MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Joint Software-Hardware Design for Green AI

Ahmed, Md Rubel; Koike-Akino, Toshiaki; Parsons, Kieran; Wang, Ye

TR2023-096 August 08, 2023

Abstract

This paper addresses the need for a framework that combines software and hardware implementation level optimization to improve the energy efficiency of sparse quantized deep neural networks (DNN). The proposed joint neural architecture optimization approach explores the best design in each paradigm, from Python simulation to hardware-FPGA implementation. As a result, it reaches the best power and area requirements in FPGA implementation. We evaluate our method on a real-time signal-processing DNN model and find that it achieves 1.7x improvements in power and 40x in area compared to the baseline implementation of the same model. Our findings demonstrate the effectiveness of the proposed framework in optimizing power and area requirements for DNNs, which is important for IoT and edge devices where resource constraints are acute.

International Midwest Symposium on Circuits and Systems (MWSCAS) 2023

© 2023 MERL. This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Joint Software-Hardware Design for Green AI

Md Rubel Ahmed, Toshiaki Koike-Akino, Kieran Parsons, Ye Wang
Mitsubishi Electric Research Laboratories (MERL), 201 Broadway, Cambridge, MA 02139, USA.
{mdahmed, koike, parsons, yewang}@merl.com

***Abstract*—This paper addresses the need for a framework that combines software and hardware implementation level optimization to improve the energy efficiency of sparse quantized deep neural networks (DNN). The proposed joint neural architecture optimization approach explores the best design in each paradigm, from Python simulation to hardware-FPGA implementation. As a result, it reaches the best power and area requirements in FPGA implementation. We evaluate our method on a real-time signal-processing DNN model and find that it achieves 1.7× improvements in power and 40× in area compared to the baseline implementation of the same model. Our findings demonstrate the effectiveness of the proposed framework in optimizing power and area requirements for DNNs, which is important for IoT and edge devices where resource constraints are acute.**

***Index Terms*—co-optimization, design automation, FPGA**

I. Introduction

Deep neural networks (DNN) have gained widespread popularity in various domains, including computer vision and signal processing. However, despite their high performance, DNN models are known to be energy-intensive [1]. For instance, the energy required to train a large DNN model for natural language processing can result in a significant carbon footprint, with an estimated 284 metric tons of CO_2 emissions, equivalent to the lifetime emissions of five cars [2]. This has led to the emergence of a new research direction called "green AI" [3]–[6], which aims to balance the tradeoff between power efficiency and inference accuracy. Green AI models have shown promise in accelerating DNN models, particularly on field-programmable gate array (FPGA) platforms [7]–[9].

The energy consumption of DNN models is primarily attributed to their architecture, particularly the computationally expensive vector-matrix multiplication and bias addition operations. FPGAs often use highly customized digital signal processing (DSP) blocks to implement these operations. Additionally, the size of training parameters, such as weights and biases, is another critical factor affecting energy consumption, with larger parameter sets requiring more energy. Therefore, various green AI models have been proposed to address this challenge, including knowledge distillation, pruning, and quantization techniques, aimed at reducing the size of DNN models while maintaining high inference accuracy. These models are designed to downsize DNN models while maintaining high inference accuracy.

Although sparse DNNs have lower computational complexities from a software perspective, their efficient hardware implementation might require significant resources. Therefore, an optimized design must consider the software and hardware requirements to minimize energy consumption. By jointly optimizing a DNN model from both domains, we can balance computational efficiency and energy consumption, resulting in a fast and energy-efficient implementation. This paper proposes a framework for optimizing DNN models from Python implementation to FPGA deployment to achieve energy-efficient designs. Our key **contributions** are as follows:

- We present a software-hardware design methodology for generating machine learning models optimized for energy-efficient hardware implementation.
- We find that an optimal word size for fixed precision data that leads to compact circuitry.
- Our methodology enables the efficient implementation of sparse quantized DNN, which can significantly reduce the model's energy consumption.

The following sections are organized as follows. Section II highlights the related works, section III describes the proposed method, section IV demonstrates the efficacy of the proposed method, and section IV-D concludes the paper.

II. Related Works

Optimization techniques for machine-learning models target accuracy, lighter implementation, and reduced power consumption. This activity can be divided into two general fronts.

A. Model Optimization in Python

One popular approach to optimizing a DNN model is to reduce the precision of the weights and activations, reducing the amount of data that needs to be transferred and processed. This can be achieved through weight quantization [10], [11] and activation quantization techniques, such as Fixed-Point Quantization and Dynamic Fixed-Point Quantization. The Hardware-Aware Automated Quantization (HAQ) [12] framework leverages reinforcement learning to determine the quantization policy for different neural networks and hardware architectures, effectively reducing latency and energy consumption with negligible loss of accuracy. [13] outlines SqueezeNet, in which a set of modifications made to the network architecture to achieve energy goal, includes aggressive channel reduction, separable 3×3 convolutions, and an element-wise addition skip connection, and optimization of the architecture by simulation, but no FPGA targeted optimization is discussed. Another approach is to compress the size of the DNN model through techniques such as pruning [14], knowledge distillation [15], and parameter sharing [16]. Pruning removes unimportant connections or filters in the network. At the same time, knowledge distillation trains a smaller network to mimic the behavior of a more extensive network.

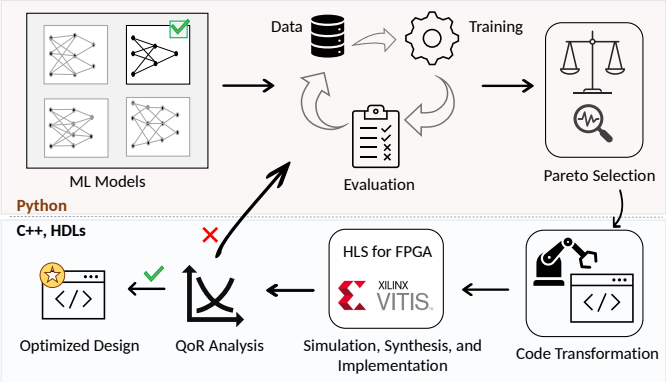


Fig. 1: Joint optimization of ML models.

B. Model Optimization in FPGA

Hardware-level optimizations have been extensively studied to improve the power efficiency of DNNs. These optimizations include designing specialized hardware accelerators and optimizing the hardware architecture, which can significantly reduce power consumption [17]. Fixed-point arithmetic is often used instead of floating-point arithmetic to represent weights and activations in DNNs, which takes less energy for computation [18]. Binary weights can also be restricted to only two possible values, which significantly reduces power consumption and hardware complexity by replacing multiply-accumulate operations with simple additions [19]. Optimization techniques can be applied to reduce power consumption in DNNs implemented on FPGAs, including software-level techniques like quantization and hardware-level techniques such as pipelining, parallelization, etc. We optimize both software and hardware domains for power efficiency.

III. Proposed Method

A general overview of the proposed approach is given in Fig. 1. The framework consists of Python simulation, C++ simulation, FPGA synthesis, and implementation for finding optimized RTL or FPGA implementation from a base ML model designed in Python. The general steps are as follows.

A. Python Simulation

In this step ML model, hyperparameter, and architecture selection are performed, and various optimization techniques are applied to maximize performance for the specific problem. Two crucial techniques used in this step are weight quantization and pruning. Weight quantization reduces the precision of neural network weights to reduce memory and computation requirements during inference. At the same time, pruning removes weights that have little effect on the output, reducing computation during inference. The optimized models or weights are the outcome of this step.

B. Pareto Selection

The complexity of a DNN model is often related to the number of non-zero parameters (nnz) associated with each ML model. Three weight quantization schemes are considered to optimize DNNs for efficient hardware implementation: no quantization, Power of Two (PoT) quantization, and Additive PoT (APoT) quantization. A Pareto optimization approach selects the most power-efficient solutions based on trade-offs between accuracy and nnz. The Pareto front is identified, and the Pareto-optimal solutions providing the best performance for a given power budget are selected based on their trade-offs between accuracy and nnz in two steps.

The first step generates Pareto solutions for each quantization scheme by plotting accuracy against the nnz. The set of solutions with the best trade-off between accuracy and complexity is selected. In the second step, the most optimized solutions are selected for RTL synthesis by comparing the Pareto fronts generated by the three quantization schemes and refining them to obtain the final Pareto front.

It is possible that some solutions only varies on one axis or, in some case, even overlap. In that case, the most promising solution can be selected using the following rules. Let S be a set of Pareto solutions and $nnz(s)$ be the number of non-zero parameters, $nmse(s)$ denote the normalized mean squared error or dB, $P(s)$ denote the power consumption, $C(s)$ denote the number of channels used, $H(s)$ denote the number of hidden layers in the solution set s. Select s^* such that:

1) If solutions vary on nnz, but nmse is constant, pick one with the lowest nnz.
$$s^* = \arg\min\{nnz(s) : s \in S\}.$$
2) If solutions vary on nmse, pick one with the lowest nmse.
$$s^* = \arg\min\{nmse(s) : s \in S\}$$
3) The most power-efficient quantization will be selected if multiple solutions overlap with different quantizations.
$$s^* = \arg\min\{P(s) : s \in S\}$$
4) If more than one solution overlaps with the same quantization, the solution with the lowest number of channels will be selected.
$$s^* = \arg\min\{C(s) : s \in S\}$$
5) If multiple solutions overlap with the same quantization and the same number of channels, then the solution with the lowest number of hidden will be selected.
$$s^* = \arg\min\{H(s) : s \in S\}$$

The aforementioned set of five rules will be henceforth referred to as "exclusion rules", with subsequent sections referring to them by their assigned rule number as appropriate.

C. Code Conversion, HLS Simulation and Synthesis

Transforming machine learning models from Python implementation to HDL code using the Vitis [20] includes converting CNN layers, activation functions, and multi-input multi-output convolutions (FIR) to C++ code while retaining the optimizations made in Python. The conversion comprises CNN layers, activation functions, and multi-input multi-output convolutions (FIR) while maintaining Python's optimizations.

To ensure accuracy, the converted C++ code must undergo csimulation. Some device and FPGA-specific tasks can be performed in this step: code transformation. The FIR function can also be optimized for performance and efficiency by creating three versions: regular multiplication, PoT, and APoT.

D. HDL Implementation

In the Vitis HLS tool, the implementation step plays a crucial role in transforming the high-level C++ design into an optimized hardware design that meets the specific constraints and requirements of the target FPGA device. This process accurately estimates the resources required for the design, including the number of logic cells, DSP blocks, memory blocks, and other FPGA resources. This step is critical for ensuring that the design meets power constraints. The implementation step generates the RTL HDL, which can program the FPGA to implement the hardware design. The HDL code can be VHDL or Verilog, depending on the user's preference. An approximate power estimation can be obtained in this step.

E. QoR Analysis

Quality of Results (QoR) is a key metric to evaluate the overall quality of FPGA designs, considering factors such as performance, power consumption, area utilization, and timing. To facilitate a comparative analysis of QoR, designers can use various methods, such as timing analysis to check if the design meets the timing constraints, resource utilization analysis to ensure the design fits within the target FPGA device's capacity, and power analysis to measure the design's power consumption, which can be optimized for low power. These analyses can be performed at different stages of the design flow, including simulation, synthesis, and implementation.

IV. Experiment and Analysis

We demonstrate the application of our proposed method for joint optimization of a real-world low-latency, high-throughput CNN network. The experiments are conducted on a machine with an Intel(®) Core™ i7-8700K CPU @ 3.70GHz, 64GB main memory, and Ubuntu 20.04.5 LTS. The target FPGA synthesis board is Xilinx ZCU104, and Vitis HLS 2022.1 is used to synthesize the kernels.

A. Proposed CNN Model

We optimize a 1D CNN-based Digital Pre-Distortion (DPD) model to mitigate distortion caused by power amplifier (PA) nonlinearities in digital communication systems. DNNs have shown promising results in mitigating nonlinear distortion in PAs [21], [22]. We use a 1D CNN-based DPD system with two input and two output channels, which can have an arbitrary number of hidden channels. Our approach involves training a neural network with large input/output signal pairs to learn the PA's nonlinear behavior. Then, the neural network pre-distorts the input signal before amplification by the PA, canceling out the nonlinear distortions introduced by the amplifier. The proposed CNN model includes adjustable network configuration parameters such as kernel size, quantization type, number of hidden channels, number of hidden layers, and percentage of weights to be pruned.

TABLE I: Candidate Pareto Solutions. Kernel refers to kernel size. Pruning (prun.) is a percentage.

kernel	quant.	hid_ch	hid_layer	prun.	nmse	nnz
7	no_quant	4	2	30	-30.13	78
		8	2	65	-29.88	78
	apot	14	2	95	-28.63	20
5	no_quant	10	2	83	-28.82	34
	apot	8	2	95	-25.80	6
		4	2	95	-21.17	4
3	no_quant	8	2	83	-28.30	16
	apot	4	2	99	-12.84	2

B. Optimization in Python and Pareto Solution Selection

The optimization scope includes kernel size (four different values: 3, 5, 7, and 11.), quantization type (no quantization, PoT, APoT), number of hidden layers (2 or 3), number of hidden channels (2, 4, 6, 8, 10, 14, 16), and pruning percentage (0, 30, 65, 83, 91, 95, 98, 99). A total of 840 samples are collected by sweeping through all configurations, with metrics including normalized mean squared error (nmse) and the number of non-zero weights (nnz). The impact of each optimization technique on QoR is analyzed, including hidden channels, pruning percentage, nmse, and nnz. Pareto solutions are identified for each quantization type based on the results.

Fig. 2 shows that 216 Pareto optimal solutions are obtained after optimizing the Python code for three types of quantization. However, the large search space reduced the number of solutions to 112 by following the second step, as shown in Fig. 3 (a). Then, further exclusion rules are applied, resulting in the top 50 solutions. Eight of these top 50 solutions are presented in Table I.

C. Optimization in HLS

The following parts of this section describe optimization in the HLS domain. Optimization occurs in a couple of aspects.

Optimal word_size: Using fixed precision representation in FPGA has several advantages, including reduced power consumption, increased speed of operations, and a more compact area requirement. Fixed precision representation reduces the number of bits required to represent a floating point number, resulting in a smaller circuit size and lower power consumption. It also allows for better control over precision levels and value ranges, which can be optimized to suit specific application requirements. We compared the convolution outcomes for various inputs and arbitrary precision points to determine the optimal word size for fixed precision representation in CNN implementation. The results of our experiments, shown in Fig. 3 (b), indicate that a word size of 16 or <16, 6>, where the integer part is 6, and the floating part is 10, provides the best outcome in terms of mean squared error (MSE).

Pragma Insertion: Pragmas can substantially benefit HLS by reducing power consumption and optimizing memory access. They are commonly used for loop unrolling, data

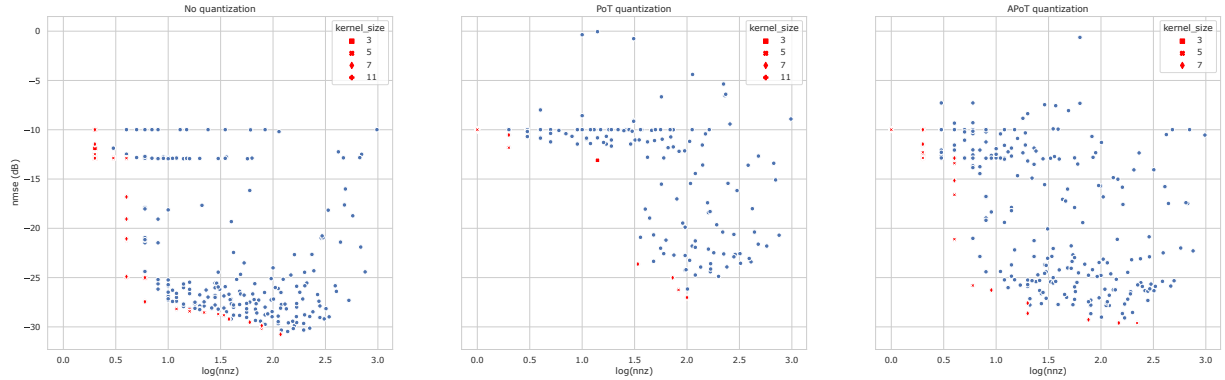
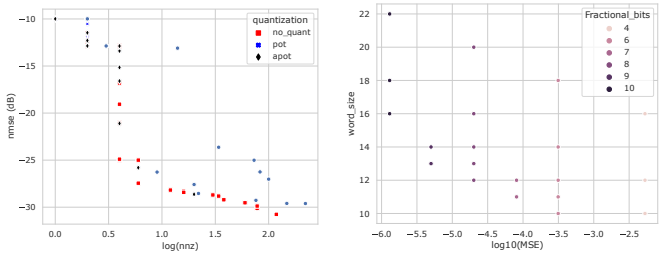
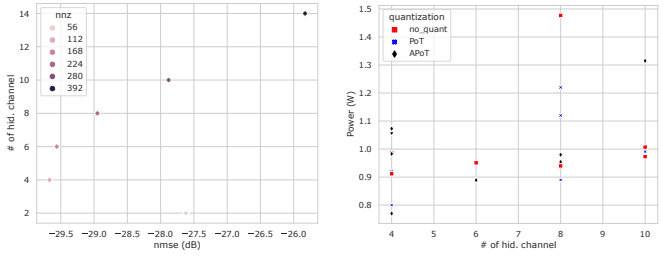


Fig. 2: Pareto solutions for each quantization type.



(a) Combined Pareto Front (b) word size vs accuracy

Fig. 3: Most promising quantization and word size selection



(a) nmse (b) Power

Fig. 4: Effect of Number of Channels on Model Performance, Sparsity, and Power Consumption

pipelining, array partitioning, etc., to minimize the number of operations and data movements in the design. By inserting pragmas in C++ source code of the CNN, DSP blocks and other resources can be conserved.

Synthesis and Implementation: After determining the optimal word size and quantization type for a neural network, specific parameters, including kernel size, number of channels, number of layers, pruning percentage, and quantization type, need to be identified from Pareto solutions. Next, we generate C++ code and synthesize it in Vitis HLS to estimate the FPGA footprint. The impact of the number of hidden channels on the power consumption and number of non-zero values (nnz) is shown in Fig. 4 (a) and (b), respectively. Based on these observations, a hidden channel size of 4 is optimal, resulting in three Pareto solutions. Synthesis and implementation profiles of these solutions are presented in Table II.

TABLE II: Synthesis and implementation profile. CNN model as kernel_hidden layers_hidden channels_pruning percentage

CNN Model	quant.	DSP	FF	LUT	Power (W)
5_1_4_95	apot	**0**	**36831**	**439933**	**0.77**
5_1_4_0	not_quant	40	82286	557820	1.2
7_1_4_30	no_quant	**38**	32901	321813	**1.07**
7_1_4_0	no_quant	50	17050	71555	1.1
3_1_4_99	apot	**0**	**77040**	**711369**	**0.95**
3_1_4_0	no_quant	25	152828	990968	1.395

D. Discussion

It is often intuitive that PoT implementation of the kernel would be more energy efficient. Still, we find that APOT yields better nmse and power consumption for a certain dataset and DNN. Our implementation of the convolution operation is zero aware. Therefore, the higher power requirement of PoT kernel operation could be explained by PoT implementation circuitry spending more energy determining if the weight value is non-zero. Table II shows the optimized designs against their non-optimized counterparts. In every row, the number of DSPs in the optimized CNN has been reduced because of the careful use of pragma or code transformation. Optimization techniques worked better in APoT quantizations but not in LUT and FF optimization in no-quantizations. The PoT kernels are absent from the top Pareto solutions as we prioritized nmse over nnz. The PoTs have a higher chance of being on the Pareto fronts if we prioritize nnz.

V. Conclusion

This paper proposes a framework to address the challenge of hardware-software optimization of DNN models by presenting a design methodology for generating optimized hardware implementation. By jointly optimizing the hardware and software components, it is possible to achieve a balance between computational efficiency and energy consumption, resulting in a system that is both fast and energy-efficient. The contributions of the framework include saving critical circuit resources and efforts spent in discovering multiple designs for rapid hardware prototyping and enabling the efficient implementation of sparse DNNs.

REFERENCES

[1] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

[2] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics.

[3] Jingjing Xu, Wangchunshu Zhou, Zhiyi Fu, Hao Zhou, and Lei Li. A survey on green deep learning. *arXiv preprint arXiv:2111.05193*, 2021.

[4] Yunhui Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[5] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *arXiv preprint arXiv:2106.08962*, 2021.

[6] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[7] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[8] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions. *arXiv preprint arXiv:1803.05900*, 2018.

[9] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. A survey of FPGA-based neural network accelerator. *arXiv preprint arXiv:1712.08934*, 2017.

[10] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[11] Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, Ye Henry Tian, and Joey Yiwei Li. Deepshift: Towards multiplication-less neural networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2359–2368, 2021.

[12] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, 2019.

[13] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[14] Yang, Tien-Ju and Chen, Yu-Hsin and Sze, Vivienne. Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[16] Pedro Savarese and Michael Maire. Learning implicitly recurrent cnns through parameter sharing. *arXiv preprint arXiv:1902.09701*, 2019.

[17] Ran Wu, Xinmin Guo, Jian Du, and Junbao Li. Accelerating neural network inference on fpga-based platforms—a survey. *Electronics*, 10(9), 2021.

[18] Mario Drumond, Tao Lin, Martin Jaggi, and Babak Falsafi. Training dnns with hybrid block floating point. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 451–461, Red Hook, NY, USA, 2018. Curran Associates Inc.

[19] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

[20] Xilinx. Vitis high-level synthesis user guide (ug1399). https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/HLS-Pragmas, February 2022.

[21] Meenakshi Rawat and Fadhel M. Ghannouchi. A mutual distortion and impairment compensator for wideband direct-conversion transmitters using neural networks. *IEEE Transactions on Broadcasting*, 58(2):168–177, 2012.

[22] Masaaki Tanio, Naoto Ishii, and Norifumi Kamiya. Efficient digital predistortion using sparse neural network. *IEEE Access*, 8:117841–117852, 2020.