MULTIPLY AND ACCUMULATION CIRCUITS AND FORWARD ERROR CORRECTION METHODS

TECHNICAL FIELD

The present disclosure relates to fault tolerant multiply accumulate circuits for hardware accelerators, for use in, for example, deep neural networks.

BACKGROUND

The remarkable success achieved by deep neural networks (DNNs) in classification tasks has promoted the deployment of deep learning accelerators for many edge applications such as autonomous vehicles, virtual assistants and healthcare. For safety and security critical applications, fault resiliency is a desirable property that can help to avert DNN misclassification to a certain extent. Deep learning models are inherently noise tolerant. Removing small redundant weights and quantizing data to shorter bitwidth will not apparently affect the classification accuracy. Therefore, directly presenting an adversarial input derived based on the software model of a DNN to its implementation on a hardware accelerator is unlikely to achieve the same desirable misclassification success rate. This is because the 8-bit fixed width multiply-accumulate (MAC) operators have filtered out most visually imperceptible perturbations in the lower order bits of the 32-bit input data. However, such built-in error resiliency is insufficient to combat malicious hardware-oriented attacks like faults injected into the weight parameters and the MAC units. Recent work shows that maliciously injecting faults into the datapaths of real-world DNN accelerators can produce incorrect inference on the target victim. A high attack success rate on multi-view object classification has been demonstrated by fault attacks. Therefore, it is imperative to re-examine the resilience of current defence methods against the new threats posed by fault-injection based attacks.

Existing fault tolerance studies on deep learning hardware mostly focus on yield and reliability enhancement against natural or passive failures such as radioactive impurities and manufacturing defects. Severe accuracy degradation can occur by these events due to bit flips in the high order bit positions of the data in DNN computations. Thus, reducing the data precision and limiting the activation range are

the common approaches that have been proposed to mitigate accuracy loss due to soft errors and stuck-at faults, respectively. It is worth noting that short bit-width data type and bounded data range are also common practices adopted for energy-quality scaling in modern DNN accelerator designs. Besides reliability issues, fault mitigations under low voltage operation have also been investigated. Aggressive voltage scaling is a key technique to improve energy efficiency. By lowering the supply voltage, the fault rate of the bitcells on SRAM and timing error rate on datapaths increase. A proposed system utilizes active fault detection circuit to detect and round the faulty weights towards zero to preserve the classification accuracy when the SRAM voltage is reduced. Another system applies per-layer dynamic voltage underscaling on the processing element (PE) array together with the timing error (TE)-Drop. The idea is to achieve energy saving without compromising accuracy by detecting timing errors and dropping the erroneous MAC results. Random bit-flips are the primary fault model considered in the most resilient DNN hardware designs. Mitigations against random faults in hardware components and memory elements for storing weight parameters are simulated in various software frameworks. The fault model and design strategy are based on randomly sampled MAC units with their error probability back annotated into the simulation environment. Deliberate faults introduced into MACs or data processing pipelines by adversarial attacks are, unfortunately not random, but highly biased and often sparse. Recently, a DNN engine has been demonstrated to tolerate timing error on MAC unit under aggressive voltage scaling. This is achieved by double sampling with time-borrowing technique. By borrowing the time from the next datapath in the pipeline for the signal to settle down in the MAC stage, the voltage is scaled until the zero-margin point is reached without the need for explicit error correction. Until now, the reported fault resilient DNN hardware designs only aim at recovering the classification accuracy from uniform errors or sparse faults. Most of them are developed for the purpose of aggressive energy-quality scaling instead of as countermeasures against hardware fault-injection attacks. While large-scale applications have been simulated for general fault tolerance studies, only the small scale 10-digit classification task has been evaluated on error-resilient DNN designs implemented on real-world DNN accelerator.

SUMMARY

In present disclosure, a fault-tolerant technique applicable to deep learning accelerators is proposed to thwart deliberate and malicious fault attacks. The proposed design relies on forward error compensation. The error detected on the current step of computation is forwarded to the next step for compensation. Thus, no pipeline stall, flushing and replay are required.

According to a first aspect of the present disclosure, a multiply and accumulation circuit is provided. The multiply and accumulation circuit comprises: a multiplication circuit configured to generate a multiplication output from at least two input signals; a main flip flop synchronized to a main clock signal and configured to receive the multiplication output as an input; an auxiliary flip flop synchronized to a relaxed clock signal which is delayed with respect to the main clock signal and configured to receive the multiplication output as an input; an accumulation adder configured to add a output of the multiply and accumulation circuit from a preceding clock cycle to the output of the main flip flop; a subtractor configured to determine a difference between an output of the main flip flop and an output of the auxiliary flip flop; an auxiliary adder configured to add the difference to a signal based on an output of the accumulation adder; and a multiplexer configured to generate an output of the multiply and accumulation circuit and configured to modify the output of the multiply and accumulation circuit to compensate for the difference between the output of the main flip flop and the output of the auxiliary flip flop.

The multiply and accumulation circuit provides a new effective error resilient design for deep learning accelerator. The new design validates countermeasures against fault injection attacks on actual hardware accelerator for ImageNet applications.

The proposed fault tolerant technique detects faults by a shadow register, and corrects the error in the next accumulation step without suspending the existing pipeline or requiring data replay. As no stall or flushing operation is required by the proposed correction mechanism, there is no penalty on throughput performance.

In an embodiment, the multiplexer is configured to compensate for the difference between the output of the main flip flop and the output of the auxiliary flip flop in a subsequent clock cycle.

In an embodiment, the multiply and accumulation circuit further comprises an accumulator flip flop which receives the output of the accumulation adder and wherein the auxiliary adder is configured to add the difference to an output of the accumulator flip flop.

In an embodiment the relaxed clock signal is delayed with respect to the main clock signal by a fraction of a period of the main clock signal.

In an embodiment, the multiplication circuit comprises a first flip flop and a second flip flop configured to receive as inputs respective signals of the at least two input signals, wherein the first flip flop and the second flip flop are synchronised with the main clock signal.

In an embodiment, the multiplication circuit comprises a plurality of multipliers and a chain adder configured to add the outputs of the multipliers of the plurality of multipliers.

According to a second aspect of the present disclosure, a hardware accelerator for a machine learning system comprising a multiply and accumulation circuit is provided.

According to a third aspect of the present disclosure, a forward error correction method in a multiply and accumulation circuit is provided. The method comprises: determining a difference between an output of a main flip flop and an output of an auxiliary flip flop, wherein the main flip flop is synchronized to a main clock signal and coupled to a multiplication output and the auxiliary flip flop is synchronized to a relaxed clock signal which is delayed with respect to the main clock signal and coupled to the multiplication output; and adding a signal dependent on the difference to an output of the multiply and accumulation circuit to compensate for the difference between the output of the main flip flop and the output of the auxiliary flip flop.

In an embodiment, the signal dependent on the difference is added to the output of the multiply and accumulation circuit in a subsequent clock cycle.

In an embodiment, the relaxed clock signal is delayed with respect to the main clock signal by a fraction of a period of the main clock signal.

BRIEF DESCRIPTION OF THE DRAWINGS

In the following, embodiments of the present invention will be described as non-limiting examples with reference to the accompanying drawings in which:

FIG.1 is a block diagram showing computation processes in a convolutional layer of a deep learning system;

FIG.2 is a block diagram showing a general architecture for DNN acceleration;

FIG.3A to FIG.3C are circuit diagrams showing an error detection circuit architecture;

FIG.4 is a block diagram showing the architecture of a fault injection block;

FIG.5 is a circuit diagram showing a multiply accumulator with forward error compensation according to an embodiment of the present invention;

FIG.6 shows a visualisation of zeroing-on-error, bit masking and sign-bit masking techniques under biased fault injection patterns;

FIG.7 is a block diagram showing an implementation of the multiply accumulator with forward error compensation according to an embodiment of the present invention in a deep learning accelerator;

FIG.8 shows the prediction accuracy of four error-tolerant designs and baseline design under the malicious fault-injection attack on all layers; and

FIG.9 is a graph showing prediction accuracy of the unprotected baseline design under individual layer fault injection attacks.

DETAILED DESCRIPTION

Deep learning processing is computationally intensive. The convolutional layer and fully-connected layer consume a lot of computing power due to the number of MAC operations required on the convolutions and matrix-vector multiplications. The computing process of the convolutional layer is illustrated in FIG.1.

FIG.1 is a block diagram showing an example of computation processes in a convolutional layer of a deep learning system. As shown in FIG.1, the processing of the convolution layer involves applying a 3D filter 110 which is made up of a number of (five in this figure) filters 111-115. The processing involves obtaining output features 130 from input features 120. The input features 120 may, for example, be pixels or groups of pixels of an image or video sequence. or other types of feature The processing involves multiplying a receptive field 140 by the filter 110 to obtain a product array 142. An accumulate operation is carried out on the product array to obtain an accumulation result 144. A non-linear function 146 is applied to the accumulation result 144 to obtain the output element 148.

The generation of each element y of an output feature map can be expressed as follows:

$$y = f\left(\sum_{i=1}^{K}\sum_{j=1}^{K}\sum_{c=1}^{D} w_{i,j,c} \times x_{i,j,c} + \mu\right)$$

where $f(\cdot)$ is the non-linear activation function, K and D are the height/width and depth of the filter size, w is the filter weight, x is the input feature element and μ is the bias. The generation of the entire output feature map requires many steps of convolutions by sliding each filter through the complete input feature map.

The fully-connected layer computes the output vector Y by a matrix-vector multiplication as follows:

$$Y = f(WX + \mu)$$

where W is the weight matrix and X is the input vector.

Unrolling computational layers into multiple multiply-accumulate (MAC) operations for parallel execution and reducing the precision of MAC units are standard practices to increase the throughput and energy efficiency simultaneously.

FIG.2 is a block diagram showing a general architecture for DNN acceleration. The hardware accelerator 200 comprises a host interface 210, a configuration unit 220, a data controller 222, a global buffer 224, an activation/pooling/normalization unit 226, a plurality of accumulators 230 and an array of processing engines (PEs) 240. The host interface 210 communicates with a host processor 212 which is coupled to a memory controller 214 which controls a host memory 216.

Different types of layer are mapped to specific hardware blocks for efficient processing. Specifically, the MAC operations are executed parallelly by the PE array 240, which consists of a chunk of homogeneous arithmetic units including multipliers, adders and accumulators. The PE array 240 can be implemented by customized datapath logic on ASIC or vendor-optimized DSP block on FPGA. Local memory such as first-in first-out (FIFO) unit and register file is coupled with each PE unit of the PE array 240 for better data reuse. The parallel PE array 240 is usually separated into individual voltage/clock domains to facilitate dynamic voltage frequency scaling (DVFS). Besides, other layers like non-linear activation, pooling and normalization are mapped to the single instruction multiple data (SIMD) engine. To minimize the energy consumption by external memory accesses, an adequate number of on-chip buffers and optimized data controllers are allocated to minimize off-chip communications. The layer execution sequence and model weights are loaded into the accelerator from the host processor 212 through the host interface 210.

The architecture shown in FIG.2 can be implemented by traditional RTL design flows that allow full customization of every hardware components. On the other hand, high-level synthesis (HLS) for hardware-software co-design allows the designer to integrate

the performance critical components (e.g., fixed-point MAC units in DNN computation) designed by RTL coding with other non-critical components described in high-level programming language (e.g., C/C++ and OpenCL). This approach has been widely adopted to reduce the design turnaround time and it offers a user-friendly hardware interface to DNN applications.

Hardware faults can be naturally or unintentionally introduced by imperfect manufacturing processes, high-energy particle radiation, temperature/voltage variations and aging effect. Although the deep learning model has inherent redundancy to tolerate small noises and distortions, errors occurred in high-order bit positions in internal computations can lead to significant accuracy degradation that cannot be healed by this natural noise immunity. Such susceptibility to high-order bit errors can also be introduced into the DNN circuit by some design techniques, as exemplified by the aggressive voltage scaling on SRAM and datapath. Hardware unreliability or voltage scaling creates random errors with sparse fault distribution. For instance, soft error and bit corruption of memory cell can be modelled as random bit-flips, where each bit has the same low fault probability. Combinational circuit operating in undervoltage condition can lead to biased faults.

However, the worst-case timing violations are rarely exercised. Compared with the damage induced by passive faults, a malicious attacker poses a more serious threat. As errors in high-order bits are more detrimental to the classifier's performance in the DNN accelerator, the adversary can directly inject faults onto those susceptible bits to achieve the objective of misclassifying a specific target. For instance, by making use of the hardware bug of DRAM, the attacker can intensively access the aggressor row(s) to flip the target bit(s) stored in the main memory to cause a misclassification of the DNN model running on a CPU cluster. In addition, large accuracy degradation has been observed by manipulating a small number of vulnerable bits of the model weights. Fortunately, falsification of arbitrary target bits by the aforementioned methods is still very restrictive on various hardware platforms as tampering the DRAM cell through Row-Hammer attack requires techniques to bypass the cache, knowledge about the virtual to physical address mapping from the OS and positions of the vulnerable bit cells on the deployed DRAM chips. Injecting a laser beam on exposed silicon can change the classification outcome to generate a faulty prediction. As the

opened chip package provides a clear tamper evidence, such an attack is conspicuous. Clock glitch attack, which exploits timing error on the datapath, can deceive the DNN classifier in a stealthy way. By injecting a small number of appropriately timed glitches (≤10%), high attack success rates on DNN models trained for ImageNet application can be achieved. This specific glitch attack pattern for the target object can be extended to fool the DNN classifier even in a multi-view scenario. Since the deep learning accelerator has replaced the input image as a target of the attack, deliberate and biased fault model should be considered in designing its mitigation technique.

Many fault mitigation methods have been proposed for DNN accelerators to resist soft errors and stack-at faults caused by energetic particles and hardware defects. These include analysing the propagation effect of soft errors in the DNN inference process by injecting faults randomly in a simulation framework. This indicates that using just-enough data precision can prevent disastrous accuracy drop, as bit-flips in high order positions have been limited. Based on similar observations, is has been suggested that clipping the intermediate activation to a boundary can avoid large errors from occurring. By observing the computational data, it has been concluded that the sign bit and the most significant bit (MSB) have the same logic value in 99% of the time. Therefore, masking the sign bit with the MSB can improve the computational fault tolerance under random fault distribution. The same uniform error probability model has been applied and evaluated. To mitigate the effect of weight manipulation attack on the quantized model, several software-oriented approaches including retraining the neural networks through binarization-aware and piece-wise clustering methods and weight reconstruction by averaging, quantization and clipping the errors over a grain of weights are suggested.

FIG.3A to FIG.3C are circuit diagrams showing an error detection circuit architecture. FIG.3A shows the in-situ error detection circuit architecture, FIG.3B shows a razor flip flop module and FIG.3C shows an error detection sequential (EDS) module.

To mitigate timing errors, both the error detection circuit and its error handling mechanism are required. FIG.3A illustrates the in-situ error detection circuit architecture 300. An input flip flop 310 outputs a logic state 1 330. A delay 320 outputs a delayed clock signal. The clock signal, the delayed clock signal and the logic state

1 are input into the in-situ fault detection circuit 340. The in-situ fault detection circuit 340 outputs an output logic state 380 (logic state 2).

A popular implementation called Razor Flip Flop is shown in FIG.3B. The razor flip flop 340A exploits the double sampling via a multiplexer 352 with a normal clock on the main flip flop 354 (i.e., the FF block on the top path) and a delayed clock on the shadow latch 356 (i.e., the Latch block on the bottom path). The sampling results are compared by a comparator 358 which compares the values latched by these two sequential elements. Once an error is detected by a mismatch in the latched value between the main and shadow registers, the comparator (i.e., XOR gate) will raise an error flag.

Error detection on a data bus can be achieved by feeding all the outputs from the comparators to a large OR gate 390. The original proposal of Razor includes an error correction mechanism. It is used to flush the pipeline and replay the instruction with the data from the shadow latch when an error occurs. This correction scheme introduces a nonnegligible performance overhead since the pipeline has to be stalled during the flushing and recomputing stages.

Another approach called error detection sequential (EDS) utilizes a similar double sampling design but switches the positions of the flip flop and latch. The architecture of EDS is depicted in FIG.3C. The replacement of flip flop to latch 362 in the datapath allows time borrowing from the datapath to alleviate the metastability problem. The error signal generated by the latch 362 and flip flop 364 outputs can be used to monitor the infrequently excited critical paths using a comparator 366. Consequently, the clock and voltage guard bands can be further relaxed to increase the throughput and reduce power consumption. Instruction replay at a lower clock frequency and multiple-issue instruction replay are two error recovery mechanisms introduced in the EDS design 340B. They require the use of pipeline speculation, which will again incur significant performance overhead. Both Razor and EDS are designed to detect and correct pipeline errors on the microprocessor.

The double sampling error detection has been widely suggested in DNN accelerator with aggressive voltage scaling. One method is to protect the read SRAM data by employing error detection on each column of the memory array. Suspected faulty

SRAM data will be masked by setting all the bits to zero (i.e., word masking) or replacing the affected bit(s) with the sign bit (i.e., bit masking). Rounding faulty data towards zero is equivalent to pruning a connection from the neural network architecture. Sparse DNN engine detects the timing violation on the memory loading stage and MAC unit by the use of EDS-like circuits. Apart from the SRAM data masking, time-borrowing from the accumulator in the MAC unit has substituted the expensive error correction as a means to reduce timing errors. Besides zeroing faulty weights of SRAM, dropping errant intermediate results of the MAC operation is another option. Erroneous results in MAC units are detected by the shadow flip flops operated with a delayed clock. Upon detection, the computation results of subsequent MAC stages will be dropped due to the incorrect inputs obtained from the normal flip flops of the previous MAC stage. Meanwhile, the outputs from the shadow flip flops will be sent to the next MAC, which is equivalent to bypassing one MAC operation, if an error is detected. This TE-Drop method avoids recomputing when a faulty calculation is detected, thus introducing less throughput loss on pipeline execution. A similar concept of dropping erroneous partial sums has been utilized to mitigate the effect of computational faults on the DNN classifier. An associated side effect of faulty weight masking and TE-Drop is that important (i.e., large) weights and MAC results can also be discarded. Although the removal of small weights and partial sums is tolerable in the DNN algorithm, larger weights (usually result in larger intermediate results) play a more important role than smaller weights in DNN inference accuracy.

The aforementioned methods aim at mitigating soft error, defect and timing error caused by voltage scaling under licit operation assumption. The fault model considered has sparse error distribution, which can be mitigated partially through the inherent redundancy of the deep learning model at the algorithmic level. In views of the rarely occurred and a limited number of high order bit errors under voltage scaling, the inherent error resiliency of the DNN model is often traded for energy efficiency in its hardware accelerator design. This has escalated the threats of its exploitation for malicious and conscious fault injection attacks on the deep learning accelerator. Unfortunately, only a small classification problem (i.e., 10 digits) has been evaluated with time-borrowing mitigation on actual deep learning accelerator, other mitigation methods report the only simulation instead of actual classification results obtained by physical implementation of DNN computations on a hardware accelerator.

The following section describes a fault resilient design developed and implemented on a real-world DNN accelerator. The design is able to recover most of the prediction errors under deliberate fault attacks.

Different fault injection attacks on deep learning accelerators have been reviewed in the previous section. From practicality and effectiveness points of view, invasive attacks such as laser injection that can be easily detected or result in permanent damages of device functionality or original appearance are not considered in this disclosure. We consider the malicious attack scenario where glitches can be injected into the MAC units to cause data corruption in intermediate computation results. These hardware-oriented attacks can lead to a high misclassification rate on unprotected deep learning accelerators. From a defender perspective, we assume that a determined attacker can launch vigorous fault injection attacks on all or selected layer(s) of the target DNN models. The resiliency of the hardware design refers specifically to the ability of the protected design to preserve the prediction accuracy under malicious glitch injection attacks.

FIG.4 is a block diagram showing the architecture of a fault injection block. To emulate the attack scenario, we implemented a programmable fault injection block 400 as shown in FIG.4 to apply different attack patterns to the PE array. The fault injection block 400 consists of a PLL module 410, a clock counter 420, a multiplexer 430 and an AND gate 440. A phase shifted clock 412 with adjustable duty cycle is generated from the reference kernel clock 402 by the PLL module 410. Glitch injection is achieved by switching the PE clock between the kernel clock 402 and the phase shifted clock 412 inputs of the multiplexer 430. The duration and injection rate of the glitches are determined by the clock counter's output and a glitch control signal. By programming the clock counter 420, cycle-accurate fault injection can be achieved. We integrate this fault injection block into the same hardware platform as the DNN accelerator to evaluate the fault mitigation capability of the forward error compensation scheme of our invention.

FIG.5 is a circuit diagram showing a multiply accumulator with forward error compensation according to an embodiment of the present invention. The multiply

accumulator (MAC unit) 500 comprises a multiplication circuit 510, a main flip flop 520, an accumulation adder 522, an accumulator flip flop 524, an auxiliary flip flop 530, an auxiliary subtractor 532, an auxiliary subtractor flip flop 534, an auxiliary adder 536 and a multiplexer 540.

In the MAC unit 500 shown in FIG.5, the multiplication circuit 510 comprises a first input flip flop 511, a second input flip flop 512, a third input flip flop 513 and a fourth input flip flop 514. Each of the first to fourth input flip flops is configured to receive a respective input signal which is combined with the other input signals by the multiplication circuit 510. The outputs of the first input flip flop 511 and the second input flip flop 512 are coupled to a first multiplier 515. The outputs of the third input flip flop 513 and the fourth input flip flop 514 are coupled to a second multiplier 516. The outputs of the first multiplier 515 and the second multiplier 516 are coupled to a chain adder 517. The output of the multiplication circuit 510 is the output of the chain adder 517 which corresponds to the product of the output of first input flip flop 511 and the output of the second flip flop 512 added to the product the output of third input flip flop 513 and the output of the fourth flip flop 514.

It will be appreciated that other arrangements of multiplication circuit are possible in MAC units with forward error compensation and the arrangement if the multiplication circuit 510 shown in FIG.5 is intended to be an example of a possible arrangement for the multiplication circuit. For example, in a simplified form, the multiplication circuit may comprise two input registers (flip flops) and one multiplier which takes the outputs of the input registers as inputs and provides the output of the multiplication operation as the product of the outputs of the two input registers.

The output from the multiplication circuit 510 is coupled to the main flip flop 520 and the auxiliary flip flop 530. The main flip flop 520 (and the other flip flops – the first to fourth input flip flops 511 to 514, the accumulator flip flop 524, and the auxiliary subtractor flip flop 534) are connect to the PE clock. In contrast, the auxiliary flip flop 530 is connect to a relaxed clock signal which corresponds to the PE clock period with a delay added ($clk + \Delta$) where clk is the PE clock period and Δ is the delay. The delay Δ is typically a fraction of the PE clock period and the value of Δ depends on the delay of the main path and the implementation technology.

The output from the main flip flop 520 is connected to the accumulation adder 522 which adds the previous output from the MAC unit 500 to the output from the main flip flop 520. The output from the accumulation adder 522 is input into the accumulator flip flop 524. The output from the accumulator flip flop 524 is coupled to the multiplexer 540 and also to the auxiliary adder 536.

The output from the auxiliary flip flop 530 is coupled to an input of the auxiliary subtractor 532. The second input of the auxiliary subtractor receives the output of the main flip flop 520. The subtractor 532 is configured to output a difference between the two inputs and this output is input into the auxiliary subtractor flip flop 534. The output from the auxiliary subtractor flip flop 534 is coupled to the auxiliary adder 536. The auxiliary adder 536 outputs the sum of the output of the accumulator flip flop 524 and the output of the auxiliary subtractor flip flop 534. The output from the auxiliary adder 536 is coupled to the second input of the multiplexer 540. The multiplexer 540 has a selector input labelled comp which selects the output of the MAC unit 500 from either the output of the accumulator flip flop 524 or the output of the auxiliary adder 536.

The proposed forward error compensation method is inspired by the carry-save addition scheme which has been widely adopted for multi-operand addition. Conventional carry propagation adders suffer from the worst-case long carry propagation delay from the carry generated in the least significant bit position to the most significant bit position. Carry propagation is inevitable for two operand additions. Multi-operand addition is required in partial product accumulation of digital multiplier. For multi-operand addition, instead of immediately adding the carry bit generated or propagated from the preceding full adder cell, the addition of the carry bit can be deferred to the next stage of full adder cells as long as there are more than two operands after each stage of addition. This way there is no carry propagation in each stage of carry-save addition until the last stage when only two operands are left. The delay of each carry-save addition stage is thus limited to only a single bit full-adder cell delay irrespective of the length of the operands. A similar concept can be applied to the MAC operations of DNN. If the error between the correct and corrupted intermediate result has been detected and recorded, the correction/compensation

procedure can be forwarded to the next stage without stalling or replaying the previous computation.

If a timing error occurs in the main flip flop (FF1) 520, which outputs an incorrect integer result a_1', the auxiliary flip flop (FF2) 530 can still latch the correct result a_1 with a relaxed clock. The erroneous result a_1' will propagate through the accumulator when it is added with a_0 to form a_2'. The latter is different from the correct output $a_2 = a_1 + a_0$. To correct the error, the difference $d_1 = a_1 - a_1'$ can be forwarded to the next cycle and compensated on a_2' by the auxiliary adder 536. It can be verified that the error-compensated result $a_2' + d_1$ is equivalent to the correct result of a_2 by the following:

$$a_2' + d_1 = a_1' + a_0 + d_1 = a_1' + a_0 + a_1 - a_1' = a_1 + a_0 = a_2$$

The comp signal which controls the output of the multiplexer 540 is generated by an error flag detected by the double sampling technique after one clock cycle. The auxiliary subtractor 532 and the auxiliary adder 536 are relatively small compared with the multipliers of the MAC unit. The basic architecture of one cycle of error forwarding can be extended to multiple-cycle error forwarding. By placing the auxiliary adder 536 next to the auxiliary subtractor 532, the difference detected in multiple stages can be accumulated on the output of the auxiliary subtractor flip flop (FF4) 534. Once the MAC operations have been completed for an existing data sample, e.g., the current step of convolution on the input sample is completed, the output of accumulator flip flop (FF3) 524 will be sent to the next pipeline operation. The latter adds the current result with the bias followed by a non-linear activation. The accumulated errors (differences) from the auxiliary subtractor flip flop (FF4) 534 can be forwarded to the bias addition stage and the final summation is corrected before it is passed to the nonlinear function. This completes the compensation of multiple errors in a one-off manner.

Compared with existing fault mitigation techniques, the proposed method incurs no performance overhead as there is no need to stall or flush the pipeline. In addition, the errors can also be corrected without data replay and re-computation.

FIG.6 shows a visualisation of zeroing-on-error, bit masking and sign-bit masking techniques under biased fault injection patterns. FIG.6 provides a visualization of correct data 602 corresponding to the values 12 and 108 and vicious data 606 obtained by applying a fault based pattern 604 to the correct data 602. FIG.6 also shows the fault mitigations in zeroing-on-error 608 (i.e., TE-Drop or word masking), bit masking 610 and sign-bit masking 612 with biased faults. The faults incurred by the timing error are usually biased due to the inevitable carry propagation of the arithmetic circuits in DNN datapaths.

Word masking and TE-Drop aim to recover timing errors on SRAM column circuitry and MAC unit, respectively. We classify these two methods as zeroing-on-error since they reset the operation result to zero if any error is detected. As shown in FIG.6, a biased fault pattern leads to vicious data corruption of large error if the sign bit and MSB are flipped. Zeroing the whole data word on detection of error seems to be effective in substantially reducing the error magnitude if the data is corrupted by the flipping of a 0 bit to a 1 bit, as depicted in the left column of FIG.6. This is because zeroing a bit is equal to resetting that bit to 0 regardless of its value. However, if it is a bit flip from 1 to 0, zeroing the data is not as helpful as shown in the right column of FIG.6. It should be noted that zeroing operations can also be exploited by the attack. It is shown that it is possible to trigger the payload of a Hardware Trojan embedded into a DNN accelerator to selectively zero the intermediate data.

Under a random bit-flip model, bit masking is considered as a strong solution. It only sets the affected bit(s) rather than the whole word of data during the SRAM read operation, which limits the error propagation and complements the sparsity of the network model. However, the effectiveness of bit masking mitigation is limited to biased fault patterns. If both the sign bit and the MSB have the same polarity, bit masking has no effect as illustrated by the masking of the MSB with the sign bit in the left column of FIG.6. The protection of the sign bit is weak in the bit masking technique. Sign-bit masking is used to protect the memory cell from soft error and permanent fault. Obviously, this mitigation is also not effective against biased fault pattern as the sign bit and the MSB can be flipped simultaneously. The errors that can be resolved by sign-bit masking are illustrated in both cases of FIG.6. A hybrid mitigation scheme has also been proposed. On the detection of a flipping of the sign bit and the MSB,

data word zeroing instead of sign-bit masking is used. However, the fault mode, i.e., soft error and stack-at fault, is not valid for fault-injection attacks where there is no clear bound on the number of faults and constraints imposed on the flipped bit positions. The use of Razor flip flop for fault detection to counter hardware-oriented fault-injection attacks is also not effective against errors like radiation induced faults that may not incur any timing violation in the datapath.

FIG.7 is a block diagram showing an implementation of the multiply accumulator with forward error compensation according to an embodiment of the present invention in a deep learning accelerator.

The error-resiliency of the proposed forward error compensation design is demonstrated by implementing ResNet50 and VGG16 models on a Terasic Starter Platform (TSP) with Intel Cyclone V FPGA 710. The FPGA chip 710 consists of 301K logic elements, 342 variable precision DSP blocks 712, 1220 M10K RAMs 714 and 8 fractional PLLs. The TSP board 720 provides 1GB DDR3 on-board memory 722, GPIO headers 724 and PCI-E connector 726. In addition, it is certified by Intel OpenCL design flow with a board support package.

The accelerator structure is implemented in OpenCL kernel code with fixed-point arithmetic operators optimized by Verilog. Both the weights and the intermediate features are quantized into the 8-bit format. The MAC operation is implemented by Multiply Adder IP on new Intel FPGA devices. To improve the efficiency of the DSP block, the chain adder is enabled to combine the multiplication results from 8-bit multipliers. The proposed fault injection block shown in FIG.4 is integrated into the accelerator to generate the clock glitches into the DSP block 712. The configuration signals such as counter settings and glitch enable signal are controlled by an Analog Discovery 2 pattern generator. The host program 732 is run on a host PC, which loads images and model weights to the FPGA accelerator through the PCI-E interface, and manages the Analog Discovery 2 device through a USB connection 734.

For a fair comparison, the proposed forward error compensation circuit as well as the zeroing-on-error, bit masking and sign-bit masking are implemented on the same FPGA accelerator. They are shown as an additional pipeline stage with S1 to S4

parallel blocks in FIG.7. The detection module is designed with shadow registers inserted before the compilation rather than in the post-placement stage. In our experiments of fault injection attack, the normal output registers of multiply adder units are operated by the multiplexed clock between the normal kernel clock and the phase shifted clock. By contrast, the shadow registers are running with a normal kernel clock without glitches. A multiplexer is used to select the output from different mitigation schemes for fault recovery analysis. The control of the multiplexer is set by the pattern generator through GPIO.

The baseline project is compiled by using the Intel FPGA SDK for OpenCL v18.1, while the extra blocks including fault injection block and mitigation circuits are manually integrated into the RTL codes obtained from the OpenCL-based high-level synthesis tool. The modified RTL codes are compiled by Quartus Prime v18.1 to generate the final bitstream for configuration into the TSP board. To emulate the real existing DNN accelerator with significantly more efficient energy-quality scaling, the clock speculation is used for adjusting the output clock frequency of the kernel PLL. This can be accomplished by updating the parameters for the reconfigurable PLL IP in a post-flow compilation. Editing the parameter of the PLL settings will not affect the placement and routing implementations. By monitoring the final prediction results, the maximum operating frequency with zero slack on the target FPGA is 180MHz and 160MHz for ResNet50 and VGG16 respectively, whereas the conservative guaranteed frequency provided by the tool is only around 90MHz.

We take the first 256 images from the ImageNet validation dataset in our evaluation. Our target is to maintain the prediction result of correctly classified input images originally under a fault-free environment.

Following the setting of effective glitch injection attacks, four fault mitigation methods including the proposed forward error compensation, zeroing-on-error, bit masking and sign-bit masking are evaluated under this fault attack. We inject 0.01%, 0.05%, 0.1%, 0.5%, 1%, 5% and 10% glitches into the DSP block during the model execution on all the layers. A baseline design without protection is also evaluated as a reference. Both ResNet50 and VGG16 have been tested in this experiment.

FIG.8 shows the prediction accuracy of four error-tolerant designs and baseline design under the malicious fault-injection attack on all layers.

For the ResNet50 model, unprotected baseline accelerator (i.e., the grey dash line) cannot maintain full accuracy even under 0.01% fault injection rate. Significant accuracy loss (>99%) has been observed with injection rate greater than 0.5% on the baseline design. In contrast, our forward error compensation design illustrated in the figure with data points shown as stars can preserve 97.5% accuracy under 0.01% fault injection rate and still maintain 96.9% accuracy under 0.5% fault injection rate. Two other mitigation methods including zeroing-on error in the circular data points and bit masking in the diamond data points can also preserve the prediction accuracy below 0.5% glitches. However, with larger than 1% fault injection rates, both zeroing-on-error and bit masking experience increasing accuracy loss. For instance, the zeroing-on-error and bit masking methods suffer 20.2% and 55.6% loss on prediction accuracy, respectively, under 10% fault injection rate. In comparison, our forward error compensation method experiences at most 8% loss of accuracy for up to 10% of fault injection rate. Its prediction accuracy is 12.2% and 47.6% higher than the zeroing-on-error and bit masking methods, respectively. It is also shown in FIG.8 that the proposed method corrects the errors with a consistently higher recovery rate than all other mitigation methods regardless of the attack strength. The performance of zeroing-on error and bit masking methods are unstable under stronger attacks.

The sign bit masking technique shown in the square datapoints is apparently weak under fault injection attacks. It shares almost the same trend as the unprotected baseline design. The reason may be attributed to the biased faulty pattern of glitch injection attacks. This sign bit masking was implemented such that the sign bit is replaced with MSB upon detection of any error. The individually masked sign bit cannot recover bit-flip on the MSB, both the sign bit and the MSB and other bit positions. It has been suggested to combine sign bit masking with word masking and bit masking. The effect is conjectured to converge to either bit masking or word masking (i.e., zeroing-on-error).

As shown in FIG.8, the robustness of VGG16 is much higher than ResNet50. The baseline design starts to show accuracy degradation after just 1% of fault injection rate

and the maximum loss is 17.4% on 10% fault injection rate. Except for sign bit masking, all the other three mitigations can attain more than 95% accuracy under this evaluation on VGG16. Nevertheless, our proposed scheme continues to maintain higher accuracy than the zeroing-on-error and bit masking method by 5% and 3.7%, respectively. Besides, the source of accuracy loss on the proposed method is also analysed. Based on the proposed error compensation scheme, the critical path to the shadow flip flop may be worse than that to the main flip flop, as the latter is integrated inside the DSP block. Thus, the zero-slack frequency for that DSP block may still introduce error on the shadow flip flop in some rare and extreme conditions. Notwithstanding the potential room for further enhancement, the robustness of the current proposed design has already been successfully validated by this experimental prototype.

Other than injecting faults on all the layers of the DNN model, the attacker can selectively inject stronger faults on the sensitive layer(s). Hence, the effect of countermeasure against layer-wise attack should also be taken into consideration. To construct this adversary model, we first evaluate the layer-wise sensitivity of the baseline design. A powerful attack with a fault injection rate of 50%, i.e., one glitch added in every alternate clock cycle, is used to assess the accuracy loss on all the individual layers of ResNet50. The results are shown in FIG.9.

FIG.9 is a graph showing prediction accuracy of the unprotected baseline design under individual layer fault injection attacks.

Different layers have diverse susceptibility to fault attacks. Based on the layer-wise evaluation of the unprotected design, the top-5 most vulnerable layers of ResNet50 are found to be layers 2, 25, 40, 41 and 52. Attack on a single layer consumes only ≤ 1.7% of the inference time. Hence, the adversary can inject glitches into multiple layers to increase the attack success rate. For this evaluation, the top-5 most vulnerable layers and their adjacent layers constitute different combinations of three layers under fault injection attacks. The same fault injection rate is applied to these selected layers of each combination. The accuracy loss of the baseline and the four fault-resilient designs are reported. The results are summarized in Table. 1 below.

Table 1: Loss of prediction accuracy (%) under multi-layer attacks

Multi-layers	(1,2,3)	(24,25,26)	(39,40,41)	(40,41,42)	(51,52,53)
baseline	99.4	100	100	100	99.4
signBit	99.4	100	99.4	100	100
bitMasking	27	1.8	3.1	4.3	3.7
zeroing	44.8	3.7	3.7	2.5	1.8
forwardError	5.5	3.1	1.2	2.5	2.5

Apparently, the proposed method maintains the most robust classification accuracy of the DNN accelerator even with deliberate faults injected into three susceptible convolutional layers simultaneously. At most 5.5% loss, or equivalently exceeding 94% of prediction accuracy, is preserved by the proposed error-resilient design. In comparison, the prediction error of zeroing-on-error under the multi-layer attack on selective layers is nonnegligible. Up to 44.8% of prediction errors are observed on the layer combination (1,2,3). Bit masking also suffers 27% accuracy loss with the attack on the same three-layer combination. These results imply that discarding intermediate results or resetting them towards zero simply constrains the algorithmic redundancy and pushes the model towards the zero margin point. This situation is aggravated by the fact that wiping off the model redundancy such as model compression and binarization are becoming popular for efficient processing and hardware implementation. Thus, augmented built-in error correction or compensation circuit should be considered in modern compact and efficient designs.

The hardware implementation cost of the proposed design can be obtained by comparing the original convolution module and our error-resilient convolution module since other modules like PCI-E interface, memory controller and simple SIMD modules are not altered by our proposed method. The overhead is due to the MAC unit. The extra small adder and subtractor required for error compensation are mapped to logic elements. To analyse the logic area overheads, the MAC units of the baseline and proposed designs are realized by logic-element mapping consistently. This implementation of an original MAC unit consumes 373 adaptive look-up tables (ALUTs). In comparison, only 19 and 33 additional ALUTs are required for the

subtractor and adder, respectively of our forward error compensation. Since the total number of MAC units remain unchanged, the area overhead incurred by these added components of a MAC unit is 13.9%. Table 2 below summarizes the overheads on FPGA resources and power consumption between the baseline and proposed convolution modules on ResNet50 implementation.

Table 2: Overheads on resource and power consumption of the proposed convolution module on ResNet50 implementation

Resource/Power	Baseline	Forward-error	Percentage
ALMs	3477.6	3748.6	7.80%
ALUTs	5272	5802	10.10%
Registers	6253	6445	3.10%
Dynamic Power	63.03 mW	64.58mW	2.50%
Total Power	90.64 mW	94.01 mW	3.70%

Given that the original complete convolution module consumes 3477.6 adaptive logic modules (ALMs) with 5272 ALUTs and 6253 dedicated logic registers, the actual overall additional cost is 7.8% in ALMs. Specifically, the combinational and sequential area overheads introduced by our proposed method are 10.1% (530ALUTs) and 3.1% (192 registers), respectively. The worst-case delay is not affected as the added subtractor and adder are placed between FF2 to FF4 and FF4 to FF3, as shown in FIG.5. These modified datapaths are shorter than the critical path of the original design. Thus, the maximum operating frequency is preserved. The power dissipations shown in the last two rows of Table 2 are obtained by the Quartus tool based on default toggle rate for IO signals. The power overheads of the forward error compensation approach are 2.5% and 3.7% on dynamic and total power, respectively.

The proposed forward error compensation approach can be implemented into the DNN accelerators implemented in ASIC and FPGA platforms to increase its robustness against computation errors induced by natural environmental variations and malicious fault attacks. Reliable deep learning accelerator is a key enabler of a variety of edge applications ranging from safety-critical mission like defect detection,

human-machine interaction, video surveillance and self-driving car to consumer applications including automatic checkout, face ID and virtual personal assistants.

Whilst the foregoing description has described exemplary embodiments, it will be understood by those skilled in the art that many variations of the embodiments can be made within the scope and spirit of the present invention.

CLAIMS

1. A multiply and accumulation circuit comprising:

a multiplication circuit configured to generate a multiplication output from at least two input signals;

a main flip flop synchronized to a main clock signal and configured to receive the multiplication output as an input;

an auxiliary flip flop synchronized to a relaxed clock signal which is delayed with respect to the main clock signal and configured to receive the multiplication output as an input;

an accumulation adder configured to add an output of the multiply and accumulation circuit from a preceding clock cycle to the output of the main flip flop;

a subtractor configured to determine a difference between an output of the main flip flop and an output of the auxiliary flip flop;

an auxiliary adder configured to add the difference to a signal based on an output of the accumulation adder; and

a multiplexer configured to generate an output of the multiply and accumulation circuit and configured to modify the output of the multiply and accumulation circuit to compensate for the difference between the output of the main flip flop and the output of the auxiliary flip flop.

2. The multiply and accumulation circuit of claim 1, wherein the multiplexer is configured to compensate for the difference between the output of the main flip flop and the output of the auxiliary flip flop in a subsequent clock cycle.

3. The multiply and accumulation circuit of claim 1 or claim 2, further comprising an accumulator flip flop which receives the output of the accumulation adder and wherein the auxiliary adder is configured to add the difference to an output of the accumulator flip flop.

4. The multiply and accumulation circuit of any one of the preceding claims, wherein the relaxed clock signal is delayed with respect to the main clock signal by a fraction of a period of the main clock signal.

5. The multiply and accumulation circuit of any one of the preceding claims, wherein the multiplication circuit comprises a first flip flop and a second flip flop configured to receive as inputs respective signals of the at least two input signals, wherein the first flip flop and the second flip flop are synchronised with the main clock signal.

6. The multiply and accumulation circuit of any one of the preceding claims, wherein the multiplication circuit comprises a plurality of multipliers and a chain adder configured to add the outputs of the multipliers of the plurality of multipliers.

7. A hardware accelerator for a machine learning system comprising the multiply and accumulation circuit according to any one of claims 1 to 6.

8. A forward error correction method in a multiply and accumulation circuit, the method comprising:

determining a difference between an output of a main flip flop and an output of an auxiliary flip flop, wherein the main flip flop is synchronized to a main clock signal and coupled to a multiplication output and the auxiliary flip flop is synchronized to a relaxed clock signal which is delayed with respect to the main clock signal and coupled to the multiplication output; and

adding a signal dependent on the difference to an output of the multiply and accumulation circuit to compensate for the difference between the output of the main flip flop and the output of the auxiliary flip flop.

9. The forward error correction method of claim 8, wherein the signal dependent on the difference is added to the output of the multiply and accumulation circuit in a subsequent clock cycle.

10. The forward error correction method of claim 8 or claim 9, wherein the relaxed clock signal is delayed with respect to the main clock signal by a fraction of a period of the main clock signal.