Efficient Error-Tolerant Quantized Neural Network Accelerators

Giulio Gambardella*, Johannes Kappauf*, Michaela Blott*, Christoph Doehring†
Martin Kumm‡, Peter Zipf§ and Kees Vissers*
*Xilinx Research Labs, Dublin, Ireland, Email: {giuliog,johannes,mblott,kees.vissers}@xilinx.com
† Coburg University, Germany, Email: doch1501@stud.hs-coburg.de
‡ Fulda University of Applied Sciences, Germany, Email: martin.kumm@cs.hs-fulda.de
§ University of Kassel, Digital Technology Group, Kassel, Germany, Email: zipf@uni-kassel.de

***Abstract*—Neural Networks are currently one of the most widely deployed machine learning algorithms. In particular, Convolutional Neural Networks (CNNs), are gaining popularity and are evaluated for deployment in safety critical applications such as self driving vehicles. Modern CNNs feature enormous memory bandwidth and high computational needs, challenging existing hardware platforms to meet throughput, latency and power requirements. Functional safety and error tolerance need to be considered as additional requirement in safety critical systems. In general, fault tolerant operation can be achieved by adding redundancy to the system, which is further exacerbating the computational demands. Furthermore, the question arises whether pruning and quantization methods for performance scaling turn out to be counterproductive with regards to fail safety requirements. In this work we present a methodology to evaluate the impact of permanent faults affecting Quantized Neural Networks (QNNs) and how to effectively decrease their effects in hardware accelerators. We use FPGA-based hardware accelerated error injection, in order to enable the fast evaluation. A detailed analysis is presented showing that QNNs containing convolutional layers are by far not as robust to faults as commonly believed and can lead to accuracy drops of up to 10%. To circumvent that, we propose two different methods to increase their robustness: 1) selective channel replication which adds significantly less redundancy than used by the common triple modular redundancy and 2) a fault-aware scheduling of processing elements for folded implementations.**

***Index Terms*—neural networks, safety, automotive, FPGA, quantized neural networks**

I. Introduction

In order to use an electronic device in a safety critical application, its dependability must be evaluated, usually composed of reliability, availability, maintainability and safety (RAMS). The functional safety analysis and its evaluation is regulated by standards depending on the application domain (e.g., IEC-61508 for industrial systems, ISO-26262 for road vehicles and EN 50126/8/9 for rail transport), with safety levels proportional to the criticality of the application. Failure modes, effects, and diagnostic analysis (FMEDA) [1] have to be evaluated for each of the components of a design in order to model the system's safety features.

Within machine learning, deep learning and especially CNNs have recently gained major visibility due to their high accuracy in many computer vision tasks. Nonetheless, their algorithm complexity is associated with enormous compute and memory requirements. Significant efforts have been made, tackling the inherent redundancy of neural networks by means of weights and synapse pruning, using less intensive layers (depth-wise separable convolution [2]) and non-arithmetic layers (e.g., ShiftNet [3]) or applying parameter quantization [4]. Many of those techniques require ad-hoc hardware back-ends to fully exploit the optimization strategies, making programmable devices like Field Programmable Gate Arrays (FPGAs) and Adaptive Compute Acceleration Platforms (ACAPs) very interesting implementation targets due to their reconfigurability and flexibility. Additionally, the programmability of FPGAs enables hardware architecture changes to customize the application demands, making those devices appealing for safe implementations of hardware accelerators. The need for dependable electronic systems in safety critical applications has led to the need for fast, reliable and affordable methodologies to assess and measure safety. In this paper we present hardware accelerated error injection for quantized neural networks. The main contributions of this paper are:

- High confidence, bit-accurate and high-speed error injection by means of FPGA implementations;
- Two orthogonal methodologies for deriving optimized hardware implementations for guaranteed worst-case accuracy drops in case of single errors.

II. Related Work

Multiple recent works are trying to assess the safety of neural networks, especially targeting autonomous driving systems [5]. To evaluate the safety level, fault or error injection can be used to get precise diagnostic coverage data but requires much more power in terms of computation, usually resorting to software simulation or hardware emulation. In order to overcome this problem, Bosio et al. [6] proposed the adoption of statistical permanent fault injection to decrease the computational needs, at the expense of confidence in the measured diagnostic coverage. Gehr et al. [7] rely on abstract interpretation to model neural network layers, in order to perform error injection at higher abstraction levels, thus decreasing the simulation time. In [8], a framework for bit-level error injection in the memory subsystem in Keras is presented, showing a 12% difference in resilience analysis

978-1-5386-8398-9/18$31.00 ©2019 IEEE

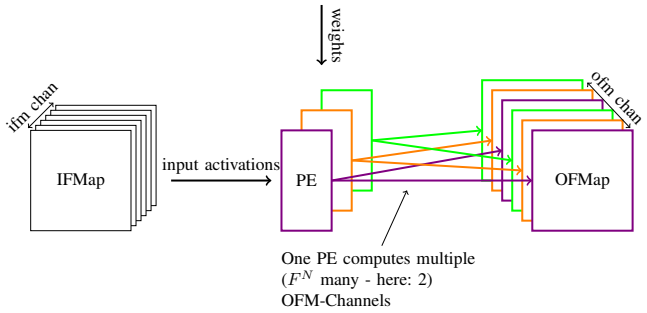

Fig. 1. Output channels folding in FINN

with respect to an hardware implementation for a multi-layer perceptron. Multiple works are targeting soft-errors only, either by performing software emulation [9] or by mean of neutron beam testing [10], proving some inherent resilience of neural networks. Even when heavily quantized, CNNs still provide some built-in fault tolerance, as shown by Nunez-Yanez et al. [11] while the system is stressed under Dynamic Voltage Scaling (DVS) and Dynamic Frequency Scaling (DFS).

III. Background

A. Neural Network Acceleration and Scheduling

CNNs are usually composed of a sequence of layers, each with its characteristics in terms of feature map sizes, channels and filters. We refer the readers to [12]–[14] for an exhaustive list of CNN accelerators targeting programmable logic. In this paper, we adopted the QNNs accelerator, code-named FINN, and described in [15], [16] which are publicly available at [17]. CNN hardware accelerators consist in general of an array of Processing Elements (PEs) on which layers or portion of layers are scheduled to be executed in sequence, with the network parameters either residing on-chip or in external memory. Whenever a single PE is faulty, it is thus going to affect multiple outputs both within the layer or among layers. The portion of the network affected by the corrupted PE depends on the scheduling of the neural network. For FINN, due to the data flow structure instantiating one compute block per layer (with multiple PEs per compute block) a single faulty PE will affect results of a single layer. This is visualized in Figure 1, which shows a case in which 3 PEs are instantiated to compute 6 output channels of a layer. The ratio between channels and PEs gives the number of clock cycles required for computing an output pixel and is defined as the folding factor f, in this case $f = 6/3 = 2$.

B. Error Injection Model

In general, QNNs are using potentially different precisions for their weights W and activations A, denoted in the following as WwAa, where w and a are the respective bit widths. Depending on the activation precision, neurons can attain different activation stages which we are injecting. The evaluated networks are thereby all using symmetric activations around zero. Thus, assuming a ternary activations network (i.e., $a = 2$), activations can have a value out of {-1, 0, 1}, which is equal to the possible stuck-at errors in our error injection context using this precision. When setting neurons in the network to stuck-at values, we are thus always injecting a bits at once.

IV. Error Injection Methodology

The adopted implementation of the FINN [15] compute data path, as an example of a typical CNN compute fabric, is based on data flow implementations of all the neural network layers. Each layer is built upon a Matrix-Vector Threshold Unit (MVTU) performing the computation. The MVTU consists of an array of multiply and accumulate engines performing the matrix-vector multiplication, and a thresholding block. This is shown in Figure 2. The thresholding block performs a comparison between the accumulation results and a set of thresholds, which are computed at design time by fusing biases, batch-normalization and activation quantization [15]. Since batch-normalization and biases, as resulted from training, are different for each OFM channel, a different set of thresholds has to be computed for each channel. In detail, the activation within an MVTU is performed by accumulating the results of $\sum_{i=1}^{N}(val > th_i)$, where val is the result of the matrix-vector accumulation operation and th_i is each of the computed thresholds. Assuming a fully binarized network (W1A1), it is possible to inject stuck-at 1 (s@1) and stuck-at -1 (s@-1) errors by setting those threshold values during run-time. We refer as s@-1 for BNN since -1 is the logical value of 0 in a BNN. A s@-1 can be implemented by fixing the th to th_{max} where th_{max} is a value which is bigger than any input can reach. Thus, the activation is never performed. Similarly, s@1 can be implemented by fixing the th to $-th_{\text{max}}$. The accumulation will always be greater than the regarding threshold. In case of multiple bit output, it is possible to fix the result activation to any possible value by fixing as many thresholds as needed to $-th_{\text{max}}$ and the others to th_{max}.

In all of the following analysis experiments, we decided to use the overall accuracy of the neural network as a figure of merit, measured by running inference on the complete testset (10,000 images, as used during training to validate accuracy) in the error-injected system. We are evaluating two different topologies, namely CNV (trained on CIFAR-10) and LFC (trained on MNIST), as introduced in FINN [15]. The networks are implemented with different precisions for weights activations to analyze accuracy and error tolerance with the precision. LFC is a multi-layer perceptron (MLP) with 3 fully connected layers while CNV is a CNN with 6 convolutional layers, 3 max pooling layers and 3 fully connected layers.

As shown in [18], neuron error (stuck-at) is one of the common error models for feed-forward artificial neural networks. This model perfectly applies to fully connected layers, in which a single error will effect a single output pixel, being the output a single dimensional feature map. Expanding on the concept to CNNs, a neuron stuck-at will affect a complete output channel of a layer. We refer to error injection instead of fault injection since we fix the activations of a whole output channel to one of the possible activations values.

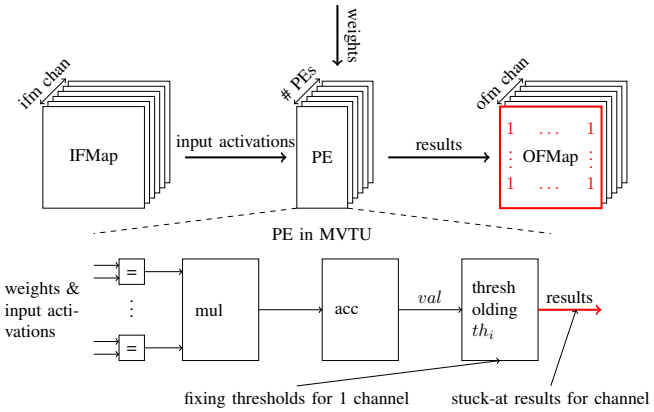

Fig. 2. Stuck-at injection for a complete channel

This means that, for multi-bit activations, multiple faults are actually injected.

V. Generic Model Optimization

A. Analysis of Stuck-At Errors

The goal of the generic model analysis is to evaluate the tolerance of the neural network itself, without considering the actual specifics of a given hardware architecture and the associated compute schedule of the layers and channels.

As explained in Section IV, we run experiments and evaluate accuracy when a complete output feature map (OFMap) channel (ofm chan) of a layer is stuck-at a fixed value, as shown in Figure 2. For each output channel a set of thresholds is evaluated. In order to perform error injection on a complete channel, it is sufficient to fix the related threshold values as explained in Section IV.

Tables I and II list the results of the injection campaign performed on LFC and CNV, respectively, at multiple precisions. For each network precision, we list the minimum accuracy achieved with one of the channel stuck-at experiments, as well as the maximum among all channels. For the fully binarized neural network, as shown in the first row of both tables, we report results of injection of s@-1 and s@1 having a binary representation of 0 and 1, respectively. When a higher bitwidth is used for activations, as in W1A2 and W2A2, we perform injection for each of the possible values of the activation. In the 2-bit activation case (actually ternary due to symmetric quantization) we inject the values -1, 0 and +1, having a binary representations of 11, 00 and 01, respectively.

From the LFC results in Table I, the overall accuracy is only slightly impacted, for all tested precisions, showing a worst-case accuracy drop of only 0.09 % in the W2A2 case for s@-1. This is mainly due to the fact that only fully connected layers are used which are inherently very redundant and able to easily tolerate a single neuron generating error results.

The situation changes significantly when the network contains convolutional layers like for CNV as shown in Table II. For the W1A1 case, accuracy will drop by 6.21% when channel 23 of layer 1 is s@1 leading to an overall worst-case accuracy (min column) of 73.01%. Here, a single channel can be identified that has a significant effect to the overall accuracy of the system. Even higher accuracy drops can be noticed at higher precisions. As expected, the higher the precision, the more accurate is the error free network. On the other hand, CNV-W1A2 shows a worst-case accuracy drop of 10.81% in layer 2, with an accuracy of 71.85% which is even lower than the worst-case for W1A1 (73.01%). This higher drop can be explained by the fact that, virtually, the error injection of s@-1 consists of 2 concurrent faults in the coding at bit level. However, increasing the weight precision to 2 bits instead achieves, as expected, a slightly better worst-case accuracy of 74.18% (drop of 10.11%) compared to CNV-W1A2.

Interestingly, in each network it is possible to identify complete channels which, when fixed to a certain value, actually show higher accuracy (max column) than the baseline. As an example, fixing the output channel 2 of layer 0 to value 1 in the W1A1 case would increase the overall accuracy of the network by 0.54%. This shows that, despite the quantization, the neural network still contains redundancy, maybe due to overfitting or limitations in training. Channel pruning could be applied to those neural networks without loss in accuracy and without the need of retraining but this is out of the scope of this paper.

It can be concluded that QNNs containing convolutional layers are by far not as robust to faults as commonly believed. A single neuron stuck-at at the wrong place can cause accuracy drops of up to 10% which may not be tolerable in a safety critical application.

B. Robustness Optimization by Selective Channel Replication

In safety critical applications, fault-tolerant systems rely on the addition of redundancy, e.g., using Triple Modular Redundancy (TMR) to achieve the desired safety integrity

TABLE I
Whole channel stuck-at effects on LFC-W1A1, LFC-W1A2 and LFC-W2A2 networks (trained on MNIST)

Net	Accuracy in %						
	Err.	single s@-1		single s@0		single s@1	
	free	min	max	min	max	min	max
W1A1	98.40	98.33	98.46	–	–	98.33	98.45
W1A2	98.49	98.44	98.55	98.45	98.54	98.43	98.55
W2A2	98.53	98.44	98.57	98.46	98.55	98.46	98.55

TABLE II
Whole channel stuck-at effects on CNV-W1A1, CNV-W1A2 and CNV-W2A2 networks (trained on CIFAR-10)

Net	Accuracy in %						
	Err.	single s@-1		single s@0		single s@1	
	free	min	max	min	max	min	max
W1A1	79.22	75.30	79.76	–	–	73.01	79.69
W1A2	82.66	73.81	83.24	79.91	83.18	71.85	83.11
W2A2	84.29	74.80	84.68	82.44	84.69	74.18	84.76

TABLE III
NUMBER OF CHANNELS TO BE TRIPLICATED AND OPERATION OVERHEAD IN ORDER TO ACHIEVE A CERTAIN WORST-CASE ACCURACY DROP

				CNV-W1A1			CNV-W1A2			CNV-W2A2			CNV-W4A4		
	layer	type	channels	$\geq 0.5\%$	$\geq 1\%$	$\geq 2\%$	$\geq 0.5\%$	$\geq 1\%$	$\geq 2\%$	$\geq 0.5\%$	$\geq 1\%$	$\geq 2\%$	$\geq 0.5\%$	$\geq 1\%$	$\geq 2\%$
#channels to trip.	0	conv	64	17	7	2	12	8	1	19	10	3	5	0	0
	1	conv	64	63	51	24	64	59	38	64	61	38	62	57	33
	2	conv	128	106	80	35	121	109	71	120	106	73	114	96	67
	3	conv	128	113	75	9	125	116	82	124	115	80	123	116	93
	4	conv	256	87	8	0	168	58	3	142	36	1	160	37	3
	5	fc	256	0	0	0	0	0	0	0	0	0	0	0	0
	6	fc	512	0	0	0	0	0	0	0	0	0	0	0	0
	7	fc	512	0	0	0	0	0	0	0	0	0	0	0	0
	ops overhead when trip. req. ch. [%]			173.47	129.70	49.87	186.24	167.65	109.49	185.24	168.86	109.36	179.63	159.60	104.96

level. Clearly, TMR adds an overhead of 200%. To avoid this, we propose selective channel replication as a method to increase the error tolerance by triplicating only the critical channels. This is performed by first identifying the channels of the neural network that cause accuracy drops which are below a certain threshold. Next, only those channels are replicated using TMR.

Table III lists, for multiple network precision bit widths, how many channels, when affected by a channel s@, would incur a worst-case accuracy drop of more than 0.5%, 1% or 2% in presence of a single error. The last row shows the percentage of operational overhead (counted in number of multiply and accumulation operations) in case of triplicating the channels which would ensure a worst-case accuracy drop that is lower or equal to the given thresholds. As it can be noticed, the lower the tolerated worst-case drop, the higher the amount of overhead, but in all cases the overhead is always smaller than the 200% given for a full implementation of TMR. Giving the results of the error injection, it is possible to evaluate how many channels need TMR in order to achieve a desired single-error worst-case accuracy, leading to minimal increase in the total number of operations.

C. Cost Analysis of Selective Channel Replication

In this section, we analyze the trade-off between accuracy and hardware cost across a spectrum of precisions. For this, we adopted the hardware cost model for High-Level Synthesis designs for MAC blocks, as explained in [16], using the formula $1.6 \cdot w \cdot a$, where w and a are, again, the bit widths for weights and activations, respectively. The resulting design space is shown in Figure 3. For each precision, the relationship between the hardware cost and the worst-case error rate in presence of a single error is shown. The optimal compromises in the design space will be located on the Pareto frontier.

Interestingly, each one of the precisions in the CNV networks on CIFAR-10 have entry points into the Pareto optimal curve. If the tolerated worst-case error rate is higher than 21.8%, the completely binarized solution is the optimal solution. However, when more channels in the binarized network require triplication to achieve the desired worst-case error rate, then the network with 2 activation bits becomes optimal. As expected, for high worst-case accuracy requirements, the solutions with higher precision for both weights and activations provide the best (and only) solution.

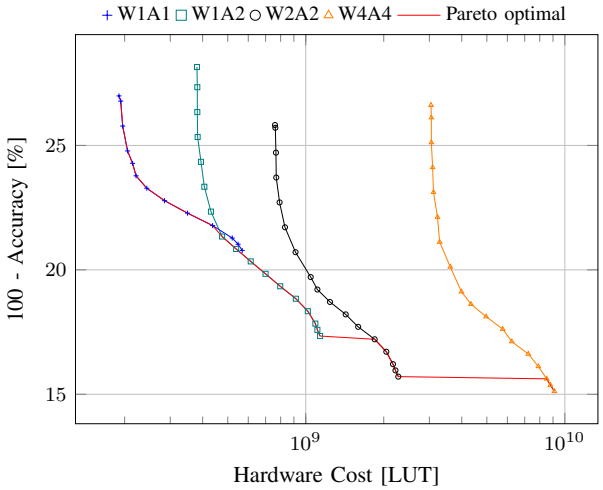

Fig. 3. Pareto frontier of single error-tolerant worst-case test error vs. hardware cost

VI. HARDWARE-SPECIFIC OPTIMIZATION

Section V showed results of error injection and analyzed the cost of selective triplication. The study was based on the assumption that the neural network accelerator uses parallelism enabling a permanent fault in a PE to affect only a limited portion of a single channel in a single layer. Nonetheless, in an overlay architecture, it is more likely that a portion of the same layer or multiple layers are physically executed on the same PE. Thus, the presence of a single fault in that PE would affect multiple aspects of the complete neural network, depending on the actual scheduling of the neural network on the array of PEs. In this section, we try to explore the effects of a single defective PE evaluating multiple portions of a layer as shown in Figure 1, first by analyzing the scheduling proposed in [15].

A. Analysis of Stuck-At Errors

In the default schedule of FINN [15], consecutive channels are computed by consecutive PEs. Channel c is computed in PE $= c \mod \#\text{PE}$. In case of a folding factor $f = \#\text{Channels}/\#\text{PEs} = 2$, with 64 channels and 32 PEs, the first PE will compute channel 0 and channel 32, while the second

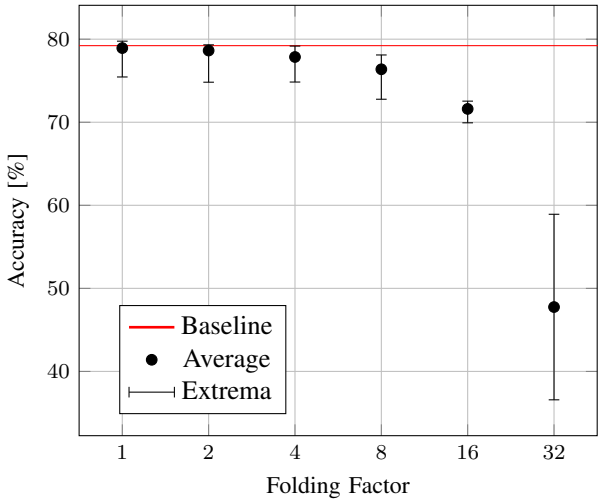

Fig. 4. Worst-case accuracy with different folding factors on CNV-W1A1 showing layer 0 stuck-at 0 for 64 channels and 64…2 PEs.

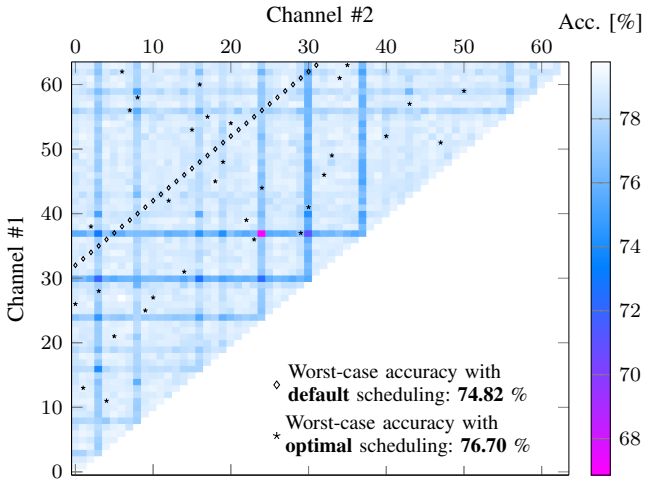

Fig. 5. Analysis of different mappings of the computations of two channels to one faulty PE in layer 0 using a folding factor of 2 (32 PEs, 64 channels).

PE will compute channels 1 and 33 (and so on). Figure 4 shows the accuracy whenever one PE is s@0 in layer 0 of the CNV-W1A1 network and assuming different folding factors for scheduling, listing both the average accuracy among all faulty PEs as well as the difference between the maximum and minimum achieved accuracy. As expected, when increasing the folding factor (decreasing the number of PEs) the average accuracy drop increases as a larger portion of the layer result being corrupt by the faulty PE. Furthermore, the accuracy drop depends on which specific channels are executed on the faulty PE, making the worst-case accuracy results being schedule-dependent. It is possible to minimize overall accuracy drop by finding and using the optimal scheduling of channels for a given number of PEs through this analysis as will be analyzed in greater detail below.

B. Robustness Optimization by Scheduling

As explained above, it is necessary to analyze the effects when combinations of output channels are faulty. To illustrate this, Figure 5 shows the achieved accuracy when one PE for two output channels (folding factor $f = 2$) is s@0, here evaluated for layer 0 of CNV-W1A1. For each combination of two channels, which are assumed to be executed on the same PE, the resulting accuracy in case of a single faulty PE is presented, having the first channel on the abscissa and the second channel on the ordinate. The worst-case accuracy in case of a single faulty PE with the default scheduling is 74.82%, indicated with $\diamond$ in Figure 5. That would be the case if PE_{30} computing channels 30 and 62 is s@0. By analyzing the complete 2-D space in Figure 5, it is possible to identify that mapping the calculation of channels 24 and 38 onto the same PE would lead to the even worse single-error accuracy of only 66.86%.

As it is possible to simply reorder the channels, this allows to optimize the schedule for higher error-tolerance. To be more precise, it is possible to find the optimal scheduling for providing the most error-tolerant behaviour towards single corrupt PEs. For that, the following Integer Linear Programming (ILP) model was used to find the schedule that maximizes the accuracy, which is explained in the following:

$$\text{maximize } \min_{\text{acc}}$$

subject to

$$\text{C1:} \quad \sum_{j=0, j\neq i}^{N-1} s_{\min(i,j),\max(i,j)} = 1 \quad \forall\ i = 0 \ldots N-1$$

$$\text{C2:} \quad (1 - s_{ij}) M_{\text{acc}} + s_{i,j} E_{i,j} \geq \min_{\text{acc}} \quad \forall\ i = 0 \ldots N-1, j = 0 \ldots i-1$$

$$\min_{\text{acc}} \in \mathbb{R},\ s_{i,j} \in \{0, 1\}$$

The formulation considers a single corrupt PE using the data from the combined error injection analysis and maximizes the minimal accuracy $\min_{\text{acc}}$ when a PE is corrupt. For that, the Boolean variable $s_{i,j}$ is used, which is one, when channel i and j out of N channels are scheduled on the same PE. As $s_{i,j}$ is equal to $s_{j,i}$ per definition, we define $i < j$ to avoid redundancy and to achieve a less complex formulation. Now, any channel i can be scheduled onto a PE with any other channel $j \neq i$, but every channel can only be scheduled once. This is realized by constraint C1. Here $\min(i, j)$ and $\max(i, j)$ are used to achieve $i < j$. Constraint C2 formulates the minimal accuracy for the scheduling. The constant $E_{i,j}$ is the accuracy, when channel i and j are scheduled on the same PE and that PE is defect. M_{acc} is the highest accuracy the network can achieve when a PE in the layer is corrupt. When the variable $s_{i,j}$ is zero, C2 is fulfilled because $M_{\text{acc}} \geq \min_{\text{acc}}$. When $s_{i,j}$ is one, which means that channel i and j are scheduled on the same PE, the constraint becomes $E_{ij} \geq \min_{\text{acc}}$ which prevents $\min_{\text{acc}}$ from getting lower than possible. Note that $E_{i,j}$ and M_{acc} are the number of correct classifications of the evaluation set with 10.000 images. If we want to maximize the accuracy for s@0 and s@1 together, we can set $E_{i,j}$ to $\min(E^0_{i,j}, E^1_{i,j})$, where $E^0_{i,j}$ and $E^1_{i,j}$ are the

s@0 and s@1 accuracies, respectively. The given formulation is for a folding factor of 2 but can easily be adopted for higher folding factors.

The resulting mapping of channels to PEs of the optimal schedule are marked with a $\star$ in Figure 5, while the default schedule is marked with a $\diamond$. The optimal schedule achieves a worst-case accuracy of 76.70%, which is an increase of 1.88% with respect to the default scheduling. Reordering the scheduling of channels just means rearranging the weights of the network and comes at no additional hardware overhead, making it an appealing solution. Nonetheless, there is no easy way to estimate the accuracy for each combination of channels mapped on a faulty PE.

The analysis relies on time consuming error injection campaigns, in which each possible combination of channels is s@ each possible value. The number of experiments also heavily increases with the number of channels c and folding factor f:

$$\#\text{experiments} = \binom{c}{f} = \frac{c!}{f! \cdot (c-f)!}$$

For example, with $f = 4$ and $c = 64$, 635,276 experiments are needed for each injected value. FPGA hardware acceleration, in case of W1A1 using a Xilinx ZCU104 board (XCZU7EV) running at 300 MHz, achieves 32,880 frames per second. This implementation enables $\sim$3 experiments per second, finishing the characterization in $\sim$59 hours. This is 5.8$\times$ faster than running Theano using an NVIDIA P40 where the overall campaign took more then 14 days.

To conclude, the worst-case accuracy can be further increased by additionally using the Selective Channel Replication method proposed in Section V-B. However, when the PEs computing M out of the N channels are replicated, the scheduling should be adjusted to consider only the remaining $N - M$ channels.

VII. Conclusions

In this paper, we presented a methodology to characterize and systematically increase the robustness of QNNs with an FPGA-accelerated error injection analysis and two orthogonal methods for robustness optimization. The FPGA-accelerated error characterization enables the fast identification of the most susceptible channels. Here, our experiments showed that QNNs with convolutional layers are less robust to single faults than commonly believed. As a countermeasure, the information retrieved from the analysis can be used to decide which channels have to be replicated to achieve a given level of error-tolerance in the generic model. This allows to trade robustness against hardware complexity leading to significant resource reductions compared to common TMR schemes. In the common case when PEs compute multiple channels, we showed how scheduling influences the robustness and proposed an ILP model that finds the optimal schedule regarding single-fault robustness without any hardware overhead. Future work will analyze modeling of hardware errors during training to train error-tolerant neural networks, as well as generalizing the injection tool to model single-event upsets (SEU).

References

[1] W. Goble and A. Brombacher, "Using a failure modes, effects and diagnostic analysis (fmeda) to measure diagnostic coverage in programmable electronic systems," *Reliability Engineering & System Safety*, vol. 66, no. 2, pp. 145 – 148, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832099000319

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[3] B. Wu, A. Wan, X. Yue, P. H. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," *CoRR*, vol. abs/1711.08141, 2017. [Online]. Available: http://arxiv.org/abs/1711.08141

[4] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[5] S. Jha, T. Tsai, S. Hari, M. Sullivan, Z. Kalbarczyk, S. Keckler, and R. Iyer, "Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors," in *Proceedings of the 3rd IEEE International Workshop on Automotive Reliability & Test*, ser. ARC '18. IEEE, 2018, pp. 1–18.

[6] A. Bosio and E. Sanchez, "Deep neural networks reliability: A case study," in *Proceedings of the 3rd IEEE International Workshop on Automotive Reliability & Test*, ser. ARC '18. IEEE, 2018, pp. 1–18.

[7] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 3–18.

[8] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, June 2018, pp. 1–6.

[9] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 8:1–8:12. [Online]. Available: http://doi.acm.org/10.1145/3126908.3126964

[10] F. F. d. Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three gpu architectures," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, June 2017, pp. 169–176.

[11] J. L. Nunez-Yanez, "Energy proportional neural network inference with adaptive voltage and frequency scaling," *IEEE Transactions on Computers*, pp. 1–1, 2018.

[12] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA Based Neural Network Accelerator," *arXiv.org*, Dec. 2017.

[13] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural Computing and Applications*, Oct 2018. [Online]. Available: https://doi.org/10.1007/s00521-018-3761-1

[14] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 56–39, Jul. 2018.

[15] 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*. ACM, 2017, pp. 65–74.

[16] M. Blott, T. Preusser, N. Fraser, G. Gambardella, K. O'Brien, and Y. Umuroglu, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," 2018.

[17] [Online]. Available: https://github.com/Xilinx/BNN-PYNQ/

[18] A. P. Singh, P. Chandra, and C. S. Rai, "Fault models for neural hardware," in *2009 First International Conference on Advances in System Testing and Validation Lifecycle*, Sep. 2009, pp. 7–12.