ONNC Compiler Used in Fault-Mitigating Mechanisms Analysis on NVDLA-Based and ReRAM-Based Edge AI Chip Design

Samuel Liu[1,2], Jen-Ho Kuo[1], Luba Tang[1], Ning-Chi Huang[2], Der-Yu Tsai[1], M.-H. Yang[2], Kai-Chiang Wu[2],
Skymizer Taiwan Inc.[1] National Yang Ming Chiao Tung University, Hsinchu, Taiwan[2]
{samuel, hankkuo, luba, a127a127}@skymizer.com;{kcw, nchuang}@cs.nctu.edu.tw; msyang.cs08g@nycu.edu.tw

Abstract—The calculation of the convolution is considered the most important part in convolutional neural network-based artificial intelligence (AI) deep learning accelerator (DLA) architecture.

Regardless of employing the traditional von Neumann architecture or the non-von Neumann compute-in-memory architecture, the variations arising from the integrated circuit (IC) process at the mass production stage must be considered. The AI model accuracy also needs to be improved.

The open neural network compiler (ONNC) is an architecture-aware AI software development kit toolchain, which is capable of evaluating the AI model on fixed-point Edge AI DLAs for the open neural network exchange model.

In this paper, the ONNC toolchain and the NVIDIA DLA (NVDLA) field-programmable gate array are employed to investigate the NVDLA static random-access memory stuck-at fault impact on model accuracy. The fault-mitigating results after applying a recovery mechanism in the IC process hard failure are also discussed.

A method for utilizing the ONNC framework and the recovery guidelines for the resistive random-access memory -based computer-integrated manufacturing DLA architecture is also proposed.

Index Terms—Deep Learning Accelerators, Compilers, NVDLA, ONNC, Fault-Mitigating Mechanisms, SRAM Stuck-at Fault, ReRAM, CIM Simulator

I. Introduction

The NVIDIA deep learning accelerator (NVDLA) provides free intellectual property licensing to anyone desiring to build a chip that utilizes deep neural networks (DNNs) for inference applications. The open neural network compiler (ONNC) provides an extensible compiler, which is a quantization calibrator supporting running DNN models on NVDLA-based systems-on-chips (SoCs). In an artificial intelligence (AI) classification model employing MAC processing convolution as the most important computing core, the quantitative data obtained indicate that the location of different faulty bits is clearly related to the inference accuracy. Also, under the same fault rate, a stuck-at fault occurring in the high-order bits can seriously affect the final result.

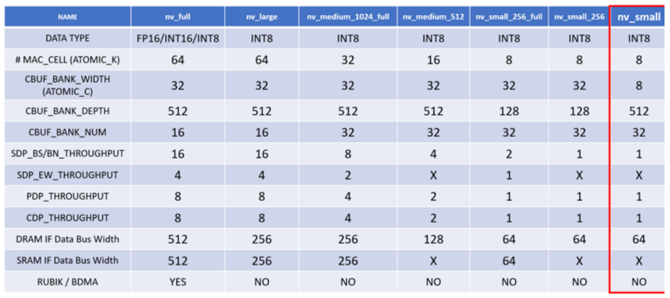
In this paper, some lightweight fault-mitigating mechanisms to enhance effectively the robustness of NVDLA-based Edge AI chips when encountering an internal static random-access memory (SRAM) built-in self-test fault is presented. This is useful when applying a fault-mitigating mechanism for error-tolerant applications such as image classification using an AI convolutional neural network (CNN) model. The proposed non-accurate MAC calculation analysis, which is applied to the entire convolution computation, leads to very promising results compared with the case of employing an exactly accurate convolution operation. The fault-mitigating mechanism analysis and design presented in this paper can also easily be applied to similar fixed-point AI SoCs designs and opens new opportunities for research and product development.

The DNN model is very capable of solving many challenging problems encountered in applications such as self-driving cars, robotics, medical image classification, and high-performance games. In many cases, it is capable of extracting advanced features from raw sensory data instead of using expertly designed manual features or rules, which exceed the human accuracy. DNN research and development has made remarkable progress, leading to a demand for widespread deployment of commercial products and stimulating significant research work in the field of hardware acceleration for advanced AI applications. The computationally intensive and memory-intensive DNN nature has been driving the research and development direction of DNN accelerators [1]. The NVDLA is a free and open-source neural processor architecture, originally designed for NVIDIA's Xavier SoC inference accelerators [2]. With its modular architecture, the NVDLA is scalable and highly configurable and aims to simplify integration and portability. Many chip manufacturers have been interested in integrating the NVDLA into their SoCs for commercial AI applications due to the following three reasons: 1) free of license fees and royalties, 2) flexibility and scalability, and 3) open-source application-specific integrated circuit (IC) solutions for edge computing. Despite attracting significant attention, the NVDLA still faces several obstacles in product deployment, including the lack of an open-source NVDLA compiler, incomplete verification of hardware design, and incomplete software stack support. The ONNC [3], which connects the open neural network exchange (ONNX) [4] model to the NVDLA, provides a compilation framework for NVDLA-based designs. The NVDLA and ONNC together provide low-cost and fast-track opportunities for DNN inference. Simultaneously, they have strict energy consumption, computing, and memory cost constraints in edge computing.

The most important hardware engine in the NVDLA is the CONV processor. On average, more than 60% of the inference time is consumed in the CONV processor. The performance of the CONV processor is proportional to the number of MAC units employed in this processor. Maximum performance can be achieved by maximizing the MAC utilization rate. In the NVDLA design, there is an internal convolution buffer in the CONV processor that stores feature data and weight. The NVDLA first fetches the feature data and weight into the convolution buffer. Then the data and weights are transferred to the MAC units, which perform the calculations. Using this buffer, long latencies during tensor fetching from DRAM are avoided, and the MAC is kept busy all the time. Therefore, the convolution buffer size also affects the inference.

NVDIA parameterized the NVDLA after Xavier was fabricated. The NVDLA became the low-cost version module for IoT devices and some other configurations. In Fig. 1, the parameters of different

NVDLA configurations in the Github hardware repository are compared. The hardware parameters shown in rows 2–5 in Fig. 1 are the most critical in terms of performance. They involve the configuration of MAC and convolutional buffer (CBUF). The ATOMIC_K represents the number of MAC cells in the convolution engine, whereas the ATOMIC_C represents the number of multipliers within each MAC cell unit. The product “ATOMIC_K × ATOMIC_C” represents the computational capability, and it is sometimes reflected in the configuration names.



NAME	nv_full	nv_large	nv_medium_1024_full	nv_medium_512	nv_small_256_full	nv_small_256	nv_small
DATA TYPE	FP16/INT16/INT8	INT8	INT8	INT8	INT8	INT8	INT8
# MAC_CELL (ATOMIC_K)	64	64	32	16	8	8	8
CBUF_BANK_WIDTH (ATOMIC_C)	32	32	32	32	32	32	8
CBUF_BANK_DEPTH	512	512	512	512	128	128	512
CBUF_BANK_NUM	16	16	32	32	32	32	32
SDP_BS/BN_THROUGHPUT	16	16	8	4	2	1	1
SDP_EW_THROUGHPUT	4	4	2	X	1	X	X
PDP_THROUGHPUT	8	8	4	2	1	1	1
CDP_THROUGHPUT	8	8	4	2	1	1	1
DRAM IF Data Bus Width	512	256	256	128	64	64	64
SRAM IF Data Bus Width	512	256	256	X	64	X	X
RUBIK / BDMA	YES	NO	NO	NO	NO	NO	NO

Figure 1: NVDLA configuration

The ATOMIC_C also determines the bank width of the CBUF in the convolution engine. The CBUF size is determined using the BANK_DEPTH and BANK_NUM.

Weight and data share the CBUF during convolution, and the user can program the number of banks for data and weight during compilation time. These numbers are critical to the performance, and the user is responsible for selecting the most suitable bank numbers.

In this paper, the NV_SMALL configuration is adopted, and the total SRAM CBUF size for convolution is 128 Kbytes. This is composed of 32 64 × 512 bit SRAM blocks.

Examples of integrated circuit yield models used by IC manufacturers include the Murphy, Poisson, Seeds, and Exponential models. All models imply that yield is related to the defect density and chip area.

From the floor plan of ITRI’s NVDLA 64MAC test chip [5] shown in Fig. 2, it can be extracted that the SRAM area occupies roughly 50% of the die size. This area is much larger than the MAC area. Therefore, after the CP/FT test procedure, the impact on the overall chip yield due to process defects on the SRAM part will also be revealed.

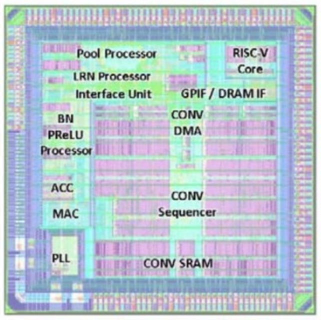


Figure 2: ITRI’s NVDLA test chip layout

The SRAM-yield loss can be avoided by adding redundant or spare rows and columns of storage cells so that the faulty cells can be redirected to redundant cells. Memory repair includes row repair, column repair, or a combination of both.

Built-in self-repair (BISR) and built-in redundancy analysis modules help calculate the repair signature based on the memory-failure data and the implemented memory redundancy scheme. They also determine whether the memory is repairable in production testing environments. However, a redundant memory cell is required.

Considering the cost of the error correction code (ECC) scheme in the implementation of the NVDLA NV-SMALL configuration, both the weight and activation data are configured as 64-bit SRAM macros with a 512 length. For a 64-bit word, extra 8-bit check bits are required, which result in a 12.5% storage overhead. Also, the decoding and encoding of a multi-bit ECC for protecting the SRAM-based cache design in Edge AI cost-sensitive devices are expensive to implement.

Regarding the convolutional (CNN)-AI-model usage, the obtained image data may exhibit deviations due to the image sensor lens or data differences caused by the internal analog-to-digital conversion in the IC. However, these deviations do not affect the usability of a specific application. This is because the best answer is not always found for each inference result based on imperfect input data. When dealing with the classified AI model, we can just try to maintain the order of the final inference results, even after error compensation. In this way, users can still obtain appropriate recommendations in the end. Therefore, the lightweight fault-mitigating mechanisms proposed in this paper are designed by taking advantage of this feature and the input data characteristics.

The work presented in this paper is based on our previous work [9]. Here more experiments and analysis are conducted. An ONNC-based software development platform for configurable NVDLA designs is presented in Fig. 3. This platform facilitates the software/hardware co-design process and early-stage software development for NVDLA-based designs. The quantization results after calibrating from the FP32 pre-trained ONNX model to the INT8 deployed model are discussed in Section II. The accuracy reduction when stuck-at faults are injected is described in Section III, and the proposed recovery mechanisms are described in Section IV. Subsequently, we use our demo program to check the fault-mitigating behavior, which is described in Section V. The fault-mitigating selection policy is proposed in Section VI, and the conclusions are presented in Section VII.

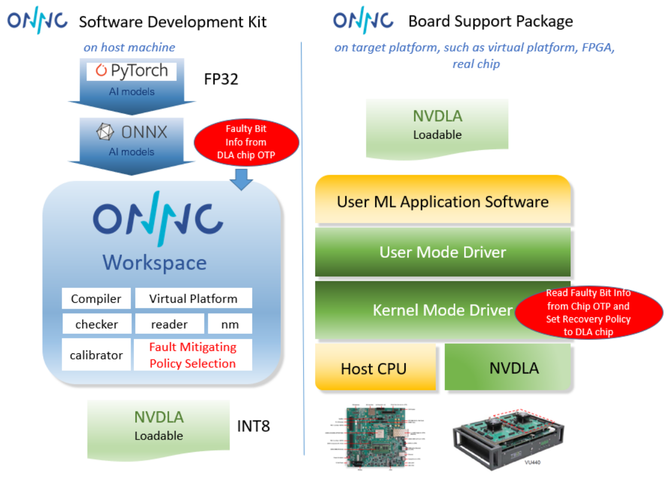

ONNC Software Development Kit
on host machine
PyTorch
AI models
FP32
ONNX
AI models
Faulty Bit Info from DLA chip OTP
ONNC
Workspace
Compiler
Virtual Platform
checker
reader
nm
calibrator
Fault Mitigating Policy Selection
NVDLA Loadable
INT8
ONNC Board Support Package
on target platform, such as virtual platform, FPGA, real chip
NVDLA Loadable
User ML Application Software
User Mode Driver
Kernel Mode Driver
Read Faulty Bit Info from Chip OTP and Set Recovery Policy to DLA chip
Host CPU
NVDLA

Figure 3: ONNC Compiler process for compiling the PyTorch FP32 model to generate the NVDLA loadable process

II. Quantization Results

Initially, several state-of-the-art CNN pre-trained models were selected from the PyTorch torchvision package and converted to

ONNX format models. The ONNC was used to compile these models and generate the NVDLA loadables. Next, a calibration procedure was conducted to generate the INT8 NVDLA models. Finally, the ImageNet ILSVRC2012 test set was conducted to check the accuracy of the Xilinx ZCU102 field-programmable gate array (FPGA). From the results shown in Fig. 4, it can be observed that the quantization slightly reduces the TOP5 accuracy within 2%. In this paper, this quantization accuracy result was used as a base to check the accuracy variation caused by the weight or activation input encountering bit stuck-at faults. Also, the accuracy result was set as the target for evaluating the fault-recovery mechanisms.

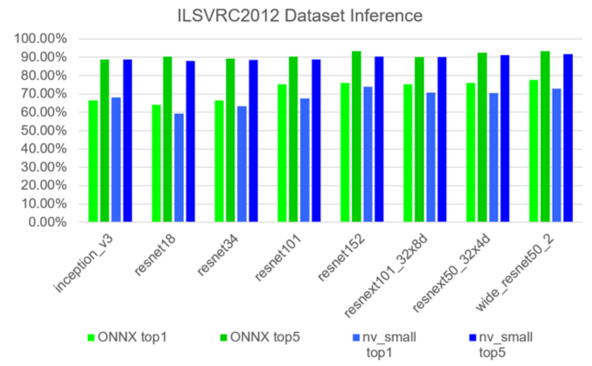


Figure 4: Accuracy comparison between the FP32 and INT8 inference

III. Fault Model and Fault Injection

According to [6], the general SRAM block structure includes both a row driver and a column driver. If an error occurs in the column-based Power Source (Vdd, Vss), the output of the same column may also have a stuck-at fault. Because this situation is more serious than the single-cell access problem, it is adopted as the fault model for the analysis. By considering that the proposed recovery mechanism is capable of dealing with this serious type of error, a single-bit cell will be easier to handle.

In this paper, the NV_SMALL 64 MAC configuration version was adopted as the CNN model inference platform. Initially, it must be checked whether the fault is due to the CBUF SRAM, which affects the convolution computation. Therefore, each bit is treated separately to check the accuracy variation when faults occur from the low-order bit to the high-order bit.

This experiment was conducted by injecting stuck-at faults in the NVDLA open-source CMAC Verilog code. Then, the register transfer language implementation of the MAC array was synthesized (Fig. 5), and its gate-level netlist was generated. Finally, the bitstream was written out and was employed as a Xilinx ZCU102 FPGA boot image to boot-up the NVDLA Linux platform.

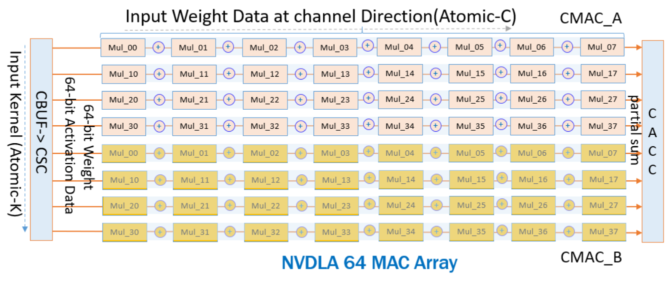


Figure 5: NVDLA 64 MAC architecture

For physical design optimization, the NVDLA CMAC is divided into two parts, CMAC_A and CMAC_B, as shown in Fig. 5. The same bit error is injected to both CMAC units. Thus, two MACs out of a total 64 MACs will obtain error data with a 3.1% fault rate.

When the SRAM weight data encounter a data stuck-at fault, it can be concluded that if there is an error in the position above bit2, the error will significantly affect the inference results. This is confirmed by the results shown in Fig. 6.

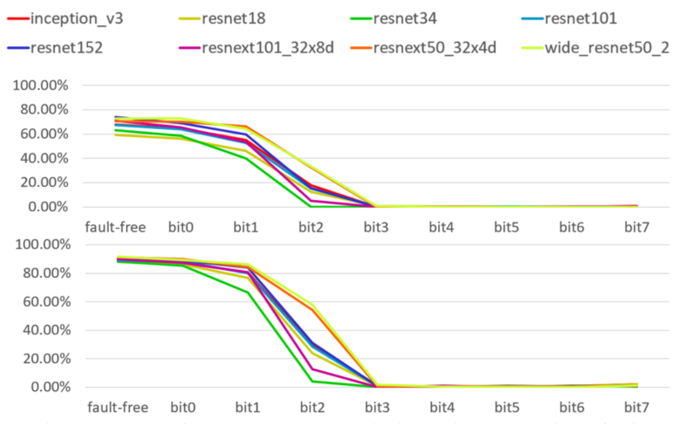


Figure 6: Top1/Top5 accuracy variation when a stuck-at fault occurs in the weight input

When the SRAM activation data encounter a data stuck-at fault, it can be concluded that if there is an error in the position above bit4, the error will significantly affect the inference results. This is confirmed by the results shown in Fig. 7.

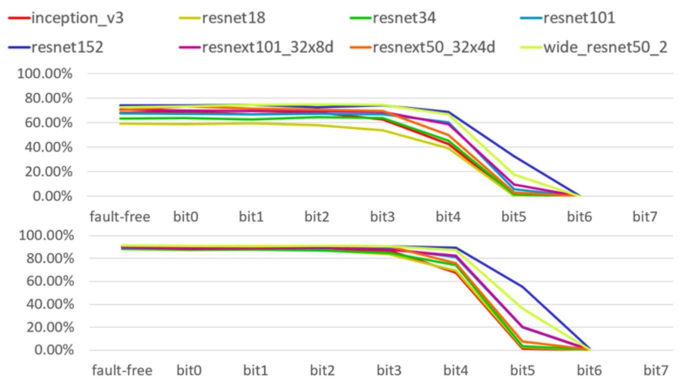
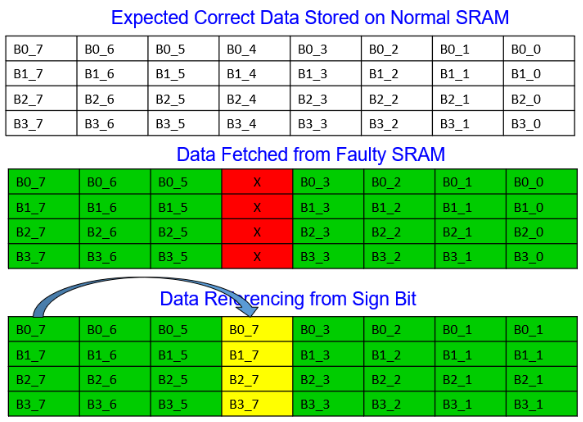


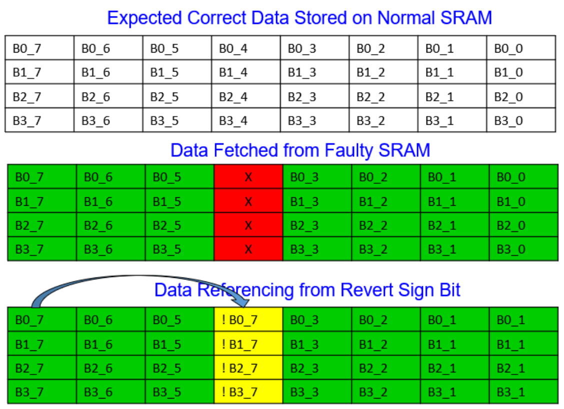
Figure 7: Accuracy variation when a stuck-at fault occurs in the activation data input

IV. Proposed Fault-Mitigating Mechanisms

Because the trained AI model inference does not necessarily rely on the exactly accurate convolution computation, some error might be acceptable to lead to acceptable final accuracy results [7]. Thus, the sign-bit policy is analyzed first (as shown in Fig. 8) [8], and more analysis is performed on the inverse sign-bit policy (as shown in Fig. 9). Then, three lightweight fault-recovery mechanisms are proposed, and the results are checked.

For a fixed error falling into a certain bit position, we know that the original numbered value should be within a certain range, where the sign bit is used to replace the error bit. This sign bit represents the lower limit of the absolute value of the original value. The error bit is replaced with the inverse sign bit, which represents the upper limit of the absolute value of the original value. Therefore, these two compensation mechanisms are first introduced to understand whether the eight test models are affected by the change in the error bit value.


Expected Correct Data Stored on Normal SRAM
B0_7 B0_6 B0_5 B0_4 B0_3 B0_2 B0_1 B0_0
B1_7 B1_6 B1_5 B1_4 B1_3 B1_2 B1_1 B1_0
B2_7 B2_6 B2_5 B2_4 B2_3 B2_2 B2_1 B2_0
B3_7 B3_6 B3_5 B3_4 B3_3 B3_2 B3_1 B3_0
Data Fetched from Faulty SRAM
B0_7 B0_6 B0_5 X B0_3 B0_2 B0_1 B0_0
B1_7 B1_6 B1_5 X B1_3 B1_2 B1_1 B1_0
B2_7 B2_6 B2_5 X B2_3 B2_2 B2_1 B2_0
B3_7 B3_6 B3_5 X B3_3 B3_2 B3_1 B3_0
Data Referencing from Sign Bit
B0_7 B0_6 B0_5 B0_7 B0_3 B0_2 B0_1 B0_1
B1_7 B1_6 B1_5 B1_7 B1_3 B1_2 B1_1 B1_1
B2_7 B2_6 B2_5 B2_7 B2_3 B2_2 B2_1 B2_1
B3_7 B3_6 B3_5 B3_7 B3_3 B3_2 B3_1 B3_1
Figure 8: A sign bit is used to replace the error bit


Expected Correct Data Stored on Normal SRAM
B0_7 B0_6 B0_5 B0_4 B0_3 B0_2 B0_1 B0_0
B1_7 B1_6 B1_5 B1_4 B1_3 B1_2 B1_1 B1_0
B2_7 B2_6 B2_5 B2_4 B2_3 B2_2 B2_1 B2_0
B3_7 B3_6 B3_5 B3_4 B3_3 B3_2 B3_1 B3_0
Data Fetched from Faulty SRAM
B0_7 B0_6 B0_5 X B0_3 B0_2 B0_1 B0_0
B1_7 B1_6 B1_5 X B1_3 B1_2 B1_1 B1_0
B2_7 B2_6 B2_5 X B2_3 B2_2 B2_1 B2_0
B3_7 B3_6 B3_5 X B3_3 B3_2 B3_1 B3_0
Data Referencing from Revert Sign Bit
B0_7 B0_6 B0_5 ! B0_7 B0_3 B0_2 B0_1 B0_1
B1_7 B1_6 B1_5 ! B1_7 B1_3 B1_2 B1_1 B1_1
B2_7 B2_6 B2_5 ! B2_7 B2_3 B2_2 B2_1 B2_1
B3_7 B3_6 B3_5 ! B3_7 B3_3 B3_2 B3_1 B3_1
Figure 9: A reverse sign bit is used to replace the error bit

In the case of the weight data error, a sign bit is used to replace the error bit. Then, the top1/top5 accuracy variation shown in Fig. 10 is obtained.

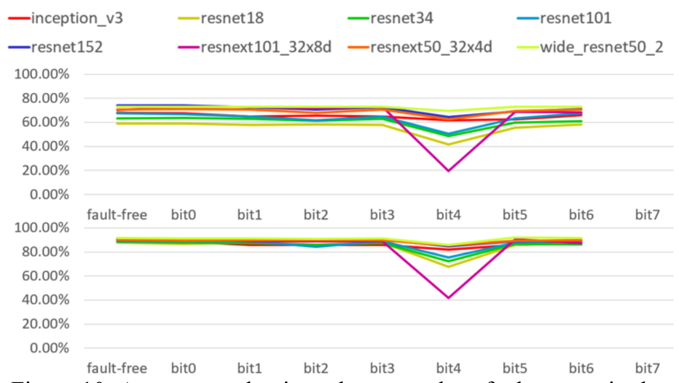


Figure 10: Accuracy reduction when a stuck-at fault occurs in the weight input and recovery using a sign bit

An inverse sign bit is used to replace the error bit. Then, the top1/top5 accuracy variation shown in Fig. 11 is obtained.

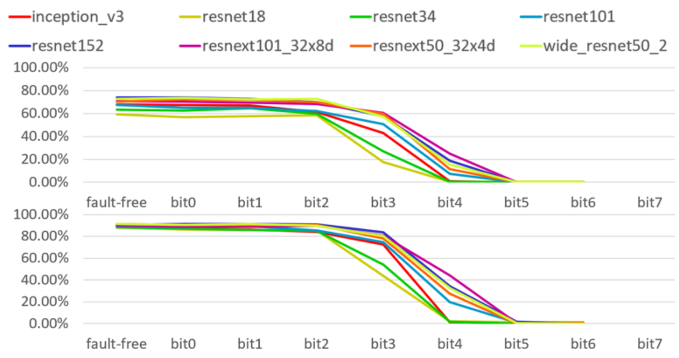


Figure 11: Accuracy reduction when a stuck-at fault occurs in the weight input and recovery using an inverse sign bit

In the case of an activation data stuck-at fault, a sign bit is used to replace the faulty bit. Then, the top1/top5 accuracy shown in Fig. 12 is obtained.

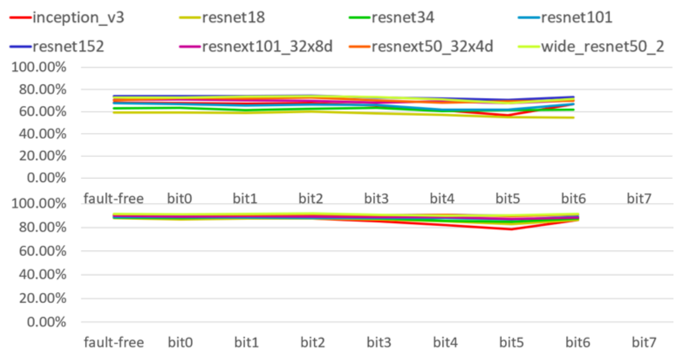


Figure 12: Accuracy reduction when a stuck-at fault occurs in the activation input and recovery using a sign bit

An inverse sign bit is used to replace the error bit. Then, the top1/top5 accuracy shown in Fig. 13 is obtained.

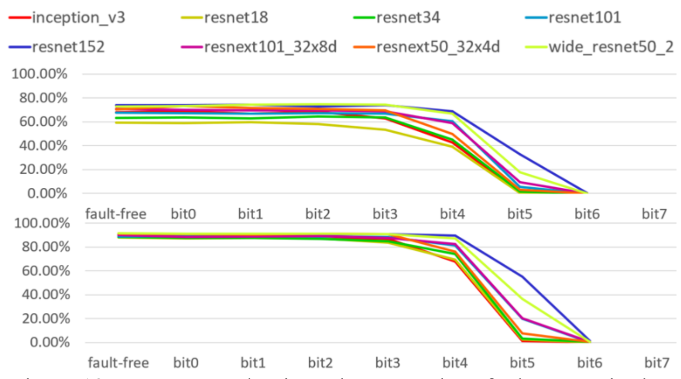


Figure 13: Accuracy reduction when a stuck-at fault occurs in the activation input and recovery using an inverse sign bit

The aforementioned experiments show that for high-order bit errors, better results can be achieved by using a sign bit to compensate for the error than using an inverse sign bit.

It should be emphasized that there is no bit7 value in Figs. 10–13. The reason is that both the weight and activation data are used for the convolution calculation as signed 8-bit values, and bit7 itself represents the sign bit. However, this method cannot compensate for the error in the sign bit.

Actually, the degree of error reduction is closely related to the result obtained from the multiplication and addition operation of the convolution. Weight is an important calculation data source, which is used and obtained after the model is trained and quantized. The activation data are the result of pooling/activating after the previous convolution is multiplied by the weight. Therefore, the weight data characteristics are very important.

The 8-bit weight value used in this paper is actually obtained using the quantization process through the ONNC calibrator. The 32-bit floating-point expression originally employed in the ONNX model was analyzed and calculated by adopting the Kullback–Leibler divergence method, and the scaling factor of each layer was determined.

The weight data characteristics of the 8 AI models used in the experiment were analyzed, and their distribution characteristics were observed. The actual data range of the weight value used for the convolution calculation is between −127 and 128. This weight value was used as a signed integer. By observing the distribution of the eight AI models used as unsigned integers in Fig. 14, the weight of the first convolution shows that the middle distribution appears very small, and most values are concentrated on both sides. This means that the absolute value of the weight value is mostly concentrated on the smaller value, whereas the larger value occurs less frequently. By observing the 8-bit integer expression of an unsigned number, most values utilize bit0 to bit3 frequently, whereas bit4, bit5, and bit6 are not used frequently.

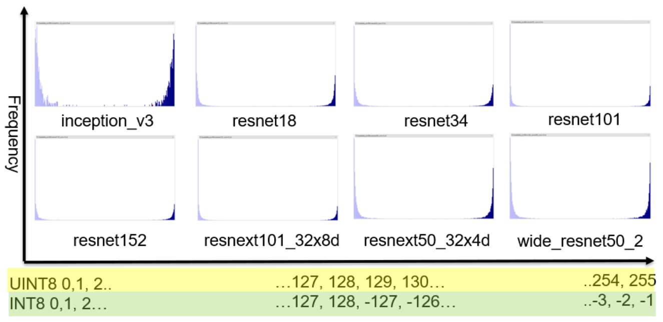


Figure 14: Weight value distribution of the first convolution layer of the eight test AI models

Actually, the weight of each convolution layer in the model exhibits a similar distribution. The resnet18 model was used as an example to show the weight distribution of its 21 convolution layers (Fig. 15).

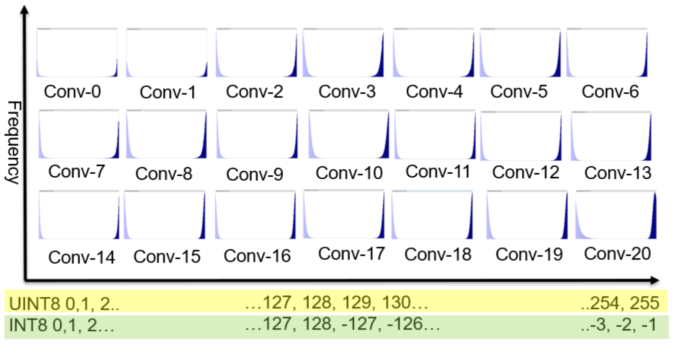


Figure 15: Weight value distribution of the 21 convolution layers in the Resnet-18 model

From the experimental results, it can be observed that the high-order bit errors have a significant impact on the final accuracy. Also, employing an inverse sign-bit policy can be worse than employing a sign-bit policy. This is because the stuck-at faults significantly affect the numerical results of the original calculation.

Actually, the CNN AI model generally employs the pooling and activation operators after the convolution layer. As a result, large absolute values will induce a large deviation from the originally expected convolution results, which can easily be transferred from the pooling and activation layer to the next convolution layer. This is also an important factor considering the error recovery mechanism.

V. Fault-Mitigating Behavior Check

The classification results obtained from the three methods, Fault Free (Control Group), Weight Bit3 SA0, and Fault Recovery, are presented in Fig. 16. These results were obtained from our implemented program, which was designed using inception_v3 for inference. The two images (quail and stage) shown in Fig. 16 were selected from the ImageNet Competition [10]. The top five classification results are shown in the top row (Fault-Free method), which is our control group. The top five classification results using the Weight Bit3 SA0 method are shown in the middle row. Serious errors can be observed in comparison with the Fault-Free method results. The Fault-Recovery method results are shown in the bottom row. It can be observed that they are the same as those of our control group.

The classification results in Fig. 16 show that the proposed Fault-Recovery compensation mechanism produces the same results as those of the control group. By comparing the top row (Fault-Free method) classification results with the bottom-row (Fault-Recovery method) results, the two pictures are the same. However, for the bottom-row pictures, the calculated confidence level percentage values of the softmax layer have been dropped from 98.18% to 87.83% and from 86.00% to 53.06%. This result does not affect the inference result, because the classifier is essentially based on the highest-value category.

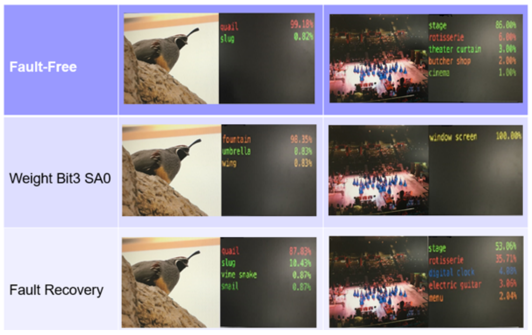


Figure 16: Inception_v3 was used to produce inference, and the actual top5 confidence level changes were checked

VI. CIM Hardware Simulator

The motivation behind this project was to create a basic framework for modeling computer-integrated manufacturing (CIM) designs. The ONNC framework provides an easy way to connect AI models with hardware execution. It has a generic runtime library for ONNX operator support. The crossbar device models have been investigated in many research projects. In these projects, a full-model simulation was adopted to collect statistics from pre-trained models.

This goal was achieved by integrating the ONNC with a CIM hardware simulator containing a CIM device error model [12]. Our simulator [11] is compatible with the ONNX version 1.3.0. It employs two execution modes: the stand-alone and ONNC modes. In the stand-alone mode, the simulator is built as a library. In the ONNC mode, the simulator obtains an ONNX model, runs the simulation, and reports the results. The simulator is also extendible.

Users can add new operators or modify the hardware simulator according to their specific application.

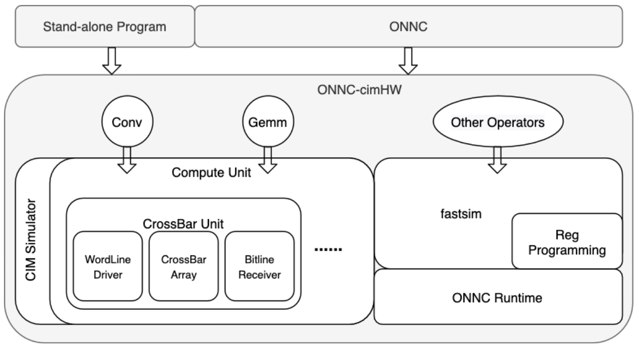


Figure 17: CIM hardware simulator

The Compute Unit may consist of several Crossbar units. Each crossbar unit contains the following three components: a Wordline driver, a Crossbar Array, and a Bitline Receiver.

The Wordline driver accepts an input vector X and quantizes a value into a given length of bits. In each cycle, it transfers a number of bits to the Crossbar Array, executes the MAC operation, and sends the result to the Bitline Receiver. The Bitline Receiver adds and shifts the result cycle by cycle to obtain the final result.

In the proposed implementation, users may specify different configurations in the crossbar unit for experimental purposes.

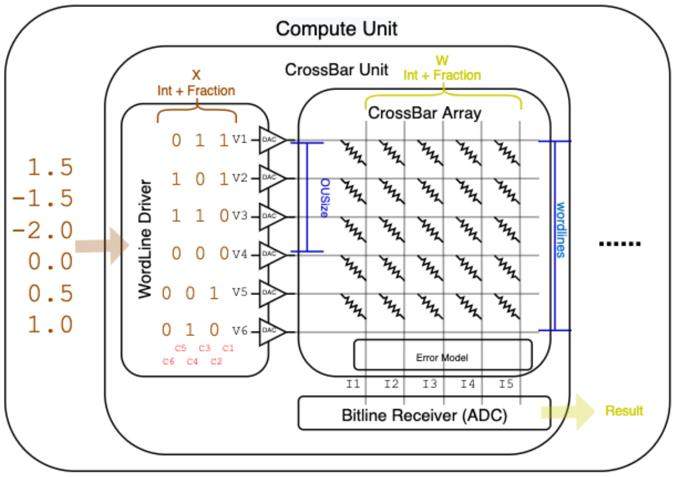


Figure 18: CIM hardware simulator

A CIM configuration employing integer with 8 bit, fraction with 8bit, and OU1/OU2 was selected for the experiment. Model zoo/Squeezenet was quantized to integer 8bit and fraction 8 bit numeric formats. Eight pictures were selected to produce a top-1 inference. The results between FP32/Quantized int_8+frac_8 and the OU1 error case were compared, as shown in Fig. 19. We knew that the random bit occurs on the MSB bit will seriously impact the final inference result.

Image No	196	253	1161	1678	1902	1968	2284
ground truth - top1	396	438	327	92	471	217	6
ONNI confidence	0.99991	0.796846	0.99997	0.99887	0.8597	0.9053	0.58514
ONNC confidence	0.99757	0.631197	0.9542	0.7852	0.39679	0.35847	0.67303

Sueezenet								
ILSVRC2012 Image No.	196	1161	1678	1968	3068	5420	5920	6603
ground truth - top1 class No.	396	327	92	217	89	739	742	16
ONNI confidence	0.992264	0.99889	0.99789	0.845496	0.999964	0.92384	0.72476	0.943304
ONNC confidence	0.99223	0.99889	0.997882	0.845373	0.999963	0.923518	0.72333	0.943207
ONNI inference result - top1 class No.	396	327	92	217	89	739	742	16
ONNC inference result - top1 class No.	396	327	92	217	89	739	742	16
Error Injection : 1st test result	315	660	351	771	977	986	403	901
Error Injection : 2nd test result	368	992	627	998	351	946	500	162
one error_OU9, F16, ONNC confidence	0.992247	0.998883	0.99787	0.844993	0.999963	0.921578	0.72339	0.943469
Error Injection : 1st test result	396	327	92	217	89	739	742	16

Fig. 19: CIM hardware simulator inference results

The impact on the results can be checked by considering the CIM error model with the OU1 setting. And we propose the hybrid-mode accelerator architecture, which only inject the error in lower 16bit, and we find the accuracy can be well-preserved.

VII. Conclusions and Future Work

The investigation presented in this paper indicates that under the same fault rate, the occurrence location of an SRAM stuck-at fault is related to the final AI model inference result. Also, a stuck-at fault occurring at a high-order bit will seriously affect the final result.

In this paper, in the case of the SRAM SA0/SA1 fault, the traditional ECC data recovery mechanism was not used to restore the data to their original value. Instead, the characteristics of similar data distribution are used, and we can check the two lightweight error mitigation mechanisms which utilize the sign bit information.

We are still conducting more experiments to confirm that the AI models commonly used in the industry for object detection and object segmentation can also apply the error mitigation and repair mechanisms mentioned in this paper.

Also, we are currently working on an interface modification of the OpenSRAM, and we propose a mechanism suitable for general SRAM BISR. The prospect of cooperating with companies involved in the AI computing field of the EDA tools, SRAM IP, DRAM IP, and the in-memory computing area is also discussed.

For the CIM architecture, better simulation accuracy can be achieved by adding a quantization adjustment circuit between the CIM module and other calculation units. Also, our scaling factor can be applied to the quantization setting to closely approach the floating-point accuracy.

Besides, a correlation computing operator can be applied after the convolution and GEMM operation, which are executed by the CIM simulator. Also, the proposed calibrator aims to consider the variation impact factors and apply compensation to approach the ideal calculation output.

References

[1] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329.

[2] NVDLA deep learning accelerator. [Online]. Available: http://nvdla.org/, 2017.

[3] ONNC open-source project. Release v1.3. [Online]. Available: https://github.com/ONNC/onnc, April 2020.

[4] Junjie Bai, Fang Lu, Ke Zhang, et al., "ONNX: Open neural network exchange." Available: https://github.com/onnx/onnx, 2020.

[5] S. Luo, "Customization of a Deep-Learning Accelerator," *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–2, 2019.

[6] K. Kim, H. Mahmoodi, and K. Roy, "A Low-Power SRAM Using Bit-Line Charge-Recycling," in *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 446–459, Feb. 2008.

[7] N. Huang, S. Chen, and K. Wu, "Sensor-Based Approximate Adder Design for Accelerating Error-Tolerant and Deep-Learning Applications," *Design, Automation, & Test in Europe Conference & Exhibition (DATE)*, pp. 692–697, 2019.

[8] B. Reagen et al., "Minerva: Enabling Low-Power, Highly-Accurate Deep-Neural-Network Accelerators," *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, pp. 267–278, 2016.

[9] S. -M. Liu, L. Tang, N. -C. Huang, D. -Y. Tsai, M. -X. Yang, and K. -C. Wu, "Fault-Tolerance Mechanism Analysis on NVDLA-Based Design Using Open Neural Network Compiler and Quantization Calibrator," *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–3, 2020.

[10] Large-Scale Visual Recognition Challenge 2012 (ILSVRC2012) [Online]. Available: http://image-net.org/challenges/LSVRC/2012/.

[11] ONNC-CIM Tutorial. [Online]. Available: https://github.com/ONNC/onnc-tutorial/blob/master/ISCA2020-slides/ISCA2020_ONNC_CIM.pdf, 2020

[12] M. Lin et al., "DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning," *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1-8, 2018