LogicWiSARD: Memoryless Synthesis of Weightless Neural Networks

Igor D. S. Miranda[1], Aman Arora[2], Zachary Susskind[2], Luis A. Q. Villon[3], Rafael F. Katopodis[3], Diego L. C. Dutra[3], Leandro S. de Araújo[4], Priscila M. V. Lima[3], Felipe M. G. França[3], Lizy K. John[2], and Mauricio Breternitz Jr.[5]

[1]UFRB, Cruz das Almas, Brazil. [2]UT Austin, Austin, USA. [3]UFRJ, Rio de Janeiro, Brazil. [4]UFF, Niterói, Brazil. [5]ISCTE, Lisbon, Portugal.

***Abstract*—Weightless neural networks (WNNs) are an alternative pattern recognition technique where RAM nodes function as neurons. As both training and inference require mostly table lookups, few additions, and no multiplications, WNNs are suitable for high-performance and low-power embedded applications. This work introduces a novel approach to implement WiSARD, the leading WNN state-of-the-art architecture, completely eliminating memories and arithmetic circuits and utilizing only logic functions. The approach creates compressed minimized implementations by converting trained WNN nodes from lookup tables to logic functions. The proposed LogicWiSARD is implemented in FPGA and ASIC technologies to illustrate its suitability for edge inference. Experimental results show more than 80% reduction in energy consumption when the proposed LogicWiSARD model is compared with a multilayer perceptron network (MLP) of equivalent accuracy. Compared to previous work on FPGA implementations for WNNs, convolutional neural networks, and binary neural networks, the energy savings of LogicWiSARD range between 32.2% and 99.6%.**

***Index Terms*—Weightless neural networks, WiSARD, FPGA, VLSI**

I. Introduction

Deep Neural Network (DNN) [1] is a disruptive machine learning (ML) technique that has revitalized the Artificial Intelligence research field and its applications, notwithstanding its computation and energy-intensive profile [2]. These characteristics can hinder the deployment of ML-based IoT solutions, given the resource constraints on edge devices [3]–[5]. As a consequence, current edge implementations typically use DNN models that are smaller and less accurate when compared with state-of-the-art architectures.

Weightless Neural Networks (WNNs) are an alternative neural model, inspired by the decode processing of input signals received by the dendritic trees of biological neurons [6]. WNNs use sets of random access memory (RAM) nodes in such a way that it avoids complex arithmetic operations during inference [7], [8]. The RAM nodes serve as WNN neurons and do not store weights, but instead represent boolean functions. The inputs (usually binary) are reorganized as *tuples* to serve as addresses for the RAM nodes, and a lookup is performed to determine their response. Training a WNN involves learning logical functions in the component RAM nodes. Each input in the training set only needs to be presented to the WNN once, as opposed to DNNs which involve iterative weight updates.

A relevant WNN-based classifier is the Wilkie, Stonham, and Aleksander recognition device (WiSARD) [9], which has been successfully used since 1984 where low computational load and low inference latency are required. In recent years, there has been an increased interest in exploring WiSARD methods for various applications since new criteria were recently devised to overcome many of its shortcomings, especially in dealing with large datasets [10].

A drawback of WiSARD models is the memory requirement to store the RAM nodes, which is directly affected by the input size and the dataset statistics. To reduce the memory footprint, dictionary/hash structures are commonly used to implement the RAM nodes, providing significant memory reduction [11], [12]. Santiago et al. [13] applied an approximate membership query structure such as Bloom filters to obtain a further memory reduction.

Although the WiSARD classification performance is still not as competitive as recent DNN models, its low complexity inference may allow attractive low-power implementation of ML on edge devices. In a debut study, Ferreira et al. [14] deployed Hash-based WiSARD models capable of performing both training and inference on an FPGA using high-level synthesis (HLS). Existing implementations [13], [14] either suffer from low accuracy or high power/energy costs. Further investigation is still necessary to conceive efficient hardware architectures for WiSARD classifiers.

This work introduces a novel approach to represent WiSARD models using only logic functions, completely avoiding memories and arithmetic circuits. With this, we derive a new class of low-latency power-efficient VLSI architectures for WiSARD models. The strategy considers generating a register-transfer level (RTL) design in such a way as to leverage the synthesis tool to perform Boolean optimizations, leading to lower hardware requirements. FPGAs are particularly well suited for our proposal, because of the abundance of configurable logic blocks that can easily implement logic functions. We demonstrate the proposed method by implementing it on Xilinx FPGAs, as well as using an ASIC flow with a standard cell library. Our evaluations show significant reduction in inference time and energy consumption compared to other approaches of equivalent accuracy, demonstrating that Logic WiSARD based classifiers are suitable for latency and power

constrained extreme edge applications.

Our contributions in this paper are the following:

1) A novel approach to WiSARD models using only logic functions (no arithmetic, no memory). This leads to efficient hardware implementations. We call this architecture, the LogicWiSARD;
2) An open-source tool chain to generate these models. This tool chain is available at: *https://github.com/igordsm83/LogicWiSARD/*;
3) Implementation of LogicWiSARD in both FPGA and ASIC technologies. This shows that our approach is versatile and technology independent;
4) Comparison of the LogicWiSARD implementation with existing state-of-the-art WNN, and also with multi-layer perceptron networks (MLP), convolutional neural networks (CNN), and binary neural networks (BNN) models of similar accuracy.

The remainder of this paper is structured as follows. Section II presents a broad overview of the WiSARD technique and its characteristics. In Section III, we present our proposed solution and discuss our design principles. Section IV shows the experimental environment and evaluation results. Finally, Section V concludes the paper and introduces future research directions.

II. Background and Related Work

A WiSARD classifier performs its task by employing multiple submodels called discriminators. One discriminator is associated with one classification class. For a WiSARD model with I 1-bit elements in each input and O output classes, there are O discriminators. Each discriminator is composed of $N \equiv I/n$ RAM nodes containing 2^n entries each. The n-bit chunks of the input presented to each RAM node are also referred to as *tuples*.

Inputs (e.g., image pixel bits) are shuffled via a mapping function that is randomly defined during training initialization. The same function is used in both the training and inference phases. An example of a mapping function is illustrated in Fig. 1.

All RAMs positions are initialized to 0. During training, inputs are presented only to the discriminator corresponding to the correct output class, and its RAM nodes are updated by setting an entry to 1 whenever the address of that entry occurs in the tuple. During inference, inputs are presented to all discriminators and each discriminator produces an output which is the sum of the output of all its RAM nodes. The discriminator output scores (also called *responses*) are compared and the one with the largest score indicates the winner class. Fig. 1 also illustrates the basic functionality of a WiSARD classifier.

During inference, if an input seen is identical to one seen during training, then all RAM nodes of the corresponding discriminator will output a 1, resulting in the maximum possible response. If the input is similar but not identical, then some of the RAM nodes may produce a 0, but many will still output a 1. As long as the response of the correct discriminator is stronger than the responses of other discriminators, WiSARD will generate the correct prediction.

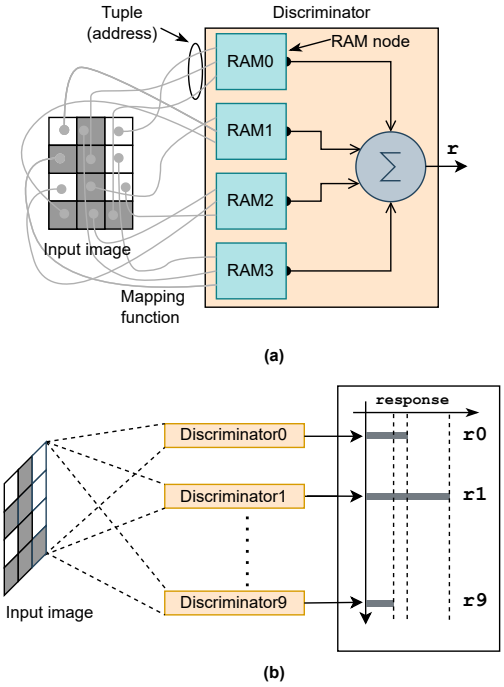

Fig. 1: (a) Example of a mapping function associating tuples to RAM nodes and a WiSARD discriminator that is trained to recognize a single class. (b) A WiSARD classifier, composed of 10 discriminators, designed to recognize digits 0-9 in an input image. In this example, the input image has a 1 and the discriminator corresponding to the digit 1 has the highest response.

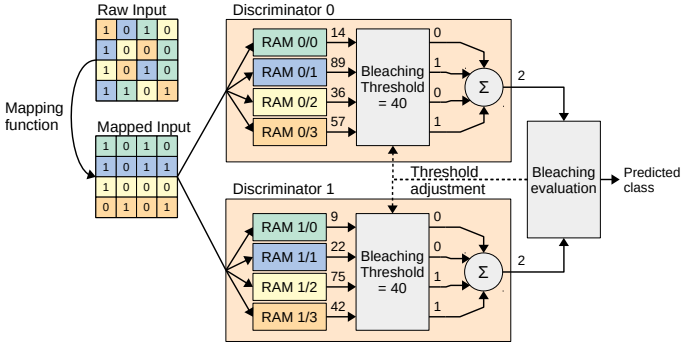

Fig. 2: Example of a two-class WiSARD classifier with bleaching algorithm during inference for $I = 16$, $n = 4$, and $N = 4$. Colors indicate the association between input tuples and RAMs.

Despite its moderate performance, this strategy is prone to overtraining (also called overfitting or saturation) as addresses with few occurrences have the same relevance as others with several occurrences. The B-bleaching [15] algorithm was

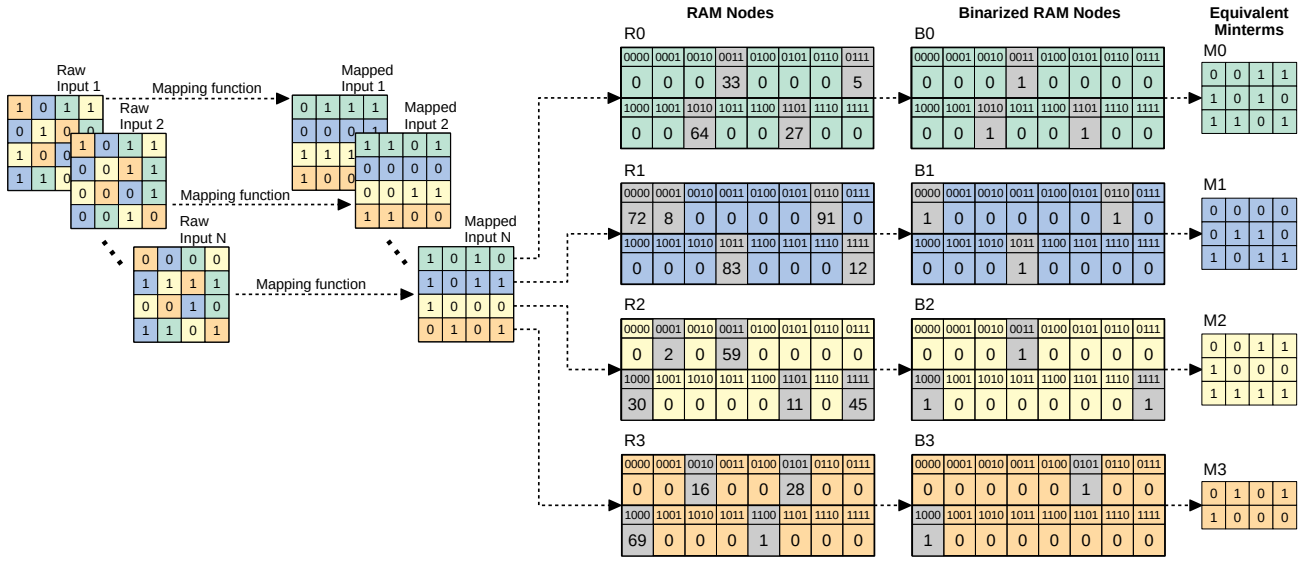

Fig. 3: Example of training, binarization, and minterm conversion of a LogicWiSARD discriminator using the proposed method. Inputs 1-N represent the input samples of the training set. Colors indicate the association between input tuples and RAMs, and the gray elements are positions whose addresses occurred in the training set. A threshold of 25 was used for binarization in this example.

introduced to WiSARD models to overcome overtraining. In this approach, RAM values are incremented by one (instead of setting to 1) whenever their addresses occur in the associated tuples across training samples. The RAMs of a trained model contain the number of occurrences of each address, again considering the association with tuples and classes. During inference, a dynamically adjustable threshold is applied to the RAM output before addition. The bleaching threshold starts with a small value for each input and increases until there are no ties among discriminators. The WiSARD classifier for $I = 16$, $n = 4$, and $N = 4$ with bleaching is depicted in Fig. 2.

As mentioned earlier, a critical issue with the WiSARD classifier is the memory requirement that grows exponentially with n. For instance, a model to classify 1024-point inputs with $n = 32$ and $N = 32$ would require 128 billion memory positions. Two successful strategies to reduce the memory footprint use dictionary/hash table structures [16] and Bloom Filters [13] to implement the RAMs. Ferreira et. al. [16] implemented a WiSARD model on a Xilinx Zynq FPGA. Although accuracies up to 92.56% could be achieved for very large model sizes, the practical model implemented on the FPGA had an accuracy of 90.73%.

III. Proposal: LogicWiSARD

In this paper we propose a method for converting trained WiSARD RAMs to Boolean functions to replace memories with logic circuits. The following sections describe each step of the proposed method.

A. Conversion of RAMs to Boolean functions

The following steps are performed to convert discriminators' RAMs to Boolean functions:

- A WiSARD model is trained;
- A suitable binarization threshold is searched;
- The model is binarized;
- Binarized RAMs are converted to Boolean functions.

This process is detailed below.

As described in Section II, a WiSARD model may be trained by sequentially presenting samples from a training set to their corresponding discriminator, according to the label of the sample. Each tuple increments a specific position at the RAM it was mapped to, producing a model that has non-binary content on its memories. Afterward, the trained model is binarized using a threshold to set RAMs' values to either 0 or 1. A single threshold value is adopted for all RAMs and discriminators.

The search for a suitable *binarization* threshold starts with an iterative process that increments the threshold until the peak accuracy is found. Accuracy is measured using a validation set, and we assume that there is only one accuracy peak in the evaluated range of threshold values. The highest threshold value that produces an accuracy close to the peak accuracy is selected. A higher threshold means more positions in the RAMs will be set to 0, which implies that the size of the model will be smaller. Hence, we always choose a higher threshold from the available options. A pre-defined tolerance is used to determine the minimum accuracy below the peak.

The *binarization* of a model is performed during training (different than bleaching), setting to 1 the most frequent RAM positions while zeroing the others. It allows the proposed

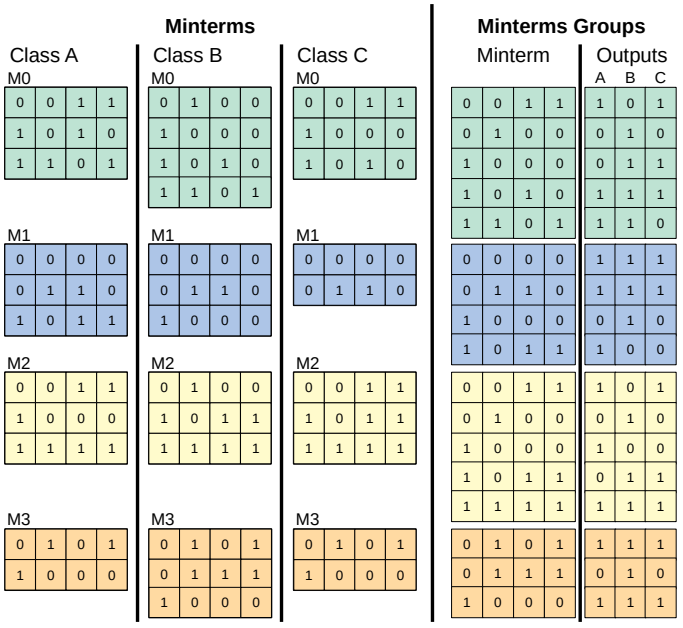

Fig. 4: Illustration of the grouping strategy for the LogicWiSARD. Multiple occurrences of a minterm across classes for the same node index (same color) are grouped into a single row of the corresponding minterm group.

method to avoid overtraining while significantly reducing memory as several RAM positions are set to zero. Because the bleaching search is avoided during inference, the final circuitry is simplified and the latency is reduced. The performance of the WiSARD with *binarization* is slightly higher than that of WiSARD with bleaching, as will be shown in Section IV.

To convert a binarized RAM to a Boolean function, we build a sum-of-minterms expression using the RAM addresses whose content is set to 1 as minterms. This can be seen as an opposite process of building a truth table. An extra step of grouping the minterms is described in Section III-B. The conversion to minterms and the grouping does not change the model functionality.

An example of this conversion is illustrated in Fig. 3 for a single discriminator. For instance, the value 33 at position 0011_2 of the RAM node R0 indicates the number of training samples whose first tuple value (in green) is 0011_2. For a threshold of 25, only positions 0011_2, 1010_2, and 1101_2 are set to 1 in the binarized RAM node B0 after *binarization* while the others are set to 0. The addresses of these three remaining positions are used as entries for truth tables which represent equivalent minterms for RAM node R0.

B. Minterms Grouping

The minterms representation described in Section III-A is compact and can be used to implement a WiSARD model only based on Boolean functions. However, it is possible to further optimize the design by rearranging the minterms to leverage the synthesis tools to perform additional Boolean optimizations.

Our grouping strategy is to build truth tables gathering coincident minterms from different classes related to the same RAM node index. The merged truth tables must have multi-bit outputs to indicate which classes contain the referred minterms. Fig. 4 shows an example of this grouping strategy.

In our experiments, a 5% reduction in hardware resources was observed by using minterms grouping before running synthesis, compared to just letting the synthesis tool perform optimizations on an ungrouped hardware implementation. We also observed that the improvement was lower for smaller models compared to larger models. Even though the compaction is low for small models, we used the grouping for all experiments presented in this work.

C. Model selection and HDL generation

Our experiments show that accuracy and model size after binarization could vary in a wide range as the mapping function is changed (a different mapping function means shuffling the bits in the input differently to obtain new tuples). Results for these experiments are presented in Section IV. Due to this variability, we select the best model for hardware implementation among others, with different mapping functions.

For each mapping function, a model is trained, binarized, and converted to minterms, as described in Section III-A. Afterward, the accuracy is computed for the test set. The model selection criteria is to choose the smallest model that meets a minimum accuracy requirement among all generated models. If none of the models meets the accuracy requirement, the smallest model among the 25% most accurate ones is picked. Also, a model size requirement is defined to stop creating the models early.

Once the model is chosen, RTL code is generated and synthesized for the target platform. Fig. 5 depicts the full processing flow for LogicWiSARD, from training to HDL generation.

We develop an open-source Python library to implement this processing flow, except for the logic synthesis step, which is available at: *https://github.com/igordsm83/LogicWiSARD/*.

The hardware architecture of the generated HDL is shown in Fig. 6. The LogicWiSARD takes one tuple every clock which is sent to the truth table associated with its index. Popcount logic is used to register minterms occurrences for each class discriminator. After all tuples are presented, the discriminator with the highest score defines the predicted class.

Several design choices are available regarding throughput, latency, and resource requirements. In our experiments, we opt to reduce the area as much as possible.

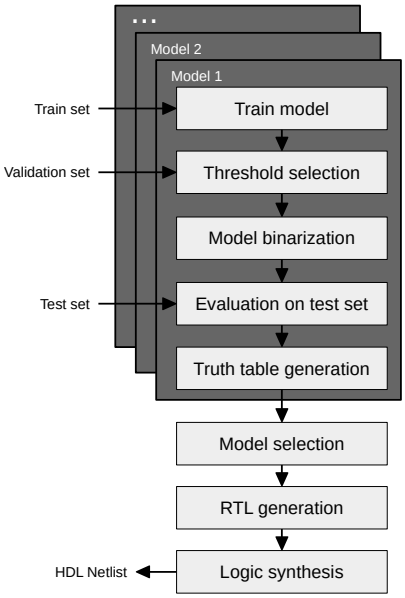

Fig. 5: Full processing flow from training to HDL generation of LogicWiSARD.

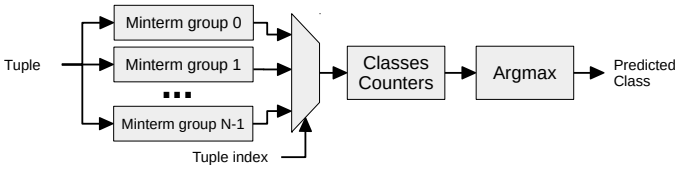

Fig. 6: The hardware architecture for LogicWiSARD. Note that full Truth tables are not stored.

IV. Experiments and Evaluation

A. Experimental Setup

For FPGA, we target the Xilinx XC7Z010-1CLG400 and XC7Z045-FFG900 FPGA devices and used Xilinx Vivado 2019.2 for synthesis and implementation. For ASIC, we targeted 45 nm technology node using the FreePDK45 [17] standard cell library and used Cadence Genus 15.20 for synthesis. Also, the memory compiler OpenRAM [18] is used for memories (to store the weights of the MLP that we compare against). The timing, power, and area numbers supplied by these tools are used for comparison and evaluation. All the FPGA results in this section are post-implementation (i.e. after placement and routing). All the ASIC results in this section are post-synthesis.

We use the well-known MNIST dataset [19] for our evaluations as it poses a typical edge ML task. This dataset contains 60k samples in the training set and 10k samples in the test set. In our experiments, we move 5k samples from the training set to a validation set that is used to find the binarization threshold. Since the MNIST images contains 8-bit pixels, we used the thermometer encoding [20] to reduce pixel resolution. This reduction in resolution may affect performance, but helps to produce feasible models for FPGA implementation.

We adjust the hyperparameters (tuple size, $n \in \{14, 28\}$, and the thermometer resolution, $T \in \{1, 2\}$) of the proposed method to produce four models with different classification performances and sizes. Each model is selected among others with the same set of hyperparameters but different random mapping functions, through the methodology described in Section III and shown in Fig. 5.

We compare the proposed method with a multilayer perceptron (MLP) hardware implementation as it still has the best tradeoff between accuracy and hardware requirements in comparison with other state-of-the-art techniques for the current edge technologies. We choose an MLP size to produce an accuracy that is equivalent to our WiSARD models so that resource requirements can be fairly compared. The chosen model is an 1-hidden layer MLP with configuration of 784-16-10 and rectified linear unit (ReLU) activation function.

Tensorflow is used for training the MLP. The trained model is then quantized (to 8-bit precision) to generate a TensorFlow Lite model. Hardware is generated using the `hls4ml` tool [21]. `hls4ml` takes 4 inputs: (1) the weights generated by TensorFlow (.h5 format), (2) the structure of the model generated by TensorFlow (.json format), (3) the precision to be used for the hardware, and (4) the FPGA part being targeted. It generates C++ code corresponding to the model, and then invokes Xilinx Vivado HLS to generate the hardware design.

We modified the generated C++ code such that the I/O interface width matched that of the hardware design for WNNs, for a fair comparison. We also updated HLS pragmas in the generated C++ code to reduce parallelization to ensure that the design fits the target FPGA. The hardware design generated by Vivado HLS (invoked by `hls4ml`) is then synthesized and implemented using Xilinx Vivado and Cadence Genus to obtain metrics such as resource usage and power consumed for FPGA and ASIC, respectively.

Additionally, we compare the LogicWiSARD with previous work on FPGA implementations for MNIST digit classification, a hashed WNN model [14], two CNN models based on the LeNet architecture [22], and a binary neural network (BNN) architecture [23]. Since these works did not present ASIC implementations for their designs, they were not included in the ASIC implementation evaluation.

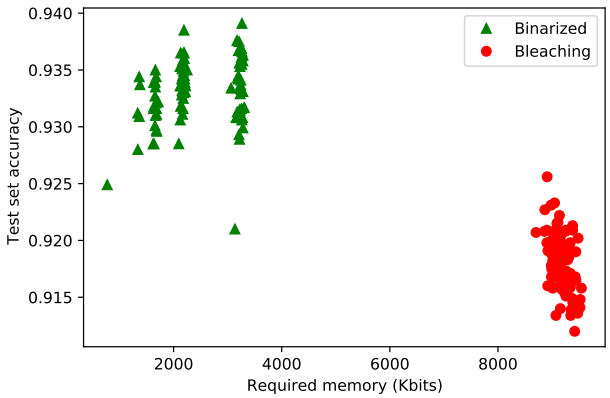

Fig. 7: Test set accuracy of 100 models with $n = 28$, $T = 1$, and different mapping functions for binarization (green) and bleaching (red) approaches.

B. Binarization in LogicWiSARD

Fig. 7 summarizes the effects on accuracy and required memory that are obtained by exploring 100 different mapping functions ($n = 28$ and $T = 1$) while using binarization and bleaching approaches. Our implementation of bleaching has a performance that is similar to the bleaching implementation reported by Santiago et al. [13], which also used $n = 28$ and $T = 1$.

As expected, the binarization provides a significant model size reduction, suggesting high effectiveness of the synthesis optimization proposed in Section III. A 4x reduction is achieved for this experiment, but it may range from 2x to 8x, depending on the model size. The observed results also indicate improved accuracy from the use of binarization for the MNIST digit classification dataset. This accuracy is achieved at reduced memory requirements.

An important observation from Fig. 7 is that, as the mapping changes, memory requirement may vary in a wide range for binarized models. Plus, there is no strong correlation between the model size and the accuracy after binarization. This relationship is not fully understood yet and deserves further exploration, as memory sizes seem to be clustered for binarized models shown in Fig. 7. However, this is used as a design opportunity as it expands the search space for a reasonable model in terms of accuracy and size. Smaller binarized models are desired as they imply fewer minterms after conversion.

In summary, binarization reduces the model size, improves accuracy, and allows model conversion from memories to logic functions. However, it increases the variance of the model size across mapping functions, hindering the model selection. Our training goal, described in Section III, is to find a binarized model near the upper-left corner of the search space depicted in Fig. 7.

C. Implementation Results and Comparison

TABLE I shows the FPGA implementation results of some variations of LogicWiSARD for MNIST digit classification. They are labeled as LogicWiSARD A, B, C, D, and E. The multiple variations illustrate how accuracy, latency, resource requirements (in LUTs, FF), and energy consumption are affected by the LogicWiSARD hyperparameters n and T. Variation E is the most costly in terms of resources and energy but provides the best classification performance. The latency ranges from 28 to 112 cycles across the five variations. Note that the LogicWiSARD implementations eliminate the need for memories (LUTRAMs and BRAMs).

The table shows results for hardware implementations of an MLP, two CNNs, and a BNN with accuracies comparable to WNNs. For the CNNs, we used the latency and resource usage numbers for optimized implementations reported by Arish et al. [22]. Power results were approximated by feeding the resource usage into the Xilinx Power Estimator (XPE) [24]. The BNN results are taken from the FINN/SFC architecture described in [23]. Also shown in TABLE I are the hardware implementation results of a state-of-the-art hashed WNN from [16].

To provide a fair comparison, the LogicWiSARDs A, B, C, and D as well as the MLP were implemented for the Xilinx XC7Z010-1CLG400 FPGA device running at 100 MHz clock, the same configuration in which the hashed WNN and the two CNNs were implemented. Likewise, the LogicWiSARD E were implemented for the Xilinx XC7Z045-FFG900 FPGA device running at 200 MHz clock, seeking to compare it with the BNN architecture. The comparisons below are made between iso-accuracy models.

We observe that LogicWiSARD A has similar accuracy to the Hashed WNN, but consumes 92.0% (53.8%) less dynamic (total) energy per inference and much fewer hardware resources. In contrast, LogicWiSARD A needs two times more cycles than hashed WNN. If a latency equivalence is desirable, the LogicWiSARD C can be compared with the hashed WNN, providing less resource utilization, less energy consumption, and more accuracy. Note that the latency of the hashed WNN design mentioned in TABLE I is the lower bound of latency based on the architecture described in [16] and assuming that it's a fully pipelined design (the latency is not provided in their paper).

Comparing the LogicWiSARD B and the CNN-2, the proposed method shows a substantial improvement in resource utilization, latency, and energy consumption. More than 99% reduction can be observed for latency and energy per inference. A comparison for hardware utilization is not straightforward in this case as the CNN-2 uses BRAMs and DSPs, but the overall reduction of the proposed method is noticeable.

The LogicWiSARD D provides a reduction in dynamic (total) energy of 63.8% (83.3%) and 99.2% (99.6%) in comparison with the evaluated MLP and the CNN-1 models, respectively. The latency reductions provided by the proposed

TABLE I: Implementation results of LogicWiSARD (proposed), MLP, WNN, CNN, and BNN (FINN/SFC) models of comparable accuracy on a Xilinx Zynq FPGA devices for MNIST digit classification

Model	Accuracy (%age)	Latency (cycles)	LUTs	LUT RAMs	FFs	36-Kbit BRAMs	DSP Blocks	Dynamic Power (Total Power) (mW)	Dynamic Energy (Total Energy) (nJ/Inference)
LogicWiSARD A (n=14, T=1)*	91.0	56	2142	0	163	0	0	17.0 (122.0)	9.5 (68.3)
LogicWiSARD B (n=14, T=2)*	92.1	112	4530	0	184	0	0	38.0 (142.0)	42.6 (159.0)
LogicWiSARD C (n=28, T=1)*	93.3	28	12134	0	142	0	0	100.0 (206.0)	28.0 (57.7)
LogicWiSARD D (n=28, T=2)*	94.5	56	37097	0	163	0	0	295.0 (404.0)	165.2 (226.2)
LogicWiSARD E (n=28, T=3)**	95.0	84	79511	0	184	0	0	1615.0 (1826.0)	1356.6 (1533.8)
MLP 784x16x10*	94.6	846	2163	0	3007	8	28	54.0 (160.0)	456.8 (1353.6)
Hashed WNN [14]*	90.7	28	9286	350	4568	128.5	5	423.0 (528.0)	118.4 (147.8)
CNN-1 (LeNet variation) [22]*	94.7	33615	5753	0	3115	7	18	58.0 (163.0)	19497.0 (54792.0)
CNN-2 (LeNet variation) [22]*	92.0	33555	3718	0	2208	5	10	43.0 (148.0)	14429.0 (49661.0)
BNN (FINN/SFC) [23]**	95.8	62	91131	-	-	4.5	-	- (7300.0)	- (2263.0)

* Synthesized for XC7Z010-1CLG400 @ 100MHz, ** Synthesized for XC7Z045-FFG900 @ 200MHz

TABLE II: Implementation results of LogicWiSARD (proposed) and an MLP model of similar accuracy on 45nm ASIC technology node for MNIST digit classification

Model	Accuracy (%age)	Latency (cycles)	Gate Count	SRAM (bits)	Area (μm^2)	Dynamic Power (Total Power) (mW)	Dynamic Energy (Total Energy) (nJ/Inference)
LogicWiSARD A (n=14, T=1)	91.0	56	9432	0	41317	1.397 (1.537)	0.782 (0.861)
LogicWiSARD B (n=14, T=2)	92.1	112	18819	0	81763	2.287 (2.552)	2.561 (2.858)
LogicWiSARD C (n=28, T=1)	93.3	28	42383	0	182073	4.772 (5.339)	1.336 (1.495)
LogicWiSARD D (n=28, T=2)	94.5	56	122128	0	520181	10.352 (11.971)	5.797 (6.704)
LogicWiSARD E (n=28, T=3)	95.0	84	282258	0	1193446	27.487 (31.130)	23.089 (26.149)
MLP 784x16x10	94.6	846	16833	39760	258795	3.802 (4.171)	32.165 (35.287)

method are 93.4% and 99.8% for MLP and CNN-1. Again, hardware utilization comparison is not simple for designs that use multiple FPGA resources. However, the proposed method does not seem to provide a better implementation than the compared methods in this case.

In the 95% accuracy range, we compare the LogicWiSARD E and the BNN models. The proposed method provides a 32.2% reduction in total energy consumption per inference and uses 12.8% fewer LUTs. Additionally, the BNN uses 4.5 BRAMs, while LogicWiSARD E does not use any. Latency, however, increases by 35.5% in the proposed method. Although the difference between the models' accuracies can not be ignored, the comparison is still valuable because there are many inference applications at the edge that can tolerate a slightly lower accuracy (0.8%), but benefit from significantly low energy and low hardware requirement.

TABLE II shows the comparison of the ASIC implementation (at 45nm technology node) of the four LogicWiSARD variations along with an MLP of comparable accuracy. The LogicWiSARD variations A, B, and C have smaller areas than MLP design, while variations D and E have larger areas. However, the reduction in energy consumption per inference ranges from 28.2% to 97.6%, for dynamic energy, and from 25.9% to 97.6%, for total energy. These results show that the LogicWiSARD approach can also be more energy efficient in ASIC technology. Although only dynamic power is explictly compared, the area consumed by a hardware implementation can be used as a proxy of the static power consumption.

Moreover, the ASIC implementation demonstrates that our approach provides a memoryless representation of WiSARD models implemented exclusively using logic functions.

V. Conclusion and Future Work

Weightless Neural Networks are a class of neural models that learn patterns without using weights. We present LogicWiSARD, a weightless neural network model that, to the best of our knowledge, is the first work that implements WiSARD models using only logic functions, completely avoiding memories and arithmetic circuits. We also present a methodology/tool flow that can be used to generate such models. This approach to designing WNN accelerators has shown much promise for achieving area- and energy-efficient designs. Compared to previous work on FPGA implementations for WNNs, CNN, and BNNs, the LogicWiSARD can produce

competitive models in terms of energy consumption, latency or hardware utilization.

By allowing trade-offs between the desired accuracy and costs of each design, our design flow provides designers with an efficient tool to search for design alternatives best suited to the project objectives. Our investigation has also uncovered interesting research paths to be explored in future work regarding the input mapping function, which may provide further model size reduction with minimal impact to accuracy.

Although WNNs trail recent machine learning algorithms in accuracy, LogicWiSARD can be deployed for several low-power latency-constrained edge applications where lower accuracy is acceptable.

ACKNOWLEDGMENT

This work was supported by the Serrapilheira Institute grant number Serra – 1912-32159 and the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES), Brazil - Finance Code 001. The authors would also like to thank the *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq), Brazil.

REFERENCES

[1] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809919306356

[2] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[3] D. L. Dutta and S. Bharali, "Tinyml meets iot: A comprehensive survey," *Internet of Things*, vol. 16, p. 100461, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660521001025

[4] M. de Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, and N. Pazos, "Robustifying the deployment of tinyml models for autonomous mini-vehicles," *Sensors*, vol. 21, no. 4, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1339

[5] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[6] I. Aleksander, M. De Gregorio, F. França, P. Lima, and H. Morton, "A brief introduction to weightless neural systems," in *17th European Symposium on Artificial Neural Networks (ESANN)*, 04 2009, pp. 299–305.

[7] W. W. Bledsoe and I. Browning, "Pattern recognition and reading by machine," in *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference*, ser. IRE-AIEE-ACM '59 (Eastern). New York, NY, USA: Association for Computing Machinery, 1959, p. 225–232. [Online]. Available: https://doi.org/10.1145/1460299.1460326

[8] I. Aleksander, M. De Gregorio, F. M. G. França, P. M. V. Lima, and H. Morton, "A brief introduction to weightless neural systems." in *ESANN*. Citeseer, 2009, pp. 299–305.

[9] I. Aleksander, W. Thomas, and P. Bowden, "WISARD·a radical step forward in image recognition," *Sensor Review*, vol. 4, no. 3, pp. 120–124, 1984. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/eb007637/full/html

[10] B. P. Grieco, P. M. Lima, M. De Gregorio, and F. M. França, "Producing pattern examples from "mental" images," *Neurocomputing*, vol. 73, no. 7, pp. 1057–1064, 2010, advances in Computational Intelligence and Learning. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231210000159

[11] H. C. C. Carneiro, F. M. G. França, and P. M. V. Lima, "Multilingual part-of-speech tagging with weightless neural networks," *Neural Networks*, vol. 66, p. 11–21, June 2015.

[12] A. S. Lima Filho, G. P. Guarisa, L. A. Lusquino Filho, L. F. Oliveira, F. M. Franca, and P. Lima, "wisardpkg–a library for wisard-based models," *arXiv e-prints*, pp. arXiv–2005, 2020.

[13] L. Santiago, L. Verona, F. Rangel, F. Firmino, D. S. Menasché, W. Caarls, M. Breternitz Jr, S. Kundu, P. M. Lima, and F. M. França, "Weightless neural networks as memory segmented bloom filters," *Neurocomputing*, vol. 416, pp. 292–304, 2020.

[14] V. C. Ferreira, A. S. Nery, L. A. J. Marzulo, L. Santiago, D. Souza, B. F. Goldstein, F. M. G. França, and V. Alves, "A feasible fpga weightless neural accelerator," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[15] D. Carvalho, H. Carneiro, F. França, and P. Lima, "B-bleaching : Agile overtraining avoidance in the wisard weightless neural classifier," in *ESANN*, 04 2013.

[16] V. C. Ferreira, A. S. Nery, L. A. J. Marzulo, L. Santiago, D. Souza, B. F. Goldstein, F. M. G. França, and V. Alves, "A feasible fpga weightless neural accelerator," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[17] NCSU. (2018) FreePDK45. [Online]. Available: https://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[18] M. R. Guthaus, J. E. Stine, S. Ataei, Brian Chen, Bin Wu, and M. Sarwar, "OpenRAM: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[19] Y. LeCun and C. Cortes, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[20] A. Kappaun, K. Camargo, F. Rangel, F. Firmino, P. M. V. Lima, and J. Oliveira, "Evaluating binary encoding techniques for wisard," in *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, 2016, pp. 103–108.

[21] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, and Z. Wu, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, pp. P07 027–P07 027, jul 2018. [Online]. Available: https://doi.org/10.1088/1748-0221/13/07/p07027

[22] A. S., S. Sinha, and S. K.G., "Optimization of convolutional neural networks on resource constrained devices," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 19–24.

[23] 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*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 65–74. [Online]. Available: https://doi.org/10.1145/3020078.3021744

[24] "Xilinx Power Estimator (XPE)," 2021. [Online]. Available: https://www.xilinx.com/products/technology/power/xpe.html