Fault Tolerant Neural Network Accelerators with Selective TMR


Published in:
*IEEE DESIGN & TEST OF COMPUTERS*

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
[Link to publication record in Queen's University Belfast Research Portal](#)

Publisher rights
Copyright 2022, IEEE.
This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Open Access
This research has been made openly available by Queen's academics and its Open Research team. We would love to hear how access to this research benefits you. – Share your feedback with us: [http://go.qub.ac.uk/oa-feedback](http://go.qub.ac.uk/oa-feedback)
Fault Tolerant Neural Network Accelerators with Selective TMR

Timoteo García Bertoa∗†, Giulio Gambardella∗, Nicholas J. Fraser∗, Michaela Blott∗, John McAllister†
∗Xilinx Research Labs, Dublin, Ireland, Email: {timoteog, giuliog, nfraser, mblott}@xilinx.com
†Queen’s University Belfast, Belfast, UK, Email: {tgarcibertoa01, jp.mcallister}@qub.ac.uk

Abstract—Neural networks are a popular choice to accurately perform complex classification tasks. In edge applications, neural network inference is accelerated on embedded hardware platforms, which often utilise FPGA-based architectures, due to their low-power and flexible parallelism. A significant amount of applications require resilient hardware against faults, being compliant to safety standards. In this work, we present Selective TMR, an automated tool which analyses the sensitivity of computations within neural network inference to the overall network accuracy. The tool then triplicates the most sensitive computations, which increases the functional safety of the neural network accelerator without resorting to full triple modular redundancy (TMR). As a result, this allows designers to explore trade-off between accelerator reliability and hardware cost. In some cases, we see an improvement in 24% minimum accuracy under a single stuck@ hardware fault, while increasing the overall resource footprint by 56%.

Index Terms—neural networks, safety, redundancy, automotive, FPGA, binary neural networks, FINN

I. INTRODUCTION

The fast growth of novel embedded heterogeneous architectures in the last decade opened doors to machine learning acceleration targeting various fields and applications, some of which are categorised as safety-critical, such as self-driving cars, flight-control systems or aerospace. The supervision and diagnosis of faults in these kind of applications is of great importance to prevent malfunctions or total system failure, hence requiring a high level of reliability.

A common approach to achieve fault tolerance in hardware is the use of triple modular redundancy (TMR), performing a majority vote on triplicated modules. However, machine learning accelerators are extremely power and resource hungry, making TMR an undesirable solution, due to a 200% hardware overhead. It should also be considered that not all portions of a neural networks (NNs) have the same effect on the overall accuracy, thus exploring in detail the inherent feature of neural networks enables more fine-grained approaches. In this work, we explore applying redundancy only to the most critical elements within the NN hardware accelerator, opening a new design space exploration to reduce the resource usage while offering a certain degree of fault tolerance.

This paper presents Selective TMR (STMR), an automated tool to export fault tolerant NNs using field programmable gate array (FPGA) based hardware. The main features of this tool are:

• It applies redundancy onto computations which, in presence of single-faults, have a greater impact on the overall accuracy of the network, achieving a fault tolerance for a highly reduced hardware cost;
• It preserves the throughput of the neural network accelerator, with negligible increase in latency;
• It offers configurable fault tolerance by setting a minimum requirement of accuracy in presence of single-fault, providing control over the degree of redundancy applied to the neural network; and
• It provides layer-level information of single-fault detection to the host, for further fault correction at system level (e.g., scrubbing).

Next sections introduce related work and required background leading to a Selective TMR model, explanation of the tool-flow and a detailed analysis based on a use-case.

II. RELATED WORK

Prior works aiming to improve the reliability of neural network inference fall into the following categories:

(a) hardware redundancy, such as TMR;
(b) numerical redundancy, such as error correction code (ECC) and algorithm-based fault tolerance (ABFT); and
(c) neural network training for robustness, such as fault-aware training (FAT) [1].

These approaches are often used in combination to create a reliable system. The approach described in this paper is a variant on (a), while presenting results when (c) is also applied. However, before discussing prior works’ methods to improve the robustness of NN accelerators, it’s worthwhile to review works which attempt to analyse the reliability of NNs without any particular fault mitigation techniques being applied.

In particular, Gambardella et al. [2] presented an error injection methodology to evaluate fault tolerance in NNs. They observed that single-faults onto single neurons at specific locations in convolutional layers can cause accuracy drops of up to 10%. To mitigate this, the authors propose a Selective TMR scheme and provide analysis of potential resultant reliability and hardware cost based on models. Similarly, Libano et al. [3] explored how the reliability of NN accelerators changes with various network and accelerator parameters. They show that reliability improves with reduced data precision and increased parallelism. More recently, Xu et al. [4] analyses the probability of system exceptions of FPGA-based accelerators due to hardware faults. The authors find that system exceptions can dominate the reliability of the system, and they evaluate
full TMR as a potential solution. In addition to the works above, Brosser et al. [5] show that periodic scrubbing (re-programming), is vitally important when FPGAs are deployed in low earth orbit (LEO) or geostationary earth orbit (GEO) satellites.

Closer to this work, several works have proposed methods to apply full or partial TMR to neural network accelerators. For example, Wang et al. [6] implemented full TMR on a custom light-weight CNN topology. Their approach significantly improved the error rate (33.59% error rate reduction) although they incur a large increase in hardware resources. An example of partial TMR, Libano et al. [7] proposed Selective Hardening, where they triplicated the most sensitive layers and validated the approach by means of neutron radiation testing, showing high fault masking (40%) with marginal hardware overhead (8%). SHIELDeNN [8] also proposed a framework applying partial TMR to the weights within sensitive NN layers. The authors showed an improvement in error resilience while incurring only little hardware overhead. Finally, Spyrou et al. [9] employed both partial TMR (to the output layer of the NN) and a training technique to improve the robustness of spiking neural networks (SNNs).

Zhao et al. [10] is an example of ABFT. The authors proposed 4 different ABFT schemes which protect against single-event-effects (SEEs) in convolutional layers. Together, the schemes provide effective fault-tolerance, while incurring little (8%) runtime overhead.

In this work, we propose fault tolerant NN accelerators leveraging parallelism using FPGAs for acceleration (as per suggested in [3]), targeting convolutional layers (as [10]), targeting reduced hardware overhead (as per suggested in [4, 6]), and applying partial TMR, as [7–9], but with a finer grained approach to Selective Hardening or SHIELDeNN, as we analyse and triplicate individual channels within a layer, instead of the entire layer. This work could be considered to be an extension to Gambardella et al. [2], developing the ideas proposed in the work into a tool which generates reliable accelerators.

III. BACKGROUND

STMR builds on three main prior works: FINN [11], error injection [2] and FAT [1]. In particular, we utilise the error injection campaign proposed by Gambardella et al. [2] with the specific aim to identify sensitive computations within the neural network. We also extend FINN to support detecting and correcting single-faults on selected output channels. Finally, we leverage FAT to compare the effectiveness of STMR on quantized neural networks (QNNs) which are trained to be resilient to faults to networks trained with standard training (SAT) techniques.

A. FINN framework

FINN [11] is an end-to-end framework which enables deployment of QNNs into FPGA-based hardware platforms, especially focusing on extreme reduced precisions down to binary neural networks (BNNs), leveraging heavily quantised weights and activations enabling their storage into on-chip memory and reducing the compute hardware cost. The architecture of machine learning accelerators built using FINN consists of a sequence of layers with dedicated processing elements (PEs), offering a high level of configurability to target specific throughput for a sub-microsecond latency. The FINN toolchain allows generating dataflow architectures configured for different low-precisions, network topologies or datasets.

For the scope of STMR, it is important to remark that FINN-based accelerators can be deployed onto FPGAs without an accompanying CPU, thus requiring redundancy to be implemented during inference rather than at algorithmic level. For TMR implementation in FINN, special attention is put onto the structure of the convolutional layer.

Each convolutional layer in FINN receives an input feature map (IFM) and produces an output feature map (OFM). The IFM is first transformed into an input matrix by a sliding window unit (SWU), which lowers the convolution to a matrix multiplication on-the-fly, by means of im2col. From here, the matrix multiplication is mapped to the main computational primitive in FINN: the matrix vector threshold unit (MVTU). The MVTU calculates matrix-vector products, and the subsequent quantised activations by means of thresholding.

The OFM number of channels ($C_{OFM}$) for each layer is determined by the number of filters used during convolution, and the level of parallelism to compute simultaneous OFM channels is determined by the number of PEs utilised within the layer. If the number of PEs, $N_{PE}$, is less than the number of OFM channels, i.e., $N_{PE} < C_{OFM}$, then each PE computes multiple output channels. The amount of OFM channels each PE computes is known as the neuron folding factor, and is given by: $F_n = C_{OFM} / N_{PE}$, where $C_{OFM} \% N_{PE} = 0$. The specific channels that the $i^{th}$ PE in the MVTU computes is given by:

$$C_{PE_i} = (i + j N_{PE})_{j=0,...,F_n-1}.$$  (1)

This property plays an important role determining the schedule described in Section IV-A.

B. Error injection methodology

Error injection is crucial to understand how the NN behaves in presence of faults, and how different faults impact the overall accuracy. Gambardella et al. [2] presented an error injection methodology to evaluate the fault tolerance of NNs. This method consists of altering threshold values utilised in the MVTU at run-time, injecting activation values in the next layer. For instance, if thresholds are set to a maximum or minimum value for a single channel of a layer in a BNN, the corresponding activations are forced to a permanent value of 0 or 1. Evaluating different channels and layers in each iteration, the overall accuracy of the network is computed and reported, obtaining information of the sensitivity against single-faults for all targeted error models. It is worth noticing how the sensitivity analysis is specific to a trained NN, and a new evaluation is needed if the same NN is re-trained or if the topology changed. This error injection methodology was
adopted to apply the channel stuck-at error model to obtain channel fault tolerance analysis reports. For those readers who wish to further understand the error model and error injection framework, we refer them to Gambardella et al. [2].

C. FAT for reliable inference

The selective TMR approach presented in this paper is independent from the process of training NNs. However, the selection of the most critical portion of the NN to be triplicated heavily depends on the trained parameters and the results of the error injection.

In this context, the FAT methodology proposed by Zahid et al. [1] introduces a new error injection layer component in the network definition, enabling error models to be utilised as part of the training process. The use of FAT for training improves the resilience of the network, proving greater tolerance than networks trained with standard training (SAT) techniques for different error models and precisions, with a higher error-free accuracy and higher minimum accuracy in presence of faults. In this paper, the use-case shown considers the utilisation of trained parameters obtained using both SAT and FAT methodologies, in order to assess the implications of the training methodology to the hardware resource usage of the STMR accelerator. However, FAT is not a requirement for STMR, rather an additional resource to seek fault tolerance.

IV. Selective TMR Modelling

A. TMR mapping

The mapping of convolution computations to the MVTU in FINN is output stationary, meaning that each PE computes all outputs on one or more entire channels in an OFM. In the case where a PE calculates more than one output channel, it is imperative that each triplicated channel in a triplet is calculated on a separate PE, in order to avoid a single point of failure. This is achieved in STMR by ensuring that the replicated channels in the triplet are each mapped to a unique PE, assuming there are three or more PEs. Specifically, this is achieved by placing the replicated channels immediately after the position of the original channel. This is shown in Figure 1a, where 3 of the 6 channels (in positions 0, 2 and 4) are identified as being sensitive to faults. After STMR mapping, the number of channels increases to 12, where channel 0 turns into a triplet of channels in positions 0, 1, and 2; channel 2 into a triplet with positions 4, 5 and 6; and channel 4 into a triplet with positions 8, 9 and 10.

B. TMR majority vote

Majority vote is implemented by an additional hardware component placed right after the MVTU or convolutional layer in FINN, named TMR check (TMRC). TMRC receives an OFM with triplicated channels as input, and outputs an OFM with valid results, as well as providing status information to the host through two flags: an error detected flag and an error corrected flag. In essence, this unit performs comparison for each triplet. Three possible scenarios could occur during the process, each case represented in Figure 1b with triplets T1, T2 and T3 respectively:

1) All three channels are identical. In this situation, the result is assumed as valid and forwarded to the output, with no errors flagged.

2) Two channels are identical and one has a different value. In this situation, the value present twice is considered valid and forwarded to the output, thus detecting and tolerating the single error. The error detected and error corrected flags are also raised.

3) All three channels are different. In this situation, the result of the first channel is selected by default as valid and forwarded to the output. The error detected flag is raised, but the error corrected flag is not.

Additionally, TMRC is configurable to define input and activation precisions, number of triplicated channels, OFM dimensions or redundancy factor, assumed as 3 in this work.

V. Selective TMR Tool Flow

The STMR tool aims at automatically generating a hardware accelerator which achieves a desired level of single-fault tolerance.

To do so, our co-design flow firstly triplicates parameters of explored sensitive channels of each layer (software), secondly
mapping these, leveraging high level synthesis (HLS) for the model transformations (hardware). This re-defines the NN model, with the advantage of computation-level redundancy for a low cost in resources.

As illustrated in Figure 2, the tool requires as input:
1) Trained parameters, including the complete NN topology definition. It should be noted how there’s no restriction on the training methodology adopted, and training is completely independent from STMR implementation.
2) A sensitivity report, as a result of the error injection campaign described in Section III-B performed on the target NN.
3) A minimum accuracy requirement in presence of single-fault, which the user can specify depending on the target fault tolerance requirement. Redundancy is regulated as a consequence of minimum accuracy choice, which aims to act as a fault tolerance guarantee, limited by the FPGA capacity.

Given the inputs, an automated process exports the fault tolerant neural network accelerator, relying on a series of Python scripts included in FINN and the HLS library which leverage the Xilinx toolchain to build and produce the final hardware accelerator. Figure 2 illustrates the steps of this process, which starts with a Python based three-step procedure:
1) Analysis of the sensitivity report obtained from the error injection stage, selecting which channels of each layer will be triplicated based on the required minimum accuracy in presence of faults.
2) Validation of the parallelism, namely the number of PEs used for each layer of the network, ensuring the TMR mapping described in Section IV-A is viable for the list of critical channels previously selected.
3) Automatic export of the weights and parameters, including the triplication of the values corresponding to each triplicated channel.

The process continues with the generation of the QNN accelerator itself. The FINN HLS components are generated for all layers within the network, with some layers augmented with triplicated channels and a TMRC layer (described in Section IV-B) placed after them. Once the HLS is generated for the fault tolerant network, Vivado HLS synthesises it, exporting an intellectual property (IP) block which can be integrated and built within a block design in Vivado.

VI. FAULT TOLERANT NETWORKS WITH SELECTIVE TMR

The STMR tool has been tested and validated on a set of QNNs inspired by BinaryNet [12], which consist of 6 convolutional layers, 2 max pool layers and 3 fully connected layers, called CNV and first proposed by Umuroglu et al. [11]. The convolutional neural networks (CNNs) have been trained on the CIFAR-10 dataset to classify images among 10 classes with reduced precisions, and referred to as CNVW1A1, CNVW1A2 or CNVW2A2, where the number after W and A are the bitwidths of weights and activations respectively.

A. STMR use-case

Let us consider a SAT-trained CNVW1A1 network whose computed error-free accuracy is 84.46%. The results of the error injection campaign performed over this network to analyse its fault sensitivity are collected in Table I, where clearly the convolutional layers (0 to 5) experience a higher accuracy drop when compared to the fully connected layers (6 and 7). More specifically, layer 1 experiences a minimum accuracy of 59.28% when a single channel was stuck at 1.

Setting the STMR minimum accuracy requirement to 83%, and using the automated flow to generate a fault tolerant accelerator, it requires 16, 50, 92, 56 and 7 triplicated channels in the first five convolutional layers. This leads to a total of 221 triplications (≈ 12% of the total channels in the network), and the implementation of one TMRC layer per each convolutional layer containing redundancy.

The same CNVW1A1 network trained using FAT gives a computed error-free accuracy of 84.8%. The minimum accuracy observed during error injection is 81.08%. Targeting 83% minimum accuracy, the STMR tool exports a fault tolerant NN with only 1, 36, 2 and 6 triplicated channels in the first four convolutional layers respectively, for a total of 45 triplicated channels (≈ 2% of the total).

These results show a greater benefit when training the network using FAT, reducing considerably the redundancy required for the same minimum accuracy target.

Identical steps were followed to perform error injection and export CNV networks with other accuracy requirements and precisions, whose cost in hardware will be addressed in next sections.

B. Validation, throughput and latency

In order to validate the functionality of STMR, error injection campaign was performed into STMR exported networks. The campaign consists of injecting errors to each individual channel of each layer, including the triplicated ones which
will be treated exactly as the original. The campaign aims at confirming that the triplication performed by the STMR tool and implemented in the hardware accelerator is able to guarantee error-free accuracy in presence of faults in one of the triplicated channels. When injecting single-faults into such channels, majority vote will tolerate the fault and the computed accuracy will be equal to the error-free accuracy, while the error detection and error correction flags will be raised by the hardware accelerator.

For instance, the result of this error injection campaign for the network example in Section VI-A, which had implemented STMR for a minimum accuracy of 83%, are shown in Table I, where, compared to the error injection results, clearly the minimum accuracy for the first five convolutional layers is kept above the 83% threshold, thus validating the STMR implementation.

The parallelism of STMR networks cannot be exactly preserved, as the number of PEs vary depending on the number of triplications and folding requirements as explained in Section IV-A. However, having a control over the parallelism for each convolutional layer for the previous example, a throughput of $\approx 21k$ frames per second (FPS) is preserved for both SAT and FAT trained networks, while leveraging the benefits of STMR. Latency experiences a slight increase due to the inclusion of TMRC layers in the network. This increase is negligible, as TMRC has been fully unrolled to perform majority vote for one pixel in one clock cycle. In the case of SAT-STMR-CNWW1A1, the addition of 221 triplications produces a latency increase from 142.3 to 145.2 microseconds, whereas the addition of 45 triplications for the FAT-STMR-CNWW1A1 case causes an increase from 142.3 to 143.3 microseconds.

### C. Hardware Resource Analysis

Area savings of STMR is one of the key reasons and motivation of this research, enabling designers to explore the design space of hardware usage versus fault tolerance. The hardware usage increase when applying STMR to a FINN accelerator can be divided in two main components: 1) Majority voter (TMRC). 2) Increased PEs to compute triplicated channels. TMRC only requires look-up tables (LUTs) and flip-flops (FFs), with a resource usage linearly dependent on activation precision and number of OFM channels (including triplications).

In all our examples, the resource usage of TMRC was less than $<7.5\%$ of the resources of the corresponding MVTU. Compared to the resource overhead of triplicating several OFM channels, this is relatively low.

The majority of hardware cost increase depends on the compute part, as well as BRAM usage increase due to partial triplication of parameters. Considering as baseline the CNWV1A1 example, LUTs, FFs and 36kb BRAM utilisation are 45.8k, 60.6k and 147.5 respectively. Hence, a full TMR solution would require $\geq 137.4k$ LUTs, $\geq 181.9k$ FFs and $\geq 442.5$ BRAMs. However, if we leverage the STMR tool to export a fault tolerant CNWV1A1 network trained using SAT targeting a minimum accuracy of 83% in presence of faults, resource utilisation is 71.5k LUTs, 91.9k FFs and 210.5 BRAMs. This means an additional cost of 56% in LUTs, compared to the theoretical $\geq 200\%$ for full TMR, while guaranteeing a maximum drop in accuracy of $<1\%$, increasing by 23.72% the minimum accuracy under single-faults.

When using the FAT trained NN with the same target accuracy in presence of faults, we experienced a resource utilisation of 55.2k LUTs and 70.5k FFs, resulting in an increase utilisation of 21% in LUTs, compared to the theoretical $\geq 200\%$ for full TMR and 56% for SAT-STMR. Additionally, optimisation of BRAM utilisation mapped to LUTs used as memory is also leveraged by the synthesis tools, as in this case only 147.5 BRAMs were utilised.

The full set of results are collected in Table II, presenting the overall hardware cost for CNWV1A1, CNWV2A1 and CNWV2A2 accelerators implemented with STMR. The target accuracy drop in presence of single-fault has been set to be less than 2% for all precisions, and the results have been reported using both SAT and FAT training methodologies. The reader can observe great area savings when comparing the hardware cost of a full TMR approach versus the STMR solution proposed. Also, it can be observed that the error-free accuracy and hardware cost increase with higher precisions.

The complete design space of hardware cost versus worst case classification error (i.e., $1 -$ minimum accuracy under a single-fault) against utilisation in LUTs is shown in Figure 3,
TABLE II: Selective TMR hardware cost when targeting < 2% accuracy drop and throughput of 21 kFPS

<table>
<thead>
<tr>
<th>Network: CNVW1A1</th>
<th>Network: CNVW1A2</th>
<th>Network: CNVW2A2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NO TMR</td>
<td>FULL TMR</td>
</tr>
<tr>
<td>SAT</td>
<td>SAT</td>
<td>SAT</td>
</tr>
<tr>
<td>SAT</td>
<td>SAT</td>
<td>SAT</td>
</tr>
<tr>
<td>Measured single-fault minimum accuracy [%]</td>
<td>59.28</td>
<td>81.08</td>
</tr>
<tr>
<td>Targeted single-fault minimum accuracy [%]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Number of triplications</td>
<td>-</td>
<td>1920</td>
</tr>
<tr>
<td>CLB LUTs [x10^3]</td>
<td>45.8</td>
<td>137.4</td>
</tr>
<tr>
<td>Targeted LUTs increase [%]</td>
<td>≥137.4</td>
<td>243.4</td>
</tr>
<tr>
<td>LUTs increase [%]</td>
<td>≥200</td>
<td>200</td>
</tr>
<tr>
<td>BRAM increase [%]</td>
<td>-</td>
<td>≥200</td>
</tr>
</tbody>
</table>

![Fig. 3: Design space of worst-case error vs. hardware cost for CNVW1A1, CNVW1A2 and CNVW2A2 with throughput of ≈ 21 kFPS](image)

where full TMR solutions marked in the graph clearly offer the minimum worst case error, but for the maximum hardware cost, and solutions without redundancy show a high worst error case which do not guarantee high levels of accuracy in presence of faults. The intermediate results define the design space for STMR fault tolerant solutions considering different minimum accuracy requirements, where the hardware cost is highly reduced. The most attractive solutions shown in this figure respond to the intermediate datapoints for FAT-STMR-CNVW1A1, FAT-STMR-CNVW1A2 and FAT-STMR-CNVW2A2 generated networks, being all pareto dominant to the SAT counterparts.

VII. CONCLUSIONS

In this work we presented Selective TMR, an automated tool to generate fault tolerant NNs for machine learning accelerators in FPGAs. It offers flexibility to trade-off between fault tolerance and hardware cost by applying TMR only to critical channels of the NN layers, previously identified by means of error injection. The fault tolerant accelerators automatically generated preserve their throughput and incur in very low latency increase. Additionally, experiments show how FAT in conjunction with STMR provides higher benefit than SAT trained NNs.

For future work, we plan to: validate the STMR approach under high radiation environments; develop robust training methods that are specific to STMR, in particular, methods which extend FAT to support a small number of outliers; and finally, extend the STMR tooflow to support more neural network layer types, such as long short-term memories (LSTMs).

REFERENCES


