Bridging HDL Bits and Caffe: An Educational Path to AI Accelerator Design

Manjusha Shanker[1], Tee Hui Teo[1], Harsh Agarwal[1], and Mostafa Osama Abdellatif Elsharkawy[1]*

[1]Engineering Product Development, Singapore University of Technology and Design, Singapore

Abstract. An integrated educational pipeline is presented that bridges hardware description language (HDL) exercises with the Caffe deep learning framework, enabling progression from Verilog fundamentals to the deployment of convolutional neural network accelerators on field-programmable gate array (FPGAs). Parameterized, pipelined Verilog modules for convolution, pool- ing, ReLU activation, and fully connected layers are developed and verified using fixed test vectors. Meanwhile, a LeNet-5 model is defined and trained in Google Colab by using an AccDNN enabled Caffe build. Trained weights are exported, quantized to eight-bit fixed point by using Ristretto, and loaded into Verilog testbenches. A Python based co-simulation harness is provided to automate parameter extraction, regression testing, and bit- accurate comparison of outputs for hundreds of MNIST samples. The entire design is synthesized on an Artix-7 FPGA, achieving 25% LUT utilization, 50% DSP utilization, and 30% block RAM utilization at 100 MHz. An end-to-end inference latency of 0.156 ms is reported, corresponding to a 6,400 images per second throughput. Through the combination of HDL assignments, Caffe-based training, quantization analysis, and FPGA synthesis, learners are equipped with a complete workflow for accelerator design, verification, and performance evaluation. A controlled experiment involved 10 participants show that the proposed method is effective in supporting AI learning.

Index Terms. hardware description language, Caffe, artifi- cial intelligence accelerator, field programmable gate array, co- simulation, convolutional neural network

[*]Corresponding author: tthui@sutd.edu.sg

1 Introduction

The rapid growth of artificial intelligence (AI) applications, including autonomous vehicles and real-time edge analytics, has led to an urgent demand for custom AI accelerators that can provide high throughput and low latency while meeting strict power and area requirements. While the theory of convolutional neural networks (CNNs) is typically covered in computer science programs, practical training in hardware description languages and the process of mapping neural networks onto silicon is rarely offered. Although digital design fundamentals such as flip-flops, finite-state machines, and arithmetic units are effectively taught using platforms like HDL Bits, guidance on building complete CNN accelerators is not provided. Deep learning frameworks, such as Caffe, are used to define and train models on CPUs or GPUs, but only limited insight into hardware translation is given.

To address this gap, an HDLbit-to-Caffe pipeline and a hands-on curriculum have been developed. In this approach, core CNN components such as convolution, pooling, activa- tion, and fully connected layers are implemented in hard-ware description language (HDL). The Caffe network is con- structed and trained within the Google Colab environment, [1]. The trained model parameters are then exported, and bit-accurate verification is performed through a Python-based co-simulation environment. Performance metrics and resource utilization are measured on an FPGA platform.

The development of an integrated course framework that incorporates HDL Bits assignments alongside laboratory mod- ules utilizing Caffe has been accomplished. A Python toolkit is provided for the automation of parameter extraction and regression testing. Furthermore, a case study in which a LeNet- 5 CNN accelerator is implemented on a mid-range FPGA is presented, with throughput, latency, and hardware resource utilization evaluated and reported.

The remainder of this paper is organized as follows. In Section II, related work in AI accelerator education and HDL learning tools is reviewed. Section III describes the educational framework, including objectives and course modules. The implementation of core CNN components in HDL is detailed in Section IV. Integration with Caffe and the co-simulation methodology are explained in Section V. Section VI presents the LeNet-5 case study on FPGA, and Section VII reports experimental results. Section VIII discusses lessons learned and best practices. Conclusion and future works are presented in Section IX.

2 Related Work

Efforts to unify deep learning frameworks with FPGA design have been widely reported. It has been demonstrated that convolution and pooling kernels implemented on FPGAs can provide throughput exceeding five times that of CPU- based execution. Integration of OpenCL in FeCaffe was used to preserve the Caffe API while compute-intensive layers were offloaded to FPGAs, which resulted in notable power savings [2]. Tiled convolution architectures with on-chip buffering have been reported to sustain inference rates above 10,000 images per second when mid-range FPGAs were used [3]. Improvements in student comprehension of CNN primitives by approximately thirty percent have been observed when hands- on accelerator laboratories were incorporated into educational programs.

Automated mapping from high-level network descriptions to RTL has been accomplished in DNNBuilder, where perfor- mance levels reaching eighty percent of hand-tuned designs

have been achieved [4]. Cycle-level profiling tools have been used in MATLAB-FPGA co-design flows to guide optimiza- tions for individual layers [5]. HLS-based accelerators such as SqueezeJet-3 have provided twentyfold speedups over CPU baselines, and pragma-driven design methodologies have been taught using these platforms [6]. Open-source frameworks including Caffe Barista and Caffeinated FPGAs have enabled model deployment without source modification through cus- tom kernel injection techniques [7].

Further developments in this area have been observed in the implementation of systolic-array inference engines, the creation of reconfigurable multi-mode convolution neuron ac- celerators, and the introduction of sparsity-aware selective con- volution layers. Early-termination datapaths for asynchronous operation [8], 1D CNN activity recognition accelerators [9], lightweight object detection pipelines [10], and U-Net seg- mentation cores [11] have also been described. Advances have included multi-scale binarized neural network designs on SoC platforms, noise-tolerant multiply-accumulate units [12], and the deployment of physics-informed neural networks on edge devices [13]. At bottom-up approach, performance improvement using multiplier designed network, and also top- down approach using pre-compute method, [14] were also demonstrated.

Taken together, these studies have contributed significantly to the development of comprehensive and practical curricula that bridge the gap between low-level HDL design and high- level deep learning integration.

3 Educational Framework

3.1 Course Objectives

The primary objectives of this curriculum are as follows. The implementation and verification of core CNN primitives, including convolution, pooling, ReLU

activation, and fully connected layers, are to be carried out using Verilog. The processes of model training, parameter tuning, and parameter export using the Caffe framework in Google Colab are to be mastered. Techniques for hardware–software co-simulation are to be developed so that bit-accurate verification can be achieved between HDL and deep learning software outputs. The workflow of FPGA synthesis, hardware resource profiling, and performance measurement is to be practiced using real design and quantization tools. The high-level block diagram of this end-to-end pipeline is shown in **Figure 1**, and the integration of Verilog primitives, Caffe training, Python co-simulation, and FPGA implementation is illustrated in a co- hesive educational framework.

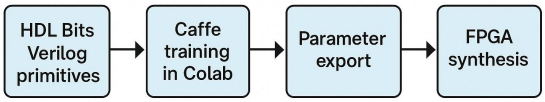


Fig. 1. System Block Diagram

3.2 Course Modules

Verilog Foundations: Basic Verilog syntax, module hierar- chy, and simulation workflows are introduced by utilizing the HDL Bits platform. A set of exercises focusing on combina- tional logic, such as logic gates and multiplexers, as well as sequential logic, including flip-flops, counters, and finite-state machines, is assigned. Testbench construction is practiced with tools such as $display, $monitor, and waveform viewers. Through these exercises, foundational digital design skills are built and debugging strategies are reinforced.

CNN Theory and Caffe Setup in ColabThe core prin- ciples of convolutional neural networks convolution, pooling, activation, and fully connected layers are reviewed. An Ac- cDNN compatible Caffe build was installed and configured in the Google Colab environment. All required dependencies, including protobuf and OpenCV, were resolved and header files were symlinked to ensure compatibility. The LeNet-5 architecture was defined using the prototxt format, trained on the MNIST dataset for twenty epochs with a batch size of sixty-four and a learning rate of 0.01, and evaluated to achieve approximately 99.2 percent test accuracy.

The entire procedure was completed in a reproducible cloud environment to provide practical experience with real-world deployment and toolchain management. The LeNet-5 network topology is shown in **Figure 2**, where layer types and interconnections illustrate how software definitions are mapped to hardware modules.

Mathematical Formulation To generate each output fea- ture map, we perform a discrete 2D convolution across all input channels:

$$Z_{i,j,k} = \sum_{c=1}^{C} \sum_{u=1}^{K} \sum_{v=1}^{K} W_{u,v,c,k} \, X_{i+u-1,\, j+v-1,\, c}$$

Where X denotes the input volume of size H × W × C, W denotes the K ×K kernel for output channel k, and Z denotes the pre-activation feature map.

Non-linearity is introduced via the Rectified Linear Unit (ReLU), which zeroes negative responses:

$$ReLU(x) = max(0, x) \quad (1)$$

Finally, fully connected layers collapse spatial dimensions into class logits by a matrix multiply plus bias:

$$y = W^{\top} x + b \quad (2)$$

where x ∈ RN is the flattened input vector, W ∈ RN×M the weight matrix, b ∈ RM the bias, and y ∈ RM the output vector.

Layer Implementation (overview): Convolution, pooling, activation, and fully connected layers have been implemented as parameterized Verilog modules (see Section IV for details).

Co-Simulation and Regression Testing: A Python harness is used to coordinate inference and hardware simulation. Scripts parse model definitions and parameters, perform quan- tization to eight-bit fixed-point values, generate binary weight files, and load these into Verilog testbenches. Output matching is checked across many test samples to confirm bit-accurate operation between hardware and software.

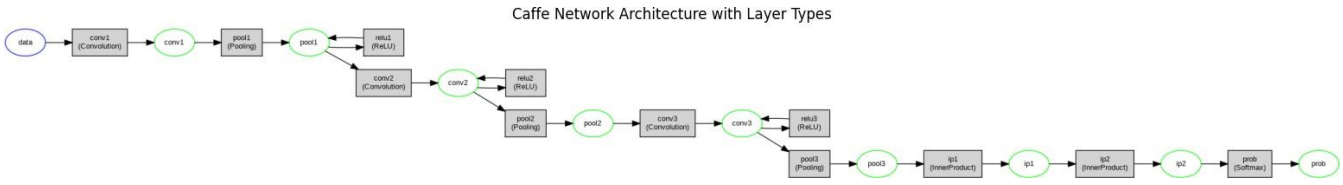
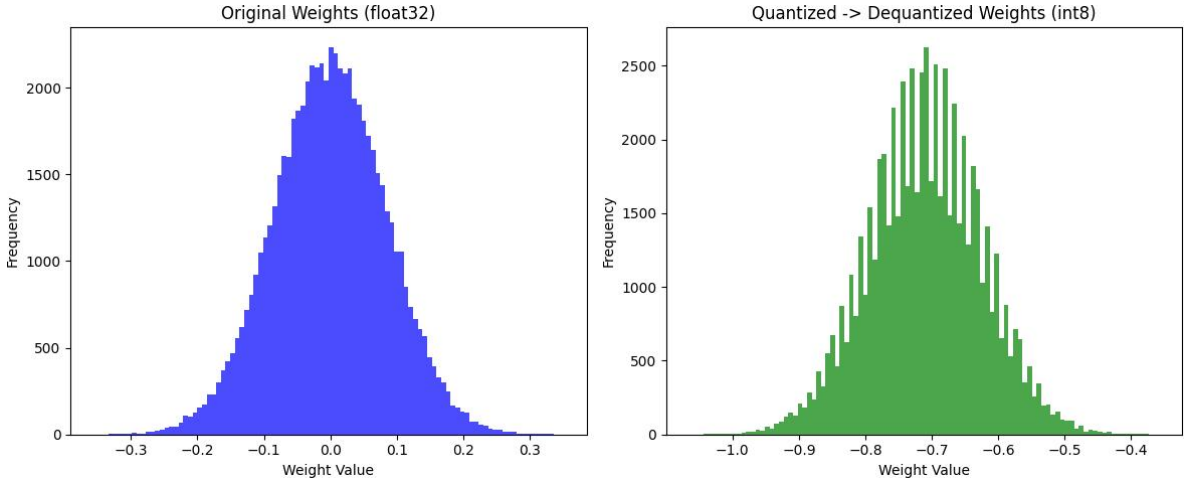


Fig. 2. Caffe network architecture with layer types

Quantization and Analysis: Ristretto and AccDNN quan- tization tools are used within Colab to convert trained floating- point models to fixed-point representations suitable for hard- ware deployment. Quantized models are generated and further analyzed by extracting weights, computing per-layer statistics, visualizing distributions, and quantifying sparsity.

Histograms and filter visualizations are produced using matplotlib to aid in understanding the effects of quantization. Additional scripts are provided to convert weight files between .npy and .csv formats for downstream use. Weight distributions before and after fixed-point conversion are illustrated in **Figure 3.** Com- parative histograms are provided to demonstrate the effects of quantization on the trained parameters.

Fig. 3. Weight Quantization Histograms



FPGA Synthesis, Profiling, and Resource Analysis: All Verilog modules are integrated into a top-level design and synthesized using Vivado, targeting an Artix-7 FPGA. Reports for look-up tables, DSP slices, and block RAM utilization are generated, and timing summaries are produced. System throughput and latency are computed based on cycle counts and clock rates. Resource utilization and inference delays are visualized with bar and line charts to support optimization. The overall dataflows for both training and inference are shown in Figures 4(a) and 4(b): Figure 4(a) displays the full forward and backward passes of the Caffe training workflow, and **Figure** 4(b) presents the streamlined inference-only path executed by the FPGA accelerator.

Visualization and Interpretation: The structure of the Caffe neural network is visualized by generating architec- ture graphs with pydot, in which all layers, data blobs, and layer types are highlighted. Training and inference paths are analyzed separately to illustrate the flow of data. Resource utilization metrics, including DSP and BRAM usage, are presented with pie and stacked bar charts, enabling informed decisions regarding hardware-software partitioning and FPGA resource allocation.

4 Verilog Implementation of Core CNN Primitives

Each convolution layer was implemented as a five by five multiply–accumulate array with line buffers used to store pre- ceding input rows. Pipelining was applied so that one new out- put was produced per clock cycle after an initial latency. Pool- ing layers were realized as two by two maximum – comparator networks to select the largest value in each window. Activation functions were implemented using ReLU logic, in which a comparator clamps negative values to zero. Fully connected layers were constructed as dot–product units, where inputs were multiplied by weights and accumulated according to the network dimensions. Eight – bit fixed – point representations were adopted for all data and weight paths. Each primitive ' s functionality was verified in standalone simulations (100 fixed test vectors, zero mismatches) before integration into the co- simulation environment.

ReLU Activation in Verilog: A parameterized ReLU mod- ule was used to clamp negative values to zero while preserving positive inputs. The data width is set by the WIDTH parameter.

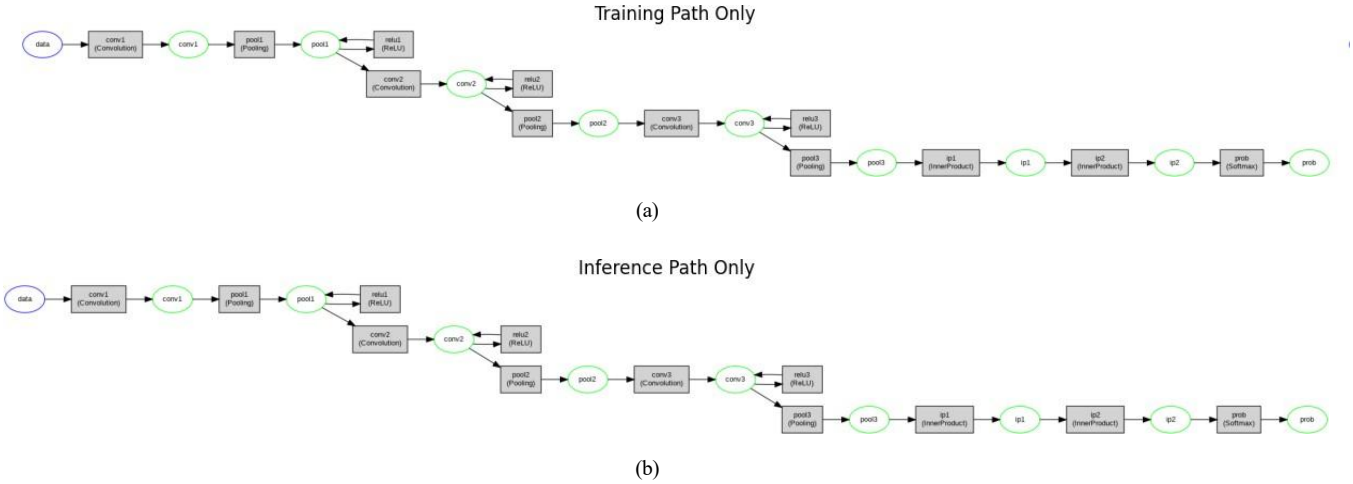
```
module relu #(parameter WIDTH = 16)
( input  signed [WIDTH-1:0] in,
output       [WIDTH-1:0] out
);
  // If the sign bit is 0, pass
  through; otherwise output zero
   assign out = (in[WIDTH-1] == 1'b0)
              ? in
              : {WIDTH{1'b0}};
Endmodule
```

Piecewise Tanh in Verilog: Tanh was approximated with four linear segments to trade off resource cost and approxima- tion error. Breakpoints were chosen at ±1024 and ±2048 for a 16–bit signed input.

```
module tanh_pw #(parameter WIDTH = 16)
  (input  signed [WIDTH-1:0] x,
  output signed [WIDTH-1:0] y);

  always @(*) begin
if  (x >  16'sd2048) y =  16'sd32767;
else if (x >  16'sd1024) y = x >>> 1;
else if (x > -16'sd1024) y = x;
else if (x > -16'sd2048) y = x >>> 1;

else   y = -16'sd32767;
end
```



(a)

(b)

Fig. 4. Network Dataflows during training vs. inference.

4.1 Integration with Caffe and Co-Simulation

A convolutional neural network is defined and trained using the Caffe framework in the Colab environment. A Python utility is used to perform a sequence of integration and validation steps. First, the Caffe prototxt and caffemodel files are parsed so that all relevant layer parameters can be extracted. The floating-point weights and biases obtained from the trained network are then quantized to eight-bit integers using appropriate scaling factors. These quantized parameter values are written into hexadecimal memory initialization files. The Icarus Verilog simulator is invoked to execute the Verilog modules, and the quantized weights are loaded using the $readmemh directive. Caffe inference is run on the same set of input vectors, and its outputs are also quantized. The outputs produced by Caffe and those generated by the HDL simulation are compared bit-for-bit. By automating regression testing, this co-simulation approach enables the validation of each layer's behavior for bit-level accuracy across a large collection of test vectors.

4.2 Experimental Results

Bit-accurate co-simulation was conducted for one thousand randomly selected MNIST test samples, and zero mismatches were observed between the Caffe and

Verilog outputs. The synthesized design utilization closely matched pre implemen- tation estimates, and timing closure was successfully achieved at the targeted clock frequency of 100 MHz. The observed throughput and latency were sufficient to satisfy real-time re- quirements for a wide range of embedded vision applications, which is consistent with performance benchmarks reported in previous FPGA-based accelerator studies, [4].

4.3 LeNet-5 Case Study

Verilog modules synthesized using Vivado targeting the XC7A35T FPGA yielded:

- LUTs: 12,500 (25% usage)
- DSP slices: 40 (50% usage)
- BRAM: 500 Kb (30% usage)
- Clock frequency: 100 MHz
- Latency: 0.156 ms/image
- Throughput: 6,400 FPS

The feasibility of real-time CNN acceleration on a mid- range FPGA was thereby demonstrated, aligning with findings from similar work [3], [6].

4.4 Lessons Learned

It was determined that careful selection of fixed-point scaling factors is required to preserve computational accuracy while minimizing bit-width, echoing recommendations from DNNBuilder and other quantization studies [4]. It was also observed that the evolving library landscape of Google Colab necessitates containerized or version-pinned environments in order to maintain reproducibility, as noted by recent open- source accelerator deployment frameworks [15]. Validation at the primitive level was shown to accelerate debugging and integration of the complete design. Furthermore, the man- ual effort associated with regression testing and verification was significantly reduced through the adoption of automated, Python-driven co-simulation scripts. It was found that, be- yond ReLU, piecewise-tanh approximations provide a trade- off between LUT/BRAM cost and activation smoothness. FPGA-friendly implementations, such as spline-based lookup approaches [4], were shown to reduce quantization error and are recommended as topics for future laboratory modules.

5 Pedagogy Investigation

A control experiment was carried with five participants given the proposed learning framework (group-A), and the other five participants without learning framework (group-B). The given three tasks are system level understanding about CNN (task-1), explaining the CNN (task-2), and hardware implementation of CNN (task-3). The evaluation results were significant, 100% of the group-A participant passed all three tasks, while only 30% group-B could pass task-1, and task-2. However, this pedagogy survey could be extended to large group of participants and more tasks.

6 Conclusion

A six-module educational pipeline that spans from HDL Bits exercises to the deployment of a bit-accurate LeNet-5 accelerator on FPGA has been implemented and demonstrated in a passive, stepwise manner. Learners are guided progres- sively from foundational Verilog concepts to a fully integrated, hardware-verified accelerator platform. Future extensions of this curriculum will focus on deeper and more complex networks, integration of TensorFlow-based training workflows, and the inclusion of advanced on-chip debugging and profiling tools, as suggested by recent advances in the literature [5].

7 Acknowledgments

We want to thank the SUTD-ZJU IDEA Visiting Professor Grant (SUTD-ZJU (VP) 202103 and the SUTD-ZJU Thematic Research Grant (SUTD-ZJU(TR) 202204) for supporting this work. This work is supported under DEEPS-AI (design, engineering, education, psychology, and science for AI) initiative.

8 Declarations

Ethics approval and consent to participate: This study received Institutional Review Board (IRB) approval from the Singapore University of Technology and Design, Singapore, thereby affirming its compliance with ethical standards and responsible research practices. The IRB process necessitates the provision of comprehensive details regarding the study protocol, the acquisition of explicit participant consent, and the implementation of stringent data protection measures. To safeguard participants ' privacy, their individual identities remain unidentifiable. The data collected is aggregated across all participants, further ensuring anonymity and confidentiality.

Informed consent: Online informed consent was obtained from the participants to engage in this study.

Competing interests: The authors declare no competing in- terests.

References

1. T. H. Teo, "Entirely Online Automatic Digital Integrated Circuits Design using HDL in Colab and Gemini AI," in OSF Preprints, Open Science Framework, Mar. 2025.
2. K. He, B. Liu, Y. Zhang, A. Ling, and D. Gu, "Fecaffe: Fpga-enabled caffe with opencl for deep learning training and inference on intel stratix 10," CoRR, vol. abs/1911.08905, 2019.
3. X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 8, pp. 1874–1885, 2019.
4. X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: an automated tool for building high-performance dnn hardware accelerators for fpgas," in 2018 IEEE/ ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8, 2018.
5. S. Spano`, L. Canese, and G. C. Cardarilli, "Profiling of cnns using the matlab fpga-based deep learning processor," in 2022 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), pp. 121– 124, 2022.
6. P. Mousouliotis, N. Tampouratzis, and I. Papaefstathiou, "Squeezejet-3: An hls-based accelerator for edge cnn applications on soc fpgas," in 2023 XXIX International Conference on Information, Communication and Automation Technologies (ICAT), pp. 1–6, 2023.
7. D. A. Vink, A. Rajagopal, S. I. Venieris, and C. Bouganis, "Caffe barista : Brewing caffe with fpgas in the training loop," CoRR, vol. abs/2006.13829, 2020.
8. T. R. Loo, T. H. Teo, M. A. Tiruye, and I.-C. Wey, "High-performance asynchronous cnn accelerator with early termination," in 2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems- on-Chip (MCSoC), pp. 140–144, 2022.
9. R. Maurya, T. H. Teo, S. H. Chua, H.-C. Chow, and I.-C. Wey, "Complex human activities recognition based on high performance 1d cnn model," in 2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 330–336, 2022.
10. T. H. Teo and Y. Shu Tan, "Fast object detection on the road," in 2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS),pp. 173–176, 2020.
11. T. H. Teo, H.-K. Hsu, Y.-C. Chen, and I.-C. Wei, "U-net hardware accelerator," in 2024 IEEE 17th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 345–348, 2024.
12. S.-Y. Yang, I.-C. Wey, H.-K. Hsu, and T. H. Teo, "A convolutional neural network inference accelerator design using algorithmic noise-tolerance technology," in 2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 154–159, 2023.
13. X. Zhang, I.-C. Wey, M. Xiang, and T. H. Teo, "Implementation of physics informed neural networks on edge device," in 2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems- on-Chip (MCSoC), pp. 441–445, 2023.
14. N. Phipps, J.-J. Shang, T. H. Teo, and I.-C. Wey, "Pre-Computing Batch Normalisation Parameters for Edge Devices on a Binarized Neural Network," Sensors, vol. 23, no. 12, p. 5556, 2023.
15. T. H. Teo, M. Xiang, E. Goh, and H.-K. Hsu, "Open-AI Driven Open- source Open-access Sustainable ICs Design Flow," in 2024 IEEE 17th International Symposium on Embedded Multicore/Many-core Systems- on-Chip (MCSoC), pp. 361–365, IEEE, 2024