS” tools under-developed compared to commercial ones
  - e.g., no support of hierarchical design in Google XLScc

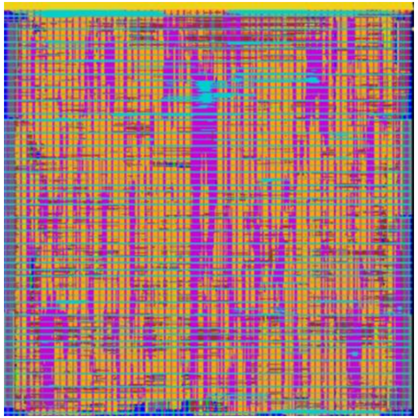
Dependency	Stage	Open?	Open Options
Zigzag	DSE	Open Source	
Catapult	HLS	Commercial	Google XLScc
Hammer VLSI	VLSI	Open Source	
Cadence DDI Suite	VLSI	Commercial	OpenROAD
TSMC, Sky130	VLSI	Mixed	

Case Study: MVS_GI Depth Estimation Model



Generated DLA
Unconstrained
(output stationary)

Technology	TSMC 16 nm	
Architecture	Unconstrained	NVDLA-like
Area	0.94 mm^2	1.00 mm^2
# MACs	1536	800
Precision	Block FP4	
Total SRAM	496 KB	552 KB
Supply Voltage	0.8 V	1.05 V
Frequency	500 MHz	987 MHz
Block Power	188 mW	864 mW
Workload	MVS_GI	
TOPs	0.60	0.20
TOPs/W	3.19 13.9x	0.23
TOPs/mm^2	0.64 3.2x	0.20



Generated DLA
NVDLA-like
(weight stationary)

Summary and Key Takeaways

- We demonstrate a fully automated model-to-layout accelerator generator for workload-optimized DLA blocks

- Use DLAGen to reduce design cost and enable new architecture and design methods
  - Heterogeneous multi-core DLAs
  - Neural architecture search with hardware co-generation
- Using tools from open-source hardware community (Zigzag, Hammer)
  - Need further dev. of open-source HLS and VLSI tools to be completely open

Thank you for your attention!

Give our tool a try if you are interested!

https://github.com/CMU-VLSI/dlagen