# A 3D Implementation of Convolutional Neural Network for Fast Inference

Narasinga Rao Miniskar*, Pruek Vanna-iampikul†, Aaron Young*, Sung Kyu Lim†, Frank Liu*, Jieun Yoo‡, Corrinne Mills‡, Nhan Tran§, Farah Fahim§, Jeffrey S Vetter*
*Computer Science and Mathematic Division, Oak Ridge National Laboratory, Oak Ridge, USA
†Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, USA
‡Department of Physics, University of Illinois Chicago, Chicago, USA
§Scientific Computing Division, Fermi National Accelerator Laboratory, Chicago, USA
Email: *(miniskkarnr,youngar,liufy,vetter)@ornl.gov,
†(v.pruek@gatech.edu, limsk@ece.gatech.edu), ‡(jyoo49, cmills10)@uic.edu, §(ntran, farah)@fnal.gov

2023 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-5109-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCAS46773.2023.10181622

***Abstract*—Low latency inference has many applications in edge machine learning. In this paper, we present a run-time configurable convolutional neural network (CNN) inference ASIC design for low-latency edge machine learning. By implementing a 5-stage pipelined CNN inference model in a 3D ASIC technology, we demonstrate that the model distributed on two dies utilizing face-to-face (F2F) 3D integration achieves superior performance. Our experimental results show that the design based on 3D integration achieves $43\%$ better energy-delay product when compared to the traditional 2D technology.**

## I. Introduction

Deploying deep learning and machine learning solutions on edge devices have many potential applications, but also poses significant technical challenges [2]. Many hardware techniques have been proposed to accelerate deep learning model inference, either to provide better inference throughput or at lower power consumption [6], [7], [9], [11]–[13], [16]–[18]. Another critical performance metric for edge inference is the latency [5], of which an active research area is to implement deep learning models on FPGA platforms [13], [21]. In this study, we introduce a design flow to generate and optimize CNN accelerators using face-to-face (F2F) bonded 3D Integrated Circuits (3DIC) [15], [19]. Although the network topology of the CNN is fixed, the weights can be reprogrammed at runtime. We demonstrate our 3DIC design flow on a 5-stage low-latency CNN accelerator, which has potential applications in high energy physics on-detector data classification.

As an application demonstration of the design flow, we implement a CNN model for the Compact Muon Solenoid (CMS) experiment [3]. Hardware accelerator chiplets which can convert raw data into physics information on the detector can be a valuable mechanism for achieving real-time track reconstruction. We have developed a compact CNN model which analyzes charge distribution patterns in the CMS pixel detector to calculate track parameters such as x,y,z coordinates, cot $\alpha$ and cot $\beta$. This chiplet architecture assumes that data hits not associated with tracks or tracks with momentum $\leq$ 0.3 GeV have already been rejected and filtered by the upstream electronics. The model currently utilizes cluster data from a single sensor readout integrated circuit (ROIC); the accuracy of the predicted values can be further improved by combining data from two correlated sensor layers. The input data for the model is generated from an analog front-end that synchronously digitizes [4] charge information every 25 ns for sensor pixels of 50 × 12.5 $\mu$m$^2$ into a 2bit value. The cluster shapes are analysed in local regions corresponding to 13 × 21 pixel

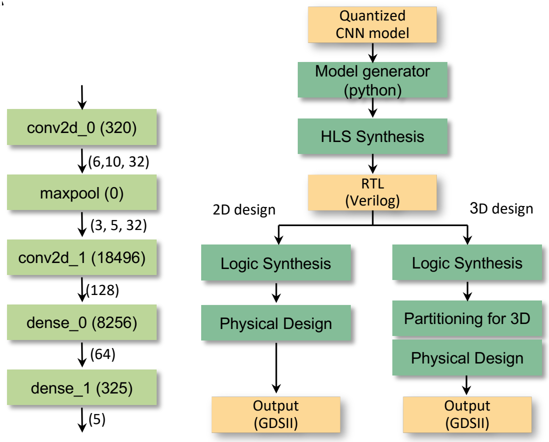

Fig. 1. LEFT: Topology of the CNN. The numbers in each module represent the numbers of weights. The numbers between each component represent the sizes of input/output tensors. RIGHT: Design flow for 2D and 3D.

## II. ASIC Design and Design Flow

The top-level logic of our CNN implementation is a five-stage pipeline, where each stage corresponds to a CNN stage shown in Figure 1. The sensor readouts are digitized in the 2bit format. Our CNN model design is also digital. We use a customized design methodology shown on the right-hand side of Figure 1, which we will explain in the rest of this section.

### A. CNN HLS Generator

The CNN generator flattens each layer of the CNN, generates HLS code for each layer, creates a data flow pipeline across layer modules, creates interfaces for input and output feature maps, and also creates interfaces for weights from each layer specific RAM. The pseudo code to generate the HLS code is shown in Figure 2. The generated HLS code can be synthesized with our HLS tool to generate the ASIC synthesizable RTL. The CNN HLS generator can intelligently select the bus width of each HLS stream and number of streams required for each input and output feature maps, and weights stream. It also considers the maximum stream bus width and SRAM memory data bus width into consideration. The HLS code generator iterates over each layer and generates HLS module code for each layer and corresponding scan chain logic if required. The runtime configuration of the weights is achieved by four independent scan chains, one for each of pipeline stage (Note that the maxpooling layer does not have any adjustable weights). The CNN generator also generates scan chain logic for each stage with weights (Convolution and Dense layers) to load the weights from SRAM memory blocks. At runtime, the scan chain logic will be invoked during the initialization to load the weights from SRAM and to set them in the internal logic registers of the network HLS module. At the end it generates a network level HLS module with a HLS data flow pipeline of all layer level HLS modules and scan chain modules with connections established among them. The process of HLS code generation from the given input quantized model is completely automated and no manual intervention is required.

```
def CNNGenerator(network, smem_datawidth,
↪  stream_max_width):
  hls_code = []
  for layer in network.layers:
    if layer.type == CONV:
      conv = ConvHLS(layer)
      hls_code +=
      ↪  conv.generate_hls_module(stream_max_width)
      hls_code += conv.generate_scan_chain(smem_datawidth)
    elif layer.type == DENSE:
      dense = DenseHLS(layer)
      hls_code +=
      ↪  dense.generate_hls_module(stream_max_width)
      hls_code +=
      ↪  dense.generate_scan_chain(smem_datawidth)
    elif layer.type == POOL:
      pool = PoolHLS(layer)
      hls_code +=
      ↪  dense.generate_hls_module(stream_max_width)
      hls_code +=
      ↪  dense.generate_scan_chain(smem_datawidth)
  hls_code += generate_network_module(network)
  return hls_code
```

Fig. 2. Pseudo code to generate HLS code for machine learning modules

The generated HLS modules for the CNN inference engine is shown in Figure 3. The scan chain module reads weights from SRAM through 128-bits data bus and fill in the registers of convolution (Conv0, Conv1) and fully-connected (Dense0, Dense1) layer HLS module stream registers. We have set the maximum stream width constraint of 4096 bits for input, output and weight streams. Conv0 requires 273 input feature elements (E:273) represented with 546 bits (B:546) in a single stream (each feature with 2 bits), 576 bits weights in a single stream, and 3840 bits of output features in a single stream. Conv1 requires 10 weight streams each of 4032 bits which can hold 2016 weights and is represented with 'B:4032 (E:2016, 10)'.

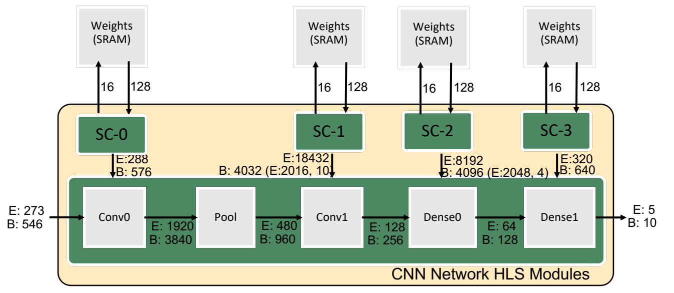

Fig. 3. CNN inference HLS Modules and connections with SRAMs (E: Number of features/weights, B: Number of bits)

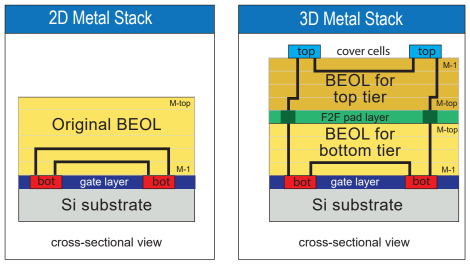

Fig. 4. Cross-sectional view of metal stack for both 2D and 3D design.

## B. Logic Synthesis and Optimization

With the resulting verilog netlist, we utilize the Synopsys Design compiler [10] to synthesize the register transfer level (RTL) from Vitus HLS into the gate-level netlist for the targeted technology node. We leverage a commercial 28nm technology to implement the ASIC design. We have set the target frequency of the synthesize module to 1GHz for all 5 layers (stages) as in Figure 3 including the the top-level module which control the data flow between different modules. There are 45 total 128-bit SRAM with 1K rows in the design. The top-level module contains the weight parsing module using a scan chain for each neural network layer (The maxpooling layer does not have any trainable weights). Therefore, there are 4 SRAM memory blocks for loading the weight. The remaining 41 SRAM blocks contains the data stream to cache the output. From Figure 3, the bus width for each stream requires 5 bank of 128-bit SRAM. So we can store 8K frames for our inputs of smart pixel network HLS modules. Table I illustrates the ASIC cell statistics after the logic synthesis stage. The cell area correlates with the number of weight element in Figure 3. The largest module is the conv1 which requires almost 18K weight elements.

## C. ASIC Physical Synthesis and Optimization

After we obtain the gate-level netlist from logic synthesis stage, we perform the physical synthesis for both 2D and 3D design. We utilize the 28nm commercial process design kit (PDK), which provides the standard cells and back-end-of-Line (BEOL) library. We generate the memory macros from the memory compiler with the same technology node. For 3D design, we integrate two 2D dies with face-to-face pads using the hybrid bonding approach, since it provides high bandwidth 3D connection with sub-micron pitch [20]. And, recently, industry has developed the F2F 3D stacking chip

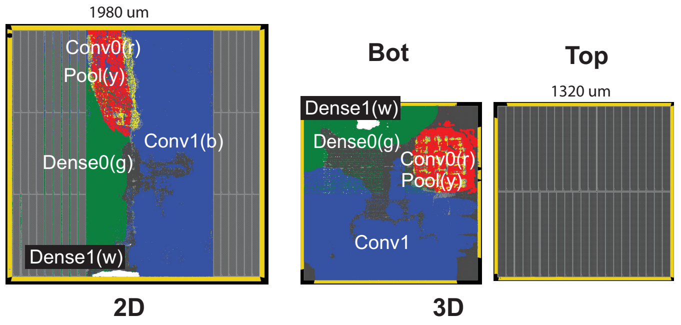

Fig. 5. Top-level Placement comparison between 2D and 3D designs. Note that the modules are color as follows: conv0 in red, pool in yellow, conv1 in blue, dense0 in green, dense1 in white

TABLE I
ASIC LOGIC SYNTHESIS STATISTICS

| Module | Cell Count ( #) | | | Cell Area |
|---|---|---|---|---|
| | Seq. | Comb. | Total | (um2) |
| **1. Conv0** | 2,255 | 105,112 | 107,367 | 125,259 |
| **2. Pool** | 3 | 3,975 | 3,978 | 6,459 |
| **3. Conv1** | 73,771 | 387,678 | 461,449 | 653,583 |
| **4. Dense0** | 32,787 | 109,532 | 142,319 | 224,748 |
| **5. Dense1** | 1,311 | 4,309 | 5620 | 8,880 |
| **6. Top level** | 10,435 | 26,176 | 36,611 | 55,004 |

(Intel Lakefield [8]). Thus, the 3D design has 12 metal layers where each tier contain 6 metal layers with one additional layer for F2F pad. The metal stacking for 3D design has been generated in this setting for parasitic extraction.

In this paper, we explore the Power-performance-area (PPA) benefits of neural network modules between 2D and 3D design with two separated experiments. We adapt the Pseudo-3D approach [14] in order to obtain the commercial quality for 3D design which will be the best estimation to compare with commercial 2D design. We adapt the memory-on-logic tier setting in the 3D design since the top-level module contains the memory connection for loading the weights and data stream. We utilize [1] to implement the memory-on-logic 3D design.

*1) Tier partitioning for 3D design:* With a given netlist, we have to perform the tier partitioning for 3D design since the netlist does not provide any information about the tier location. For 2D design, this step is not performed.

For 3D design, we place all memory on the top tier in the 3D design while all logic blocks (i.e., *conv0*, *maxpool*, *conv1*, *dense0* and *dense1*) are placed in the bottom tier.

*2) Floorplaning:* We create the partition for each neural network layer except for the top-level, which control the data flow between different modules. In 2D design the memory are placed at two sidesof the die, so the area in the middle are available for logic cells, as shown in Figure 5.

## D. Placement, Clock Tree synthesis, and Routing

After we perform the physical design flow for both 2D and 3D design [1] with floorplanning for both 2D and 3D. The final layout is illustrated in the Figure 7.

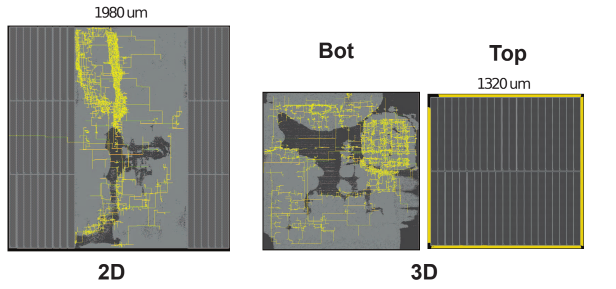

Fig. 6. Top-level Clock tree comparison between 2D and 3D designs. The yelllow lines denotes the clock nets in the design

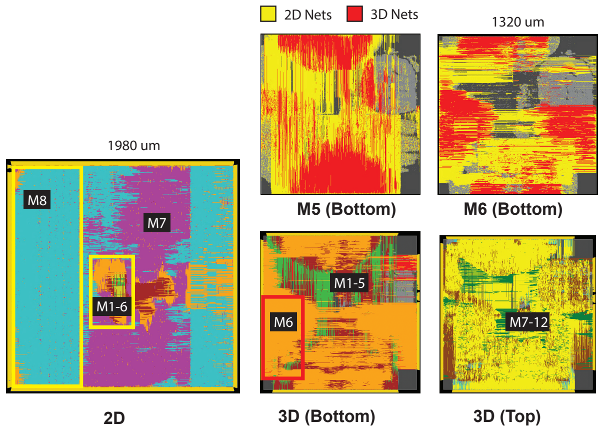

Fig. 7. Top-level GDSII comparison between 2D and 3D designs. The 3D layout include two metal layers of the bottom tier to illustrate 3D nets (highlighted in red) compared with 2D nets (highlighted in yellow).

## III. EXPERIMENTAL RESULTS

### A. Experimental setup

In this section, we perform the experiment to analyze the impact of PPA benefits of full-chip design which includes all neural network layers in Table I. In the full-chip design, we integrate the SRAM blocks for loading weights and caching data stream as mentioned in Section II-C. We flatten all blocks to evaluate the maximum achievable PPA metrics between 2D and 3D design with memory on logic setting. We use 8 metal layers in 2D design to accommodate more net connections due to top-level control logics which control the data stream among different neural network layers. For 3D design, we utilize double metal stack of 2D design with 12 metal layers (6+6), as shown in Figure 4. F2F via size, pitch, resistance and capacitance are set to be 0.5um, 1.0$\mu m$ m, 0.5$\Omega$ and 0.2fF respectively. The memory placement in 2D design is at the edge which allow the clock network to spread from the center of the die. For 3D design, we place all memory macros on the top die. The scan chains are not implemented in the standard flow because they are running at a much slower frequency than the main part of the inference engine. Hence they have negligible impact on the chip area and power.

### B. Full-Chip PPA Comparison

From Table II, we observe that the full-chip 3D design obtains higher clock frequency than the 2D design. The footprint of the 3D design is around half of the 2D design

TABLE II
FULL-CHIP POWER, PERFORMANCE, AND AREA (PPA) COMPARISON. THE PERCENTAGE VALUES IN THE LAST COLUMN INDICATE THE IMPROVEMENTS OF 3D DESIGN.

| Design | Full-Chip Design | | |
|---|---|---|---|
| | 2D | 3D | Imp. 3D |
| Effective Freq (MHz) | 742 | 881 | 18.7% |
| Footprint (mm2) | 3.9 | 1.7 | 56.4% |
| No of Cells | 892K | 806K | 10.0% |
| Wire length (m) | 64.7 | 40.6 | 37.2% |
| Total Power (mW) | 1954 | 1549 | 20.7% |
| → Internal Power (mW) | 995.9 | 826 | 17.1% |
| → Switching Power (mW) | 957.5 | 688 | 28.1% |
| → Leakage Power (mW) | 40.7 | 34 | 16.4% |
| PDP | 2633 | 1758 | 33.2% |
| EDP | 3549 | 1996 | 43.8% |

TABLE III
CLOCK TREE METRICS COMPARISON BETWEEN 2D AND 3D DESIGN

| Clock Metrics | | 2D | 3D |
|---|---|---|---|
| **Target Frequency** | **(GHz)** | 1.000 | |
| **Clock Latency** | **(ps)** | 1084 | 988 |
| **Clock Skew** | **(ps)** | 416 | 443 |
| **Clock WL.** | **(mm)** | 441 | 513 |
| **Clock Buffer** | **(#)** | 9464 | 9157 |
| **Clock Power** | **(mW)** | 151 | 158 |

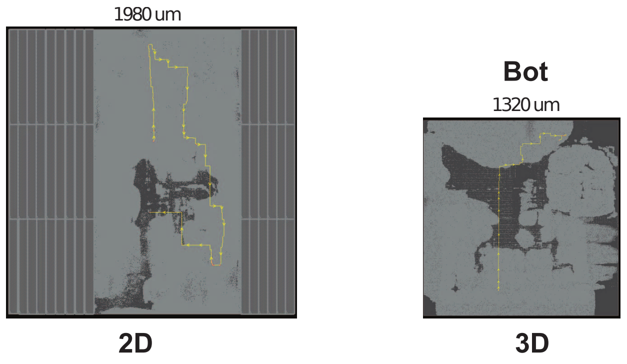

Fig. 8. Top-level critical path comparison between 2D and 3D designs

TABLE IV
FULL-CHIP TIMING COMPARISON BETWEEN 2D AND 3D DESIGN

| Parameter | | 2D | 3D | $\Delta(\%)$ |
|---|---|---|---|---|
| **Path type** | | **Reg to Reg** | | - |
| **Status** | | **Violated** | | - |
| **Launch Latency** | **ns** | 0.74 | 0.57 | 22.9% |
| **Capture Latency** | **ns** | 0.72 | 0.53 | 26.3% |
| **Skew** | **ns** | -0.02 | -0.047 | 57.4% |
| **Cell delay** | **ns** | 1.092 | 0.681 | 37.6% |
| **Wire delay** | **ns** | 0.837 | 0.403 | 43.8% |
| **Total Delay** | **ns** | 1.929 | 1.084 | 84.5% |
| **Slack** | **ns** | -0.347 | -0.134 | 61.3% |

from the die stacking. The number of cells in 3D design are fewer due to the smaller footprint and shorter I/O connection from peripherals so they do not require many buffers. The wire-length in the 3D design is also significant reduced from metal sharing and 3D nets. Therefore, the total power is reduced from smaller switching power and internal power. The internal power reduces from the fewer cell count due to shorter interconnection so the physical synthesis tools do not require as many high-speed cells with higher power to meet the timing, when compared to 2D design. The major difference is the switching power which is the result of the shorter wire-length in the 3D design. As as result, the 3D design has a significantly improved power-delay-product (PDP) and energy-delay product (EDP), at 33% and 43% respectively.

## C. Clock Metrics Comparison

For clock metrics, we consider the clock wire-length, clock latency, clock skew, and clock power in the clock metrics. The clock tree comparison of the final full-chip design is illustrated in Figure 6. We observe that the clock tree is dense in the conv0 and pool layer while the other layers mostly contain the data path for the computational logics. From Table III, we observe that the clock latency in 3D is better than 2D. However, the clock wire-length and power in 2D is better than 3D. This is because the aspect ratio of one in 3D is not the optimum for the design with high number I/O pins, which may cause some longer nets. Nevertheless, the number of clock buffers required in 3D design is fewer than 2D due to the shorter datapath interconnect benefits in 3D. Overall, the clock network in 3D is better in clock latency but the other metrics are comparable to 2D design.

## D. Full-Chip Timing Comparison

We provide a detailed analysis of the full-chip timing comparison between 2D and 3D designs. The critical paths for both designs are illustrated in Figure 8. The yellow lines denote the nets in the critical path. We observe that the critical path in 2D design is longer and has detours while the critical path in the 3D design is shorter with fewer detours. Moreover, we compare the detail within the critical path to determine the main reason of worse critical path in 2D design. Since the 3D design has a smaller footprint than the 2D design, the launch and capture latency are less with shorter clock wirelength span from the center of the die. The cell delay in 2D and 3D design are considered comparable since the input netlist is the same. The main difference in the data path delay is the wire delay in 2D design is higher than 3D due to the wire detour as in critical path layout. The reason of detour nets is from the placement limitation and larger footprint. As a result, with better clock latency and less total delay, the 3D design achieves better final timing compared to the 2D design.

## IV. CONCLUSION

In this paper, we present the design and the design methodology of a low-latency run-time configurable ASIC implementation of a five-stage CNN inference model. We conducted a comprehensive comparison between the traditional 2D technology and a face-to-face hybrid bonding 3D integration with six metal layers on each die. Detailed experimental results show that 3D integration has pronounced advantage in terms of total wire-length, power consumption, and energy-delay product (up to 43%). The performance advantages of the 3D integration makes it an ideal candidate for future extremely low-latency edge applications.

## V. Acknowledgement

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US gov- ernment retains and the publisher, by accepting the article for publication, ac- knowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public ac- cess to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

## References

[1] L. Bamberg *et al.*, "Macro-3d: A physical design methodology for face-to-face-stacked heterogeneous 3d ics," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 37–42.

[2] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[3] C. collaboration et. al., "The phase-2 upgrade of the cms tracker-technical design report," *CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2017-009, CMS-TDR-014*, 2017.

[4] L. Gaioni, D. Braga, D. C. Christian, G. Deptuch, F. Fahim, B. Nodari, L. Ratti, V. Re, and T. Zimmerman, "A 65 nm cmos analog processor with zero dead time for future pixel detectors," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 845, pp. 595–598, 2017.

[5] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, "Latency and throughput characterization of convolutional neural networks for mobile computer vision," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 204–215.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[7] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 82–95.

[8] Khushu *et al.*, "Lakefield: Hybrid cores in 3D package." in *Hot Chips Symposium*, 2019, pp. 1–20.

[9] S. Kim, G. Park, and Y. Yi, "Performance evaluation of int8 quantized inference on mobile gpus," *IEEE Access*, vol. 9, pp. 164 245–164 255, 2021.

[10] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science & Business Media, 2012.

[11] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[12] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 907–922.

[13] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, pp. 1–4, 2015.

[14] H. Park *et al.*, "Pseudo-3d approaches for commercial-grade rtl-to-gds tool flow targeting monolithic 3d ics," *Proceedings of the 2020 International Symposium on Physical Design*, 2020.

[15] S. S. K. Pentapati, D. E. Shim, and S. K. Lim, "Logic monolithic 3d ics: Ppa benefits and eda tools necessary," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 445–450.

[16] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel patterns," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 389–402.

[17] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[18] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[19] K. Salah, "Tsv-based 3d integration fabrication technologies: An overview," in *2014 9th International Design and Test Symposium (IDT)*. IEEE, 2014, pp. 253–256.

[20] T. Suga, R. He, G. Vakanas, and A. L. Manna, "Direct cu to cu bonding and alternative bonding techniques in 3d packaging," in *3D Microelectronic Packaging*. Springer, 2021, pp. 201–231.

[21] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 65–74.