Runtime Long-Term Reliability Management Using Stochastic Computing in Deep Neural Networks

Yibo Liu*, Shuyuan Yu*, Shaoyi Peng*, Sheldon X.-D. Tan*
* Department of Electrical and Computer Engineering, University of California, Riverside, CA 92521, stan@ece.ucr.edu

***Abstract*—In this paper, we propose a new dynamic reliability technique using an accuracy-reconfigurable stochastic computing (ARSC) framework for deep learning computing. Unlike the conventional stochastic computing that conducts design time accuracy power/energy trade-off, the new ARSC design can adjust the bit-width of the data in run time. Hence, the ARSC can mitigate the long-term aging effects by slowing the system clock frequency, while maintaining the inference throughput by reducing the data bit-width at a small cost of accuracy. We show how to implement the recently proposed counter-based SC multiplication and bit-width reduction on a layer-wise quantization scheme for CNN networks with dynamic fixed-point data. We validate an ARSC-based five-layer convolutional neural network designs for the MNIST dataset based on Vivado HLS with constraints from Xilinx Zynq-7000 family xc7z045 platform. Experimental results show that new ARSC DNN can sufficiently compensate the NBTI induced aging effects in 10 years with marginal classification accuracy loss while maintaining or even exceeding the pre-aging computing throughput. At the same time, the proposed ARSC computing framework also reduces the active power consumption due to the frequency scaling, which can further improve system reliability due to the reduced temperature.**

Experimental results show that new ARSC DNN can sufficiently compensate the NBTI induced aging effects in 10 years with marginal classification accuracy loss while maintaining or even exceeding the pre-aging computing throughput. At the same time, the proposed ARSC computing framework also reduces the active power consumption due to large frequency scaling, which can further improve system reliability due to the reduced temperature.

I. Introduction

One of the critical paradigm changes for today's emerging computing workloads, such as deep learning, computer vision, imaging, and audio processing, is that accurate computing becomes less critical since those applications are much more error-tolerant with analog-like outputs for human interaction. As a result, accuracy can be traded off to improve hardware footprint and power/energy efficiencies via approximate computing. One crucial approach for approximate computing is by means of stochastic computing (SC), which presents the value as the signal probability in a bit-stream instead of the traditional binary number [1]. Now SC is shown to have better error resilience, progressive trade-off among performance, accuracy, and energy, as well as cheap implementation of complex arithmetic operations.

Today's digital systems are built on less reliable devices and less robust interconnects as technology node advances. The major reliability effects for VLSI chips include Bias Temperature Instability (BTI), hot carrier injection (HCI) for CMOS devices, electromigration (EM) and time-dependent dielectric breakdown (TDDB) for interconnects and dielectrics, which are the primary consideration for the aging effects [2], [3]. We show the estimated BTI aging impacts on a counter-based SC multiplier's maximum frequency based on the Nangate 45 nm degradation-aware standard cell library from Karlsruhe Institute of Technology (KIT) [4] in Fig. 1. Those aging and long-term reliability effects are getting worse with shrinking feature sizes, and future chips will show signs of aging much faster than the previous generations [5]. To mitigate the increasing reliability and resiliency problems, traditional long-term reliability and aging analysis mainly focus on the reliability optimization at the design time of the system and physical level [6]. Recently using less accurate computing to compensate for the NBTI-induced long-term aging effects has been proposed [7]. However, this method targets at the design time so that sufficient margins can be allocated in advance.

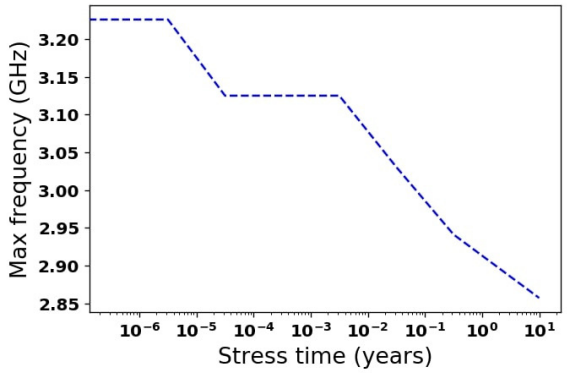

Fig. 1: The maximum working frequency decreases over years because of aging based on the Nangate 45 nm degradation-aware standard cell library from Karlsruhe Institute of Technology (KIT) [4].

On the other hand, stochastic computing has been emerging a new computing paradigm due to its low-cost and error-resilient features. One of the significant benefits for SC is that many arithmetic operations, such as multiplication, can be simply implemented by an *AND* operation (or *XNOR* gate for bipolar coding). Nowadays, SC has been applied to error-correcting codes [8], image processing [9], and recently deep neural networks (DNNs) [10]–[13].

Traditional SC, however, suffers long computing time and high randomness of the stochastic numbers for accuracy. As a result, many research works have been proposed to mitigate those shortcomings, such as high-quality random number generators (RNGs) that exhibit zero or close to zero correlation, including low-discrepancy sequences [14], bit scrambling methods [15], [16]. A more efficient and accurate SC multiplier was recently proposed to partially mitigate the traditional SC's two mentioned problems [12]. Instead of using an *AND* gate to multiplicate two bit-streams, the new multiplier counts the number of ones in part of one bit-stream. And the length of the bit-stream being counted depending on the value of the other bit-stream. Furthermore, the bit-stream to be counted can be generated in a deterministic way. As a result, the whole design is simplified into two counters and a simple bit-stream generator. In this work, we call this design *counter-based SC multiplier* (CBSC-Multiplier). CBSC-Multiplier brings two significant benefits: first, it does not require two bit-streams randomness anymore without accuracy loss. Second, it can be faster than the traditional SC as it only counts parts of the bit-stream, instead of the entire bit-stream.

Based on those observations, in this paper we propose a new accuracy-reconfigurable stochastic computing (ARSC) framework for dynamic VLSI systems reliability management. We focus on the deep learning workloads and try to leverage the latest ARSC framework to mitigate VLSI hardware's long-term reliability issue due to several degradation effects such as biased temperature instability (BTI) and electromigration (EM). Our contributions are as follows:

- Unlike the existing works that carry out the design time accuracy versus power/energy trade-off, the new stochastic computing can adjust the data bit-width in the run time. Hence it can accommodate the long-term aging effects by aggressively slowing down the system clock frequencies while maintaining the inference throughput by reducing the data bit-width.
- We show how the recently proposed CBSC multiplication and bit-width reduction can be implemented on a layer-wise quantization scheme for CNN networks with dynamic fixed-point data.
- We design and validate an ARSC-based five layer convolutional neural network design for MNIST dataset based on Vivado HLS with constraints from Xilinx Zynq-7000 family FPGA platform.
- Experimental results show that new ARSC based DNN can mitigate the long-term aging-induced effects using simple frequency scaling while maintaining the inference throughput with marginal classification accuracy loss. Specifically, we show that one-bit precision reduction (which has negligible classification accuracy loss) for input data can sufficiently compensate the NBTI induced aging effects in 10 years while maintaining the pre-aging computing throughput.
- The proposed ARSC computing framework also reduces the active power consumption due to frequency scaling, which can further improve system reliability due to the reduced temperature.

This paper is organized as follows: Section II reviews some related works such as conventional SC, CBSC etc. Section III presented the proposed ARSC concept for the DNN accelerator design so that we can trade the accuracy to mitigate long-term reliability issues. Section IV presents some implementation details on the FPGA platform. Experimental results and discussion are summarized in Section V. Section VI concludes this paper.

II. Preliminaries

Many techniques have been proposed to address the high computational complexity of DNN networks [17]. Most of the existing methods mainly focus on design time trade-off between energy efficiency or throughput and application accuracy at circuit, architecture, and even system levels. Techniques includeing more energy-efficient data flow, more efficient quantization, precision reduction/compression, and weight pruning etc. have been proposed. However, fewer efforts were investigated for run time accuracy-energy/throughput trade-off for DNN applications. In this section, we briefly review CBSC, which are the basis for the proposed work.

A. Conventional stochastic computing:

Stochastic computing (SC) provides an alternative way for arithmetic operation when the exact results are not required. At the same time, the SC-based hardware can be designed with much lower cost and power compared to the conventional binary digital designs.

Fig. 2 shows the conventional SCmultiplier, where the value of stochastic number, SN, is represented by a bit-stream, whose signal probability, or frequency of bit '1', determines its value. Naturally, the value is defined in the range $[0, 1]$, called unipolar, or over $[-1, 1]$ called bipolar. For instance the number X represents $4/8$ as we have four '1' in the 8-bit bit-stream. One of the major benefits of SC is that the multiplication can be simply implemented by an AND gate as shown in Fig. 2.

To generate the random number for SC, stochastic number generator, SNG, which essentially converts a binary number to stochastic number, takes an n-bit binary number and generates the random bit-stream as shown in the bottom part of Fig. 2. The SNG is typically implemented by n-bit linear feedback shift register (LFSR) and a comparator, which generates '1' if the random number is less than the input binary number, and '0' otherwise.

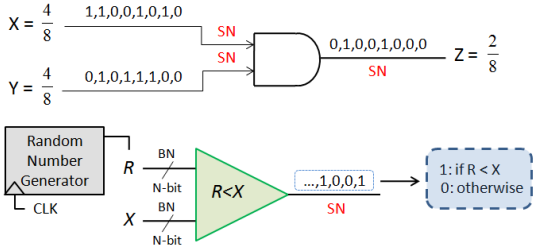

Fig. 2: Traditional SC number and multiplier.

For SC unipolar encoding, the multiplication can be done by AND operation and for bipolar encoding, the multiplication is achieved by $XNOR$ operation [1], [18]. The addition can be simply done by a multiplexer (MUX) [1], [18]. Finally, the resulting bit-stream can be converted back to a binary number by using a counter (or up-down counter for bipolar coding).

Due to its simple hardware implementation compared to the common arithmetic operations, SC is very low-cost and energy-efficient. But the traditional SC, however, suffers from long latency and inherent random fluctuation errors. The problems can be mitigated by the recently proposed CBSC method mentioned below.

B. CBSC multiplication

Assume the bit-width is n for the given two binary numbers x and w. The conventional SC multiplier using *AND* gate (for unipolar encoding) will take 2^n, which is the length of bit-stream of SN, cycles to finish the computation. To improve this, Sim *et al.* in [12] proposed the CBSC multiplier design shown in Fig. 3. The multiplier mainly consists of two counters. The first counter counts the binary value of the input w and the second counts the result of $x \cdot w$. So the operation only takes $w \cdot 2^n$ cycles to finish, in which $w < 1$, which translates to better performance than the conventional SC. One example is given in Fig. 3. Also the stochastic number of input x can be generated in a deterministic way without hurting the accuracy (actually more accurate) compared to the conventional SC. As a result, such design is more straightforward as we eliminate the two traditional SNGs (typically using Linear Feedback Shift Registers) and *AND* gates in exchange of a counter, which is much cheaper than SNG.

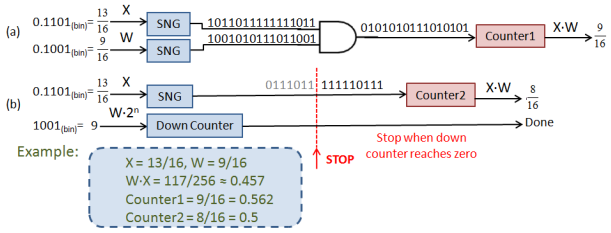

Fig. 3: (a) Conventional SC multiplier. (b) CBSC multiplier concept [12].

To generate the SN of x, which is a low-discrepancy bit-stream, the authors proposed a deterministic way to do this. The method evenly distributes the $x_{i-1}, i \in [1, n]$, which is the i_{th} bit of x, based on its binary weight 2^{i-1}. For instance, if $i = 3$, then x_2 will appear 4 times in the resulting SN bit-stream as shown in Fig. 3. Such a stochastic number generation can be simplified and implemented by an FSM and an MUX and the whole CBSC multiplication design is shown in Fig. 4 [12].

But for our problem, we propose to use fixed-point numbers for CBSC computing (more discussion on this later). The value of a SN bit-stream is always between 0 to 1, hence the input is enlarged 2^{BW} times to become a fixed-point number before put into the multiplier. Suppose the two fixed-point numbers are $x = 13$ and $w = 9$, then the multiplication result is $x \cdot w = 117$. Now if we use CBSC, we then

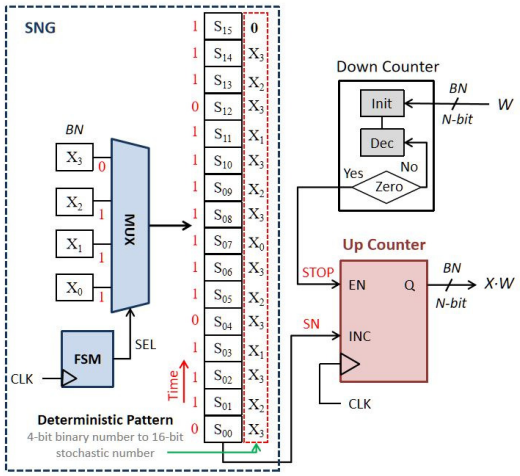

Fig. 4: Counter-Based SC Multiplier.

use the SNG in Fig. 4 to generate the SN and put 9 into the down counter to perform the SC computing. The counting result is 8 based on Fig. 3. Then the final multiplication result will be $8 \cdot 2^4 = 128$, where 4 is the bit-width. Hence the CBSC computing essentially can be viewed as

$$(x \cdot w)_{\text{CBSC}} = \text{round}(x \cdot \frac{w}{2^{BW}}) \cdot 2^{BW} \quad (1)$$

The CBSC will introduce the random error which depends on the value of the result during the computing process. For signed CBSC computing, the multiplication will become $(x \cdot w)_{\text{CBSC}} = \text{round}(x \cdot w/2^{BW-1}) \cdot 2^{BW-1}$.

III. Proposed Dynamic ARSC Deep Neural Networks

In this section, we present the proposed accuracy-reconfigurable stochastic computing (ARSC) for the DNN accelerator design so that we can trade the accuracy to mitigate the long-term reliability issues. The main idea is to *dynamically adjust the bit-width* for multiplication intensive convolutional computing to reduce the accuracy of the computing progressively using SC.

At the same time, we also reduce the effective latency of the computing logic so that we can compensate for the aging-induced delay increases. As a result, even though the operating frequency of the chip is reduced due to the aging effect, we are still able to maintain the same throughput of the whole computing process as required by the application.

A. ARSC architecture for the DNN application

The proposed DNN accelerator architecture with the ARSC-based MAC units is displayed in Fig. 5. For the sake of demonstration, we use a simple CNN network, which consists of 2 convolutional (CONV) layers and 3 fully connected (FC) layers. The 5-layer network is similar to the originally proposed LeNet [19]. Both CONV layers utilize 5×5 convolutional kernels, followed by a max pooling layer. The DNN accelerator uses dataflow optimization as the layer-wise pipeline so that the clock interval of the DNN accelerator depends on the hardware layer with the maximum interval. Note that the computation bottleneck of the DNN network of the inference is at the CONV layers, as CONV is computational intensive. Even with 5-bit precision, the CONV layer has longer operation intervals than the FC layer.

The Fig. 6 illustrates the proposed ARSC MAC unit where the input data elements are fetched from the *Input Buffer* and truncated to the desired bit-width. Then the CBSC multiplier takes the input data and weight element to implement SC multiplication [12]. Finally,

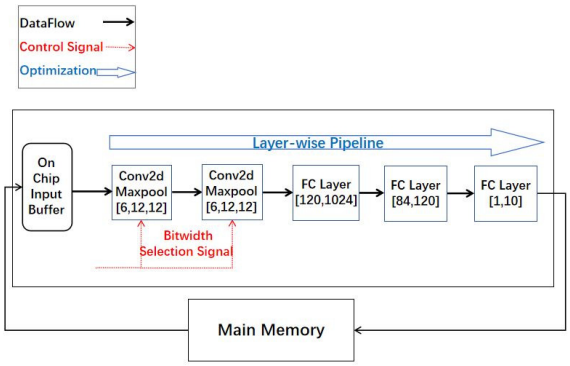

Fig. 5: The proposed DNN accelerator architecture with the ARSC-based MAC unit

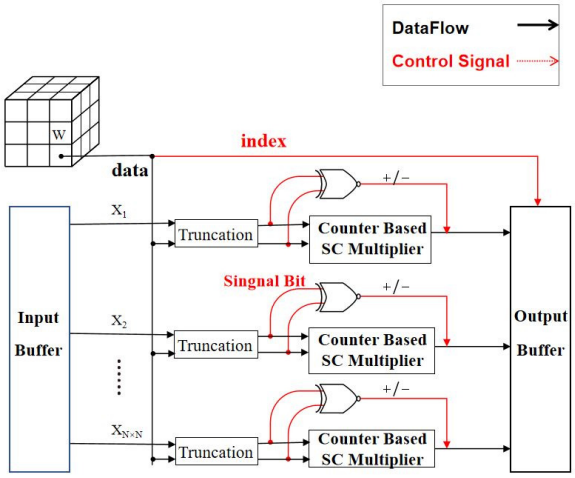

Fig. 6: The proposed ARSC-based MAC unit

add the current multiplication product to the corresponding location in the Output Buffer. The proposed dynamic accuracy reconfiguration function of the ARSC module is enabled by adjusting the input data truncation bit-width before the SC multiplier.

For an N-bit input data including a sign bit, we need an x-bit selection signal to indicate how many bits to keep for the N-bit input data, in which x=$\lceil \log_2(states - 1) \rceil$. The $state$ represents the number of all possible CBSC bit-width selections. For example, in our design, the CBSC multiplier is designed to work from 8-bit to 5-bit configurations. Hence the $states$=4 and x=2 will be enough to distinguish these four states. The truncation here keeps the sign bit and truncate the least significant bits from the right side, which is compatible with the bit-width based progressive SC computing scheme.

At the end of ARSC multiplication, we add back the sign bit to the result and add 0 at the end of the output number by left shifting the multiplication result. These make the result have the same bit-width as the input data, which keeps the data bit-width consistent among different layers.

An N-bit signed binary number will be converted to a 2^{N-1} bits SN bit-stream. Since the computation load of CBSC is proportional to the SN bit-stream length, a one-bit truncation in binary number will cut down half of the SN bit-stream length. The significant SN bit-stream length reduction effectively reduces the computing latency, thus mitigating the aging effects induced frequency degradation.

Note that our CBSC multiplier should only deal with uni-polar number multiplication within $[0, 1]$ range. We use the dynamic fixed-

point datatype in the DNN accelerator, so we do not need to do any scaling or other adjustments to the CBSC input data. However, we need to left-shift the result to convert it back to the dynamic fixed-point datatype before putting the result into the output buffer. When we truncate more bits from the input data, the resulting error from the CBSC multiplier will be larger. Still use the example in Fig. 3, if the original input data is $9/16$ and $13/16$, truncate 1 bit will change the input value to $4/8$ and $6/8$, which increase the computation error. However, from the DNN application perspective, such errors may not lead to significant accuracy loss as shown in the experimental section.

The leftmost bit of data is the sign bit. We keep the sign bits for all the weights w_j and input feature data x_i to perform an XOR operation to determine the sign bit of the result obtained from the CBSC multiplier. Then we add back the sign bit to the result at the end of ARSC multiplication.

From Fig. 4, we know that the computation time of the CBSC multiplication is proportional to the weight w_j. If we perform the convolution multiplication in parallel with all weight elements in the kernel, the computation time depends on the largest weight element. To mitigate this problem, we follow a similar idea of BISC based matrix vector multiplier (MVM) as [12]: We change the convolution loop order so that the multiplication operations that use the same weight element should carry out simultaneously, which is shown in Fig. 6. This contrasts with the conventional method where the products of the input feature map x_i and the filter kernel w_j are summed up at once to obtain the output element.

B. Dynamic fixed-point data for ARSC

A typical 16-bit fixed-point data type is usually sufficient for training neural networks for hardware accelerators with no loss in classification accuracy [20]. 8-bit precision is sufficient for inference with minimal accuracy error loss [21]. In this work, we use the dynamic fixed-point data type for each CONV layer [22] .

A dynamic fixed-point x stores a value as follows:

$$value = (-1)^s \cdot \sum_{i=0}^{BW-2} 2^i \cdot x_i \cdot 2^{-fl} \tag{2}$$

where x_i is the ith bit of data x. Here BW is the bit-width with the highest bit as the sign bit. 2^{-fl} is the scale factor and fl is determined by the floating range and quantized range for each CONV layer. The equation (2) can be viewed as an ordinary binary fixed-point number multiplied by a scale factor. As a result, for dynamic fixed-point data, we only needs the integer operation and bit shifting operations in DNN computing, which is more efficient.

To convert a floating-point number to a dynamic fixed-point data, we perform the following symmetric quantization scheme for both weights and input activation:

$$x_{\text{quant}} = \text{round}(\frac{x_{\text{float}}}{2^{BW-1-fl}} \cdot (2^{BW-1} - 1)) \tag{3}$$

If we define $f_{\text{up}} = \text{argmin}(2^{f_{\text{up}}} > x)$, which means $2^{f_{\text{up}}}$ is the minimum value that is larger than the floating data x of interest. For example, if the maximum weight is 3.2, then the minimum power of two can cover 3.2 is 4, hence $f_{\text{up}} = 2$. BW is the bit-width of the quantized number with a sign bit, then $fl = BW - f_{\text{up}} - 1$.

Now, we perform some error analysis for ARSC computing. If we use the dynamic fixed-point data type in regular multiplication operation, the multiplication result is $x_{\text{quant}} \cdot w_{\text{quant}}$. The measurement error of x_{quant} or w_{quant} is within there quantization stepsize.

As discussed in subsection II-B, if we use the CBSC multiplier with the same bit-width, the result is $round(x_{\text{quant}} \cdot w_{\text{quant}}/2^{BW-1}) \cdot 2^{BW-1}$. When we truncate 1 bit from the data (weight or activation), the CBSC multiplication result becomes:

$$\text{round}(\frac{x_{\text{quant}}}{2} \cdot \frac{w_{\text{quant}}}{2^{BW}}) \cdot 2^{BW+1} \tag{4}$$

Using the same example in Fig. 3, the 5-bit signed CBSC multiplication has input $x = 13$, $w = 9$ and $BW = 5$. When we reduce the CBSC input and computation bit-width from 5-bit to 4-bit, which means $BW_{new} = 4$ and truncate x from 13 to 6, truncate w from 9 to 4, then based on (4), the CBSC counter output will be 3. The final result is $3 \cdot 2^5 = 96$. As we can see, one-bit truncation will enlarge the computation error because the output result resolution has dropped from 2^{BW} to 2^{bw+1}. The floating number can be approximately recovered from the quantized number by simply multiplying the scale factor:

$$x_{\text{float}} = x_{\text{quant}} * 2^{-fl} \tag{5}$$

For the proposed ARSC DNN accelerator, we adopt the layer-wise quantization scheme in which each CNN layer has its quantization scale factor. In this way, we replace the floating number multiplication by the bit-shifting operation to reduce the DSP occupation and improve the throughput.

For the stochastic computing, it was well known that the random errors in multiplication become significant when the result is close to zero as [10]. However, due to the L1 and L2 regulation, many weights are trained to have normal distribution around zero [23]. To mitigate this problem, following a similar strategy as [10], we try to remove near-zero weight and perform the re-training to avoid this issue. We remove near zero weights if the weight is less than a threshold W_{th} as defined below:

$$W_{th} = \alpha \cdot \text{std}(W^k) \tag{6}$$

where $\text{std}(W^k)$ indicates the standard deviation of weights in layer k. α is determined experimentally. After such near-zero weight removal, re-training is carried to mitigate the accuracy loss.

We note that to make this proposed run time aging mitigation method work for practical VLSI processors, one needs to have on-chip aging sensors such as the oscillation based BTI aging sensor [24]. Then an online intelligent controller will be employed to map the aging sensor reading to the required actions for frequency adjustment and bit-width reduction configuration of the DNN layers in the run-time. But the main goal of this paper is to demonstrate the feasibility of such dynamic aging mitigation strategy based on the ARSC framework. Also due to page limitation, the specific dynamic aging controller design based on the ARSC framework will be presented in our future work.

IV. Implementation

The training and inference processes of CBSC-embedded DNN are performed on different platforms separately. Before we conduct the CBSC in DNN accelerator, we implement the training and adoption process with methods mentioned in Sec. III, which includes the initial model training, near-zero weight removal, and data quantization. The training and adoption process are implemented on local PC platform with Pytorch 1.5.0. We summarize this process in Fig. 7. The initial training step generates a normal DNN network with floating type parameters and activations. For large networks, the initial training step can also be replaced by loading pre-trained model parameters to speed up the design process. Then we remove the weight parameters within the threshold to adopt the model to SC. Similar to [10], we start from a small threshold W_{th} and increase it based on the accuracy test result. The accuracy test step emulates the fixed-point datatype calculation behavior in python, so that we can directly use the server platform to evaluate the current model without switching to other platforms. The quantization step here converts the model data from floating-point to fixed-point, the quantized results can be applied in both accuracy emulation test and the final hardware implementation.

To evaluate the performance of the ARSC based DNN module, including delay, power and resource occupation, we implement the proposed design in C and synthesized by Xilinx vivado HLS for XC7Z045 device of Zynq-7000 family, and obtain resource utilization

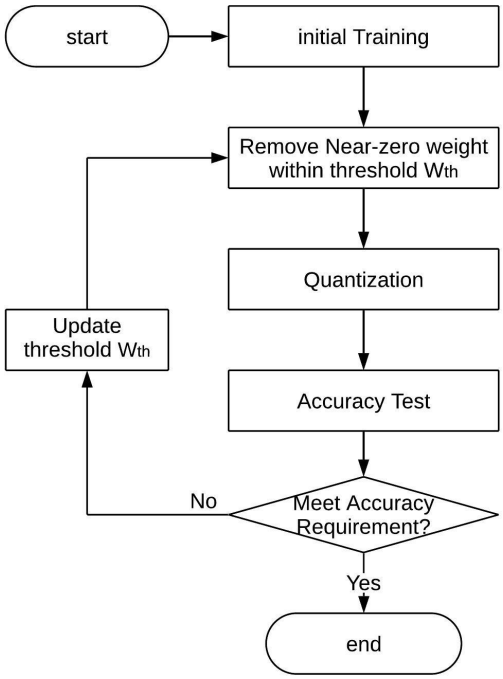

Fig. 7: The training process

#BRAM18K	#DSP48E	#FF	#LUT
18	4	27711	111917

TABLE I: ARSC-based DNN resource utilization. Note: BRAM18K: 18K Block RAM, DSP48E: DSP 48E slice, FF: flip-flop in Zynq 7000 FPGA

report, and critical path delay from the synthesis report. The details of the hardware resource utilization information are show in Table. I.

As mentioned in Sec. III, we use parallel SC blocks to accelerate the computation and improve the throughput. We also apply loop-wise pipeline to further increase the throughput with negligible extra resource occupation. Different from the ASIC-based module, FPGA mainly use LUT to perform arithmetic operations.

We use the Xilinx Power Estimator to evaluate the power consumption, which can easily obtain the total power consumption by hardware resource utilization and the desired frequency. We'll discuss the power consumption of the design later in Sec. V.

For the delay and latency evaluation, Vivado HLS synthesis reports provide the layer-wise critical path and latency. With layer-wise dataflow optimization, the latency of the DNN network will roughly be the maximum layer-wise latency. The resulting critical path of the 5-layer DNN design, which is also calculated from the worst layer critical path, is 9.51ns, which means that the our ARSC-based DNN can run under the highest frequency of 105.3MHz. We will also analysis the throughput effect from CBSC bit-width adjustment in the Sec. V.

V. Experimental Results and Discussion

In this section, we present results from the proposed ARSC computing DNN accelerator design for long-term aging management. The proposed CNN network has 2 CONV layers and 3 FC layers as we mentioned before. The 5-layer CNN is used as the demonstration of the proposed concept and we remark that the proposed method ARSC can be applied to general DNN networks. We trained the proposed ARSC based CNN on the server for the MNIST dataset [19] which consists of 50000 training images and 10000 testing images.

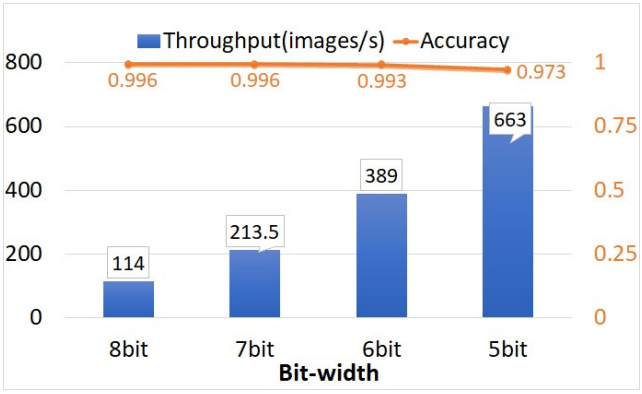

Fig. 8: The throughput and accuracy versus different bit-width

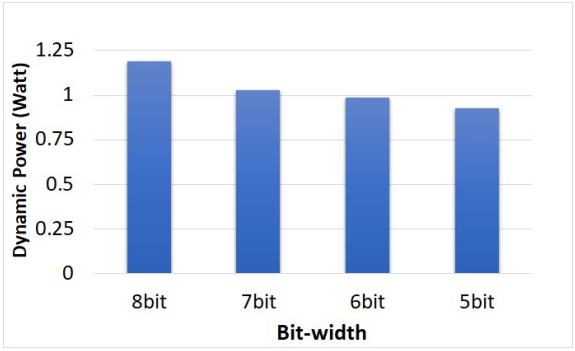

Fig. 9: The dynamic power versus different bit-width

To estimate the frequency decrease caused by the NBTI induced aging process, we refer the time-dependent degradation aware Nangate 45nm standard cell library from Karlsruhe Institute of Technology (KIT) [4] to calculate the chip frequency within 10 years. We use Synopsys design suite to synthesize the ARSC-based DNN design. The timing analysis shows that the working frequency of the ASIC-based design of the DNN will decrease from 3.23GHz to 2.86GHz after 10 years, as shown in Fig. 1. The aging process have negative impact on the throughput, which can be simulated on FPGA by slowing the clock frequency with same percentage as from 3.23GHz to 2.86Ghz in the mentioned ASIC-based aging analysis. Adjust the frequency from 105.3MHz to 93.2MHz, in which 105.3MHz is the original FPGA clock frequency and 93.2Mhz is determined by decreasing same proportion as the ASIC-based design frequency drops from 3.23GHz to 2.86GHz. Note that such mapping may not be perfect as the design technologies used in our ASIC and FPGA are different. The above aging effect decreases the DNN accelerator inference throughput. Hence, we apply the proposed ARSC to mitigate the throughput deterioration by decreasing the bit-wdith of SC. As mentioned in Sec. II, decreasing 1-bit in the SC bit-width will approximately reduce half of the computation workload and almost double the throughput, which is enough to make up throughput decreasing caused by NBTI induced frequency decline. The lower the bit width of SC, the more the improvement of throughput. The Table II shows the frequency for different SC bit widths to maintain the same throughput as 8-bit SC in original 105.3MHz. For example, when the FPGA frequency drops to no less than 55.93MHz, reducing the SC bit width from 8-bit to 7-bit is enough to keep the throughput no lower than its original value.

Bit-Width	Frequency (MHz)	Power (W)	Accuracy
8	105.26	1.190	0.996
7	55.93	1.026	0.996
6	30.65	0.986	0.993
5	18.00	0.925	0.973

TABLE II: Key performance metric comparison under the same throughput.

Based on interval clocks cycles that comes from simulation results of different data bit-width on Vivado HLS, we calculate the throughput at different clock frequencies in Fig. 8. As we can see by sacrificing 3-bit precision, which means that we use 5-bit precision to do the inference based on the weights obtained from the 8-bit precision, the inference classification will only decrease 2.3% in accuracy. At the same time, we can increase the throughput (number of images per second) substantially by 482%. The inference throughput and accuracy under different bit-width is shown in Fig. 8. As we can see, initially (when the aging process hasn't started yet), if we use the full 8-bit precision for the inference, the throughput at 105.3MHz clock frequency is 114 image per second. When the clock frequency decrease due to the aging process, the throughput will decrease to 101. We can make up for the throughput if we truncate the precision by only one bit (from 8 to 7), which makes the throughput increase to 189. Furthermore, by decreasing the precision of SC multiplication to 5-bit, the throughput will be about 5.8 times of the 8-bit precision, which shows huge potential we can mitigate the aging effects if such accuracy is still accepted in practical applications.

From Fig. 9, we analyze the power consumption under different clock frequency by Xilinx Power Estimator. Estimation results show that when making the trade off between the throughput and the accuracy mentioned before, we can save near 12.3% of total device power and 22.2 % of ARM core and uncore (also called Processing System or PS) and FPGA related dynamic power in XC7Z045 device of Zynq-7000 FPGA, which hosts the DNN design. As active power can be reduced when frequency is scaled down, we can further improve the long-term reliability due to the decreased chip temperature [25].

VI. Conclusion

In this paper, we proposed a novel accuracy-reconfigurable stochastic computing (ARSC) framework for dynamic reliability management for deep neural network applications. The new ARSC design can dynamically change accuracy via bit-width change of the data. In this way, the new method can accommodate the long-term aging effects by slowing the system clock frequency at the cost of small accuracy loss while maintaining the throughput of the computing. We showed how the recently proposed CBSC multiplication and bit-width reduction can be implemented on a layer-wise quantization scheme for CNN networks with dynamic fixed-point data. We designed and validated the ARSC-based 5-layer convolutional neural network designs for MNIST dataset based on Vivado HLS with constraints from Xilinx Zynq-7000 family xc7z045 FPGA platform. Experimental results showed that new ARSC based DNN can sufficiently compensate the NBTI induced aging effects in 10 years with marginal classification accuracy loss while maintaining or even exceeding the pre-aging computing throughput. At the same time, the proposed ARSC computing framework also reduce the active power consumption and thus on-chip temperature by large frequency scaling, which further contributes to the aging mitigation.

For the future work, we will investigate the an online intelligent controller and manage algorithms to map the aging sensor reading to the required actions for frequency adjustment and bit-width reduction configuration of the DNN layers in the run-time.

References

[1] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515–1531, 2018.

[2] "Failure Mechanisms and Models for Semiconductor Devices." In JEDEC Publication JEP122-A, Jedec Solid State Technolgy Association, 2002.

[3] "Critical Reliability Challenges for The International Technology Roadmap for Semiconductors (ITRS)," 2003. In International Sematech Technology Transfer Document 03024377A-TR, 2003.

[4] "Degradation-aware cell libraries, v1.0." http://ces.itec.kit.edu/dependable-hardware.php.

[5] S. X.-D. Tan, H. Amrouch, T. Kim, Z. Sun, C. Cook, and J. Henkel, "Recent advances in EM and BTI induced reliability modeling, analysis and optimization," *Integration, the VLSI Journal*, vol. 60, pp. 132–152, Jan. 2018.

[6] S. X.-D. Tan, M. Tahoori, T. Kim, S. Wang, Z. Sun, and S. Kiamehr, *VLSI Systems Long-Term Reliability – Modeling, Simulation and Optimization*. Springer Publishing, 2019.

[7] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Towards aging-induced approximations," in *Proceedings of the 54th Annual Design Automation Conference 2017*, pp. 1–6, 2017.

[8] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed stochastic decoding of ldpc codes," *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5617–5626, 2011.

[9] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 449–462, 2013.

[10] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2016.

[11] H. Sim, D. Nguyen, J. Lee, and K. Choi, "Scalable stochastic-computing accelerator for convolutional neural networks," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 696–701, IEEE, 2017.

[12] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2017.

[13] R. Hojabr, K. Givaki, S. Tayaranian, P. Esfahanian, A. Khonsari, D. Rahmati, and M. H. Najafi, "Skippynn: An embedded stochastic-computing accelerator for convolutional neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, p. 132, ACM, 2019.

[14] S. Liu and J. Han, "Energy efficient stochastic computing with sobol sequences," in *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 650–653, European Design and Automation Association, 2017.

[15] F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *2017 Euromicro Conference on Digital System Design (DSD)*, pp. 1–8, IEEE, 2017.

[16] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 256–261, IEEE, 2016.

[17] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[18] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*, pp. 37–172, Springer, 1969.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[20] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, pp. 1737–1746, 2015.

[21] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[22] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv: Learning*, 2014.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016. http://www.deeplearningbook.org.

[24] M. Tehranipoor, H. Salmani, and X. Zhang, *Integrated Circuit Authentication*. Springer, 2014.

[25] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "Ramp: A model for Reliability Aware Microprocessor Design," *IBM Research Report*, 2003.