Parallel Operation of Self-Limited Analog Programming for Fast Array-Level Weight Programming and Update

Hanchan Song, Jangho An, Seoil Son, Young Seok Kim, Juseong Park, Jae Bum Jeon, Geunyoung Kim, and Kyung Min Kim*

Memristive neural networks perform vector matrix multiplication efficiently, which is used for the accelerator of neuromorphic computing. To train the memristor cells in a memristive neural network, the analog conductance state of the memristor should be programmed in parallel; otherwise, the resulting long training time can limit the size of the neural network. Herein, a novel parallel programming method using the self-limited analog switching behavior of the memristor is proposed. A Pt/Ti:NbO\(x\)/NbO\(x\)/TiN charge trap memristor device for the programming demonstration is utilized, and a convolutional neural network is emulated to train the MNIST dataset, based on the device characteristics. In the simulation, the proposed programming method is able to reduce programming time to as low as \(1/130\), compared with the sequential programming method. The simulation suggests that the programming time required by the proposed method is not affected by array size, which makes it very promising in a high-density neural network.

1. Introduction

The use of memristive neural networks for neuromorphic computing has attracted enormous interest recently.\(^{[11-11]}\) In memristive neural networks, memristor crossbar arrays are used as artificial neural networks, where the conductance of the memristor at each crosspoint refers to the weight value of the synapse. To mimic synaptic behaviors such as long-term potentiation (LTP) and long-term depression (LTD), or spike timing-dependent plasticity (STDP), the memristor should have analog conductance switching characteristics.\(^{[12-14]}\)

Self-rectification is also highly desirable to eliminate sneak current-associated issues, such as programming and sensing disturbances, and higher energy waste while driving the crossbar array device. In this regard, a charge trap memristor, whose structure and operating principle are similar to those of a charge trap flash memory, is very suitable for memristive neural networks because of its low current analog switching and self-rectifying characteristics.\(^{[13-19]}\) Also, its forming-free characteristic can eliminate the stochastic nature of the forming process, and thus lead to higher uniformity, along with simpler device operation.\(^{[20-23]}\)

Authors have reported self-limited analog programming (SAP) method that can be used to program the analog data in the charge trap memristor efficiently.\(^{[24]}\)

In the SAP method, the memristor \(M\) is initially in a high-resistance state (HRS) and serially connected to a variable resistor \(R_s\). Once a programming voltage \(V_{PGM}\) is applied in the \(M-R_s\) configuration, the conductance of the memristor gradually increases and the voltage potential on the memristor \(V_M\) decreases accordingly, due to the voltage divider effect produced by the constant \(R_s\).\(^{[25-27]}\) Eventually, the memristor conductance is saturated at a specific value and the self-limited set switching is achieved; that is, the \(R_s\) limits the total current, such as a compliance current. Then, by appropriately controlling the resistance of \(R_s\), the saturated conductance state of \(M\) can be controlled. The SAP method, which allows the \(V_{PGM}\) to be fixed and changes the \(R_s\) allows faster and more accurate programming of the analog states than the conventional programming method, which changes the amplitude of \(V_{PGM}\) or changes the number of pulses.

During the training of the memristive neural network, the weight values of all cells are programmed dynamically and actively. In the conventional programming method, each cell must be programmed sequentially with a sequence of write-and-verify steps.\(^{[5,28-30]}\) In that approach, the total weight programming (or updating) time is proportional to the array size. Thus, reducing programming time is a significant issue in a high-density memristive neural network. The parallel programming capability of the crossbar array can resolve this problem.

Gao and co-workers proposed a parallel write-and-read scheme for a memristive crossbar array and showed that switching speed could be theoretically improved by 30 times at an array size of \(100 \times 300\).\(^{[31]}\) Although the study showed the parallel programming method could reduce programming time, many unresolved issues remain before the parallel programming method that can be implemented in a practical system. For example,
the variation of the LTP and LTD, and sneak currents needs to be considered in the sizeable array-level analysis. Also, the method should be able to compensate for uneven weight update behavior, which depends on the previous states during the training process, due to the nonlinear and nonsymmetric behavior of the LTP and LTD of the synaptic device. Furthermore, the method needs to work in the presence of cycle-to-cycle and device-to-device variations. Otherwise, it will not be possible to realize a high-performance memristive neural network with sufficient programming accuracy and efficiency.

In this study, we show the parallel operation capability of the SAP method (which we designate SAPP, or SAP in parallel) in a crossbar array utilizing a Pt/Ti:NbO$_x$/NbO$_x$/TiN charge trap memristor (Figure S1, Supporting Information). We developed a novel biasing scheme for the SAPP and its optimized operation algorithm, and examined how SAPP reduced programming time for various array sizes and number of analog levels. We used the optimized SAPP method in the on-chip training of a convolutional neural network (CNN) and evaluated the improvement in its programming efficiency.

2. SAP in Parallel

The SAP method performs data programming sequentially; therefore, the total programming time is proportional to the array size (Figure S3, Supporting Information). Considering the high frequency of weight update events during the training, the increase in programming time at larger array sizes will eventually degrade the high-performance neuromorphic computing.

This programming speed issue can be resolved by performing the SAP operation in parallel (SAPP). Figure 1a shows a crossbar array layout and biasing conditions for SAPP to program multiple selected cells along the selected word (vertical) line to the target conductance state. Herein, the selected word line in the memristor array is biased to a programming voltage ($V_p$), the selected word lines in the resistor array are grounded ($V_G$), and all unselected word lines are floated ($F$), which are identical to the biasing conditions used for the SAP operation. The only difference between SAPP and SAP operation is the biasing condition of the bit (horizontal) lines. For parallel operation, the multiple bit lines connected to the multiple target cells to be programmed are floated together. In Figure 1a, for example, two cells (yellow dashed squares) are selected. Other unselected bit lines are biased to $\frac{1}{2} V_p$. Figure 1b shows an equivalent circuit of Figure 1a. In the circuit, two individual SAP units comprising one memristor ($M$) and one selected resistor ($R_{Sel}$) are present in parallel, and their middle nodes are connected through parasitic unselected resistor ($R_{Unsel}$) components. In this configuration, if the initial resistance of the two memristors and the selected $R_{Sel}$ are identical, the potential difference between the two middle nodes is zero. Then, the SAPP operation can be understood as multiple independent SAP operations at once.

However, in the practical case, the initial state of $M$ can be variable. Therefore, we examined the effect of variation in the initial state on the final programmed state during SAPP.
operation. Figure 1c shows a SAPP simulation setting where eight cells with all different initial states (from state 1, the lowest conductance state, to state 8, the highest conductance state) are to be programmed to state 8. Figure 1d shows the conductance change of each cell as a function of programming time. The dynamic conductance change characteristic was simulated based on the pulse height-dependent potentiation data of the device (Figure S2, Supporting Information). The inset enlarges the moment of the end of programming. Figure 1e shows the change in the memristor potential \( V_M \) on all memristor cells as a function of programming time. At the early time, when the initial conductance is lower, the applied voltage is higher. As a result, the lower conductance state can be quickly switched to the higher conductance state, which reduces the conductance variation over time. After a saturation time of 0.193 ms, the conductance of all cells approaches state 8, with variation reduced as low as 1%. The results confirm that the final states programmed by the SAPP operation are independent of the initial state. In the SAPP operation, all of the selected cells can be programmed together regardless of the initial states as long as the programming time is long enough. Figure 1f shows the total programming time needed to program eight cells to the target state in parallel, depending on the initial states of the cells. It shows that programming from the lowest conductance state (state 1) to the highest conductance state (state 8) takes the longest time. The experiments confirmed the simulated SAPP operation. For the experiments, an 8 x 8 crossbar array of NbO\(_x\)-based charge trap memristor was prepared. Then, eight cells along the first column were programmed to eight discrete states from the lowest to the highest conductance state by the SAP operation. The inset of Figure 1g shows the top-view image of the device. The cross-sectional area of the device was 5 \( \mu m \times 5 \mu m \).

3. Incremental Conductance Programming Algorithm for SAPP

The results in Figure 1f indicates that the challenge of fast programming is programming the highest conductance state as quickly as possible. To accomplish that, we developed a programming algorithm that partially programs the higher conductance states during programming of the lower conductance state. We call this i-SAPP, an incremental conductance programming algorithm applied to the SAPP operation. Figure 2a shows the baseline task used for the performance comparison of the programming methods, where the size of the memristor array is identical to Figure 1c, and the eight cells along the shared word line are initially programmed to state 1, and are to be programmed to eight different states. Figure 2b compares the programming algorithms of SAPP method and i-SAPP method. In both algorithms, the programming operation starts from the initialization step that resetting all cells to state 1, the lowest conductance state. Then, the first programming step performs programming of the selected cells to the next conductance state, i.e., state 2. At this stage, the SAPP method only selects cell #2 to program it to state 2, whereas the i-SAPP method selects all of the cells that will be at a higher conductance state.
conducting state than state 2, i.e., from cell #2 to cell #8. Then, programming of the next conductance state, i.e., state 3, starts from state 1 by selecting only cell #3 in the SAP method, whereas it starts from state 2 by selecting the cells that will be higher conducting states than state 3, i.e., cell #3 to cell #8. Figure 2c shows the estimated working time for executing the baseline task in Figure 2a using the SAP method, where the programming of each state is executed sequentially. The SAP method is sequential so that the next programming starts as soon as the previous programming ends; it took 1.085 ms in total to complete the task. Figure 2d shows the estimated working time using the i-SAPP method to program eight states as described previously. Here, each programming sequence starts from the previous states, and the total working time was 0.699 ms, which is about 35.6% faster than the SAP method. Figure 2e shows the total time required to program cells on a single word line as a function of the number of bit lines. The i-SAPP algorithm shows a constant programming time, regardless of the number of cells. Therefore, the benefit of the i-SAPP method increases as the array size increases. Meanwhile, interconnect wire resistance may affect the i-SAPP operation. Considering a typical resistivity of metal wires ($\rho = 10^{-12} \text{\Omega m}$) and a feature size ($F = 10 \text{nm}$) in modern technology, the wire resistance ($R_w$) between the nearest cells is $\approx 10 \text{\Omega}$. The $R_w$ value is negligible compared with the resistance range ($10^{10} - 10^{11} \text{\Omega}$) of the charge trap memristor. The simulation results confirmed the immunity of the i-SAPP operation against the wire resistance (Figure S6, Supporting Information).

4. Influence of Sneak Currents on i-SAPP

In array programming, sneak currents can affect programming time. Figure 3a shows a crossbar array layout composed of an $(n \times m)$ memristor array and an $(n \times l)$ resistor array, where $n$ is the number of rows in the array, and $m$ and $l$ are the number of columns in the memristor and resistor array, respectively. The $l$ also refers to the programmable number of analog states. Here, two leakage current paths are shown: one is from the memristor array (path 1, a blue arrow) and the other one is from the resistor array (path 2, a green arrow). Here, the former path can be negligible because the memristor possesses a self-rectifying characteristic as high as $10^6$ at the reading voltage. However, the sneak current from the resistor array (path 2) is crucial as there is no selection device with the resistors.

Figure 3b shows the self-limited saturation behavior for programming various states as a function of programming time considering (lines) and neglecting (dashed lines) the sneak currents. Here, the sizes of the memristor and resistor arrays are $8 \times 8$ and $8 \times 8$, respectively. We define the programming time ($t_{PGM}$) as the time until the conductance of $M$ reaches 99% of the target state. The square dots indicate the $t_{PGM}$, the endpoint of the programming. The saturation characteristic confirms that the SAP method works well even in the presence of sneak currents.

Figure 3c shows the change in $V_M$ as a function of time with and without the $R_{Unsel}$ components, for programming eight conductance states. When the sneak currents are present, it lowers the $V_M$ and the conductance change slows. As a result, the $t_{PGM}$

---

**Figure 3.** The effect of sneak currents on SAP and i-SAPP method. a) A schematic crossbar array circuit configuration composed of an $(n \times m)$ memristor array and an $(n \times l)$ resistor array where $n$ is the number of rows, and $m$ and $l$ are the number of columns of the memristor and resistor array, respectively. In the crossbar array, two leakage current paths are shown. b) A self-limited saturation behavior of the memristor for programming various states as a function of the programming time considering sneak currents (lines) and neglecting them (dashed lines). c) The memristor voltage ($V_M$) over time for different target states with sneak currents (lines) and without sneak currents (dashed lines). The programming time required for programming eight cells to the different states as functions of the number of word and bit lines by the d) SAP method and e) i-SAPP methods. f) The programing time comparison by SAP and i-SAPP methods with and without considering the sneak currents.
grows while programming the same target state. Figure 3d,e show the \( t_{\text{PGM}} \) while programming eight cells to different states using the SAP and i-SAPP operations, as functions of the number of words and bit lines of the memristor array. In both the SAP and i-SAPP cases, the increase in the number of bit lines has a greater effect on the increase in \( t_{\text{PGM}} \) than the increase in the number of word lines, because the primary sneak current paths originate from the unselected bit lines biased to \( 1/2 \) \( V \). Figure 3f shows the influence of the sneak currents on the total \( t_{\text{PGM}} \) using the SAP and i-SAPP operations, where the sizes of the memristor and resistor arrays are \( 8 \times 8 \) and \( 8 \times 8 \), respectively. After considering the sneak currents, the programming times for both SAP and i-SAPP increased by 25% and 13%, respectively, which suggests that the i-SAPP operation has more tolerance against the sneak current issue. This is because, during i-SAPP operation, multiple cells are selected together so that the number of sneak paths from unselected cells can be reduced. Considering such sneak current tolerance of the i-SAPP operation, the advantage of the i-SAPP operation is effective regardless of the array size.

5. Application of i-SAPP to a CNN

The i-SAPP method can reduce the weight update time, so it allows faster on-chip training of the neural network. During the on-chip training process, the weight values \((W)\) of the neural network should be updated for every training iteration. The weight update can be performed in two ways: by reprogramming (initialization followed by programming) and by differential updating. Figure 4a schematically shows the reprogramming and differential updating methods for modifying the weight values. In reprogramming, the memristor cell is initialized to the initial state by reset switching, and then reprogrammed to the target state. In differential updating, the target state is attained directly from the previous state. If the weight differential is a positive value (i.e., the new weight value is higher than the old one), the i-SAPP method can be used. If the weight differential is negative, the i-SAPP cannot be used, and the reprogramming method should be used. Considering the one-directional weight update capability of the differential updating method, a dual memristor system was adopted, where two memristors, representing the positive and negative values of the weight, respectively, are combined to define one synapse.\(^{[12–14]}\) Figure 4b shows a conductance diamond map where the \( G^+ \) and \( G^- \) axes refer to the conductance of the two coupled memristors. Here, the vertical component of the sum of two vectors represents the weight value of one synapse. The blue and orange arrows represent the positive and negative weight update vectors, respectively. The figure also provides an example of weight value changes during neural network training. For the positive (or negative) weight update, the \( G^+ \) (or \( G^- \)) memristor is potentiated. In this method, repeating the weight update process will eventually lead to the highest conductance state \((G_{\text{max}})\) of the memristors. Then, no further update is possible, and the effective conductance range decreases, which will deteriorate the neural network performance. Accordingly, in the dual memristor system, reprogramming of both cells should be performed regularly as per a fixed number of iterations during training. We define this as the reprogramming interval.

Figure 4. Memristive CNN simulation for MNIST data recognition. a) A schematic illustration of reprogramming and differential updating methods for tuning the memristor conductance state. Reprogramming method includes an initialization step (reset step) and a programming step. b) Conductance diamond map representing a weight value of synapse by two coupled memristors \((G^+ \) and \( G^-)\). The \( G^+ \) and \( G^- \) axes refer to the conductance of two coupled memristors, and the y-axis vector components of the sum of \( G^+ \) and \( G^- \) vectors represent the weight value. c) The hierarchy of the CNN. Feature extraction (left panel) sequences are executed by software-based digital computing, and classification sequences are done by memristive hardware-based simulation. d) The flow chart of the CNN learning process.
The i-SAPP programming method was applied to a software-based neural network emulator, and the reduction in programming speed was evaluated. In this study, we took a CNN with a transfer learning method as a demonstration platform, and the training and inference test was performed using the MNIST dataset. We also performed the same simulation with a multilayer perceptron (MLP); those results can be found in Figure S5, Supporting Information.

In the CNN, the convoluted data size was greater than the original data size. Thus, its array requires more bit lines than the MLP, making the parallel programming method more beneficial. Figure 4c shows the hierarchy of the CNN. In the CNN, the input data with a 28 × 28 size were delivered to a pretrained convolutional layer composed of 20 kernels of 9 × 9 size with a valid convolution and one stride. The output data from the convolutional layer, called a feature map, have a 20 × 20 × 20 volume, and delivered max pooling layers of 2 × 2 size for downsampling. Then, a 2000 × 1 sized downsampled feature map is delivered to a fully connected layer in which a vector matrix multiplication (VMM) is executed in a 2000 × 10 memristor crossbar array. Figure 4d shows a flow chart of the CNN training process. Pretrained convolution, pooling, and activation function were executed by digital computing, whereas the VMM was executed in the hardware-mimicking memristor crossbar.

Figure 5a shows MNIST data recognition accuracy as a function of the number of conductance states of the memristor. At eight conductance levels, the accuracy was 98.25%, which was slightly lower than the ideal value, 98.74%, due to the 5% intrinsic programming error rate. Figure 5b shows the training time for the SAP and i-SAPP methods as a function of the reprogramming interval. During the simulation, the programming time considered the sneak currents, as shown in Figure 3. Note that the presence of the sneak current does not degrade the programming accuracy, but it increases the programming time. Therefore, reducing the programming time is crucial at the given sneak currents. Overall, the total programming time was significantly reduced by adopting the i-SAPP method by a factor of 1/130 to 1/20 depending on the reprogramming interval. Figure 5c shows the recognition accuracy as a function of the reprogramming interval. As the reprogram interval becomes smaller, the total training time is slower, but the recognition accuracy is higher. The decrease in the accuracy is due to the increasing number of the \(G_{\text{max}}\) cells, which do not allow further weight update (Figure S4, Supporting Information).

Lastly, the influence of cycle-to-cycle or device-to-device variability of the memristive neural network on the i-SAPP operation is investigated to confirm the practical feasibility of the method. Figure 6a shows an example of the cycle-to-cycle variability of the device during the programming. Although the conductance increases linearly in average, a subtle but obvious conductance fluctuation is observed. The device possesses about 5% of the coefficient of variation in conductance (Fig. S1b, Supporting Information). Figure 6b shows a simulated programming accuracy considering the variability. It shows that the cycle-to-cycle variability does not harm the programming accuracy by the i-SAPP operation. It is because the saturated programming conductance (\(S_{\text{PGM}}\)) is determined by the \(R_x\) irrespective of the device variation. Accordingly, the neural network performances are almost constant independent of the cycle-to-cycle variability. Figure 6c shows the programming characteristic representing the device-to-device variability. Unlike the cycle-to-cycle variability, as shown in Figure 6e, the programming error rate of the i-SAPP operation increases as the device-to-device variability increases. It is because of the nonsaturation of the conductance during the i-SAPP operation, which results in inaccurate programing. Interestingly, despite the high error rate at high device-to-device variability, the neural network performances shown in Figure 6f are not deteriorated. It infers that the programming error can be self-adjusted during the on-chip training process of the neural network, which is the well-known error-tolerant characteristic of the neural network.

6. Conclusion

In this study, we proposed a novel SAP method working in parallel (SAPP) for fast weight updating of a high-density memristive neural network. We also demonstrated the i-SAPP method, which applied an incremental programming algorithm to the SAPP, to maximize the efficiency of the SAP method. The estimated programming time showed that the SAPP method can perform analog data programming in less time, regardless of the array size. This makes it suitable for programming larger arrays. We applied the i-SAPP method to the CNN and the MLP to estimate programming efficiency and concluded that the total

Figure 5. Memristive CNN simulation result. a) MNIST data recognition rate as a function of the number of conductance states of the memristor. b) The comparison of training time of SAP and i-SAPP method as a function of the reprogramming interval. c) MNIST recognition rate as a function of the reprogramming interval.
programming time, and the energy consumed in the training process, could be reduced by $1/130$ – $1/20$, and $1/60$ – $1/6$, respectively, depending on the reprogramming interval.

In closing, there are some possible questionnaires associated with the practical issues of the device operation. First, the non-ideality of the device such as a wire resistance may influence the programming accuracy during the parallel programming operation, considering the required large array size for the deep neural network. In our device, due to the low current operation capability, the wire resistance can be negligible (Figure S6, Supporting Information). However, for other devices where the wire resistance is not negligible, the parallel programming method may be revised, which can be the next issue of the technology. In addition, the cell-to-cell or device-to-device variability of the device may affect the ideal i-SAPP operation. Our simulation confirmed that those variabilities may affect the programming accuracy of the device. However, they do not harm the performance of the on-chip trained neural network because the weight update process can compensate for the possible errors in previous weight values. In conclusion, there are no drawbacks due to the adoption of the i-SAPP in the memristive neural network.

Supporting Information
Supporting Information is available from the Wiley Online Library or from the author.

Acknowledgements
This research was supported by the Ministry of Trade, Industry & Energy (MOTIE; grant nos. 20003655 and 20003789) and Korea Semiconductor Research Consortium (KSRC) support program for the development of the future semiconductor device.

Conflict of Interest
The authors declare no conflict of interest.

Keywords
analog programming, stateful logic, neuromorphic devices, self-limited switching, memristors

Received: January 23, 2020
Revised: March 17, 2020
Published online: April 24, 2020

[1] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, D. B. Strukov, Nature 2015, 521, 61.
[2] D. Querlioz, O. Bichler, P. Dollfus, C. Gamrat, IEEE Trans. Nanotechnol. 2013, 12, 288.
[3] C. Mackin, H. Tsai, S. Ambrogio, P. Narayanan, A. Chen, G. W. Burr, Adv. Electron. Mater. 2019, 5, 1900026.
[4] P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, H. S. P. Wong, H. Qian, Nat. Commun. 2017, 8, 15199.

[5] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, J. P. Strachan, Adv. Mater. 2018, 30, 1705914.

[6] C. Li, Z. Wang, M. Rao, D. Belkin, W. Song, H. Jiang, P. Yan, S. Williams, J. J. Yang, Q. Xia, Nat. Mach. Intell. 2018, 1, 49.

[7] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, Nat. Commun. 2018, 9, 2385.

[8] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, J. Yang, G. B. Wurth, Nature 2019, 558, 60.

[9] Z. Wang, S. Joshi, S. Savel’ev, W. Song, R. Midya, Y. Li, M. Rao, P. Yan, S. Asapu, Y. Zhuo, H. Jiang, P. Lin, C. Li, J. H. Yoon, N. K. Upadhyay, J. Zhang, M. Hu, J. P. Strachan, M. Barnell, Q. Wu, H. Wu, R. S. Williams, Q. Xia, J. J. Yang, Nat. Electron. 2018, 1, 137.

[10] Z. Wang, C. Li, W. Song, M. Rao, D. Belkin, Y. Li, P. Yan, H. Jiang, P. Lin, M. Hu, J. P. Strachan, N. Ge, M. Barnell, Q. Wu, A. G. Barto, Q. Qiu, R. S. Williams, Q. Xia, J. J. Yang, Nat. Electron. 2019, 2, 115.

[11] Z. Wang, C. Li, P. Lin, M. Rao, Y. Nie, W. Song, Q. Qiu, Y. Li, P. Yan, J. P. Strachan, N. Ge, N. Mcdonald, Q. Wu, M. Hu, H. Wu, R. S. Williams, Q. Xia, J. J. Yang, Nat. Mach. Intell. 2019, 1, 434.

[12] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, W. Lu, Nano Lett. 2010, 10, 1297.

[13] S. Kim, C. Du, P. Sheridan, W. Ma, S. Choi, W. D. Lu, Nano Lett. 2015, 15, 2203.

[14] Z. Wang, S. Joshi, S. E. Savel’ev, H. Jiang, R. Midya, P. Lin, M. Hu, N. Ge, J. P. Strachan, Z. Li, Q. Wu, M. Barnell, G. L. Li, H. L. Xin, R. S. Williams, Q. Xia, J. J. Yang, Nat. Mater. 2017, 16, 101.

[15] K. M. Kim, J. Zhang, C. Graves, J. J. Yang, B. Choi, C. S. Hwang, Z. Li, R. S. Williams, Nano Lett. 2016, 16, 6724.

[16] J. H. Yoon, J. S. Song, I. H. Yoo, J. Y. Seok, K. J. Yoon, D. E. Kwon, T. H. Park, C. S. Hwang, Adv. Funct. Mater. 2014, 24, 5086.

[17] J. H. Yoon, K. M. Kim, S. J. Song, J. Y. Seok, K. J. Yoon, D. E. Kwon, T. H. Park, Y. J. Kwon, X. Shao, C. S. Hwang, Adv. Mater. 2015, 27, 3811.

[18] J. H. Yoon, S. Yoo, S. J. Song, K. J. Yoon, D. E. Kwon, Y. J. Kwon, T. H. Park, H. J. Kim, X. L. Shao, Y. Kim, C. S. Hwang, ACS Appl. Mater. Interfaces 2016, 8, 18215.

[19] J. H. Yoon, D. E. Kwon, T. H. Park, X. L. Shao, C. S. Hwang, Nanoscale 2017, 9, 11920.

[20] C. C. Hsieh, A. Roy, Y. F. Chang, D. Shahrijerdi, S. K. Banerjee, Appl. Phys. Lett. 2016, 109, 223501.

[21] P. Bousoulas, I. Michelakaki, E. Skotadis, M. Tsigkourakos, D. Tsoukalas, IEEE Trans. Electron Devices 2017, 64, 3151.

[22] D. Sakellaropoulos, P. Bousoulas, D. Tsoukalas, J. Appl. Phys. 2019, 126, 044501.

[23] G. S. Kim, H. Song, Y. K. Lee, J. H. Kim, T. Park, H. J. Kim, K. M. Kim, C. S. Hwang, ACS Appl. Mater. Interfaces 2019, 11, 47063.

[24] H. Song, Y. S. Kim, J. Park, K. M. Kim, Adv. Electron. Mater. 2019, 5, 1800740.

[25] K. M. Kim, J. J. Yang, E. Merced, C. Graves, S. Lam, N. Davila, M. Hu, N. Ge, Z. Li, R. S. Williams, C. S. Hwang, Adv. Electron. Mater. 2015, 1, 1500095.

[26] K. M. Kim, S. R. Lee, S. Kim, M. Chang, C. S. Hwang, Adv. Funct. Mater. 2015, 25, 1527.

[27] K. M. Kim, J. J. Yang, J. P. Strachan, E. M. Grafals, N. Ge, N. D. Melendez, Z. Li, R. S. Williams, Sci. Rep. 2016, 6, 20085.

[28] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, R. S. Williams, J. Yang, H. P. Labs, in 53nd ACM/EDAC/IEEE Design Automation Conf., IEEE, Austin, TX, USA, 2016, pp. 1–6.

[29] E. J. Merced-Grafals, N. Davila, N. Ge, R. S. Williams, J. P. Strachan, Nanotechnology 2016, 27, 365202.

[30] F. Alibart, L. Gao, B. D. Hoskins, D. B. Strukov, Nanotechnology 2012, 23, 075201.

[31] P. Y. Chen, L. Gao, S. Yu, IEEE Trans. Multi-Scale Comput. Syst. 2016, 2, 257.

[32] M. Suri, O. Bichler, D. Querlloz, O. Cueto, L. Pernioli, V. Sousa, D. Vuillaume, C. Gamrat, B. DeSalvo, in Int. Electron Devices Meeting, IEEE, Washington, DC, 2011, pp. 4.4.1–4.4.4.

[33] G. W. Burr, R. M. Shelby, C. Di Nolfo, J. W. Jang, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. Kurdi, H. Hwang, in IEEE Int. Electron Devices Meeting, IEEE, San Francisco, CA, 2014, pp. 29.5.1–29.5.4.

[34] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. Kurdi, H. Hwang, IEEE Trans. Electron Devices 2015, 62, 3498.

[35] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Proc. IEEE 1998, 86, 2278.

[36] J. Donahue, Y. Jia, O. Vinyals, J. Hoffmann, N. Zhang, E. Tzeng, T. Darrell, 31st Int. Conf. Machine Learning, Vol. 32, JMLR, Beijing, China 2014, pp. 647–655.

[37] Y. LeCun, C. Cortes, C. Burges, THE MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/ (accessed: April 2020).