An Energy Efficient In-Memory Computing Machine Learning Classifier Scheme

Shixiong Jiang, Sheena Ratnam Priya, Naveena Elango, James Clay, Ramalingam Sridhar
Department of Computer Science and Engineering,
University at Buffalo, The State University of New York
Email:{shixiong, naveenae, sheenara, jnclay, rsridhar}@buffalo.edu

***Abstract*— Large-scale machine learning (ML) algorithms require extensive memory interactions. Managing or preventing data movement can significantly increase the speed and efficiency of many ML tasks. Towards this end, we devise an energy efficient in-memory computing kernel for a ML linear classifier and a prototype is designed. Compared with another in-memory computing kernel for ML applications [1], we achieve a power savings of over 6.4 times than a conventional discrete system while improving reliability by 54.67%. We employ a split-data-aware technique to manage process, voltage and temperature variations. We utilize a trimodal architecture with hierarchical tree structure to further decrease power consumption. Our scheme provides a fast, energy efficient, and competitively accurate binary classification kernel.**

Keywords — Machine Learning, Classifier, In-memory computing, Low power, Hybrid, Trimodal

I. INTRODUCTION

In-memory computing (IMC) promises a natural solution to memory bottlenecks, such as those derived from ML task prediction [1]. In this paper, we develop a low power, large scale in-memory computing scheme for linear classification. Linear classifiers provide accuracy on par with deep neural networks when datasets contain reasonable levels of linear separability, and they also decrease training time [2] [3]. In a conventional von Neumann architecture, the CPU orchestrates data movement and operates on it; we note that ML classification-based tasks require large amounts of data movement to fuel the training process [4]. Consequently, an I/O bottleneck forms due to the classification process, which then increases energy consumption [5] [6]. These challenges illustrate the difficulty ML algorithms face in the mobile environment: as memory-bound tasks, they perform inefficiently due to their von Neumann execution environment, while also worsening battery drain.

IMC provides power improvements and latency reductions while offering an ML training environment that improves both speed and energy efficiency. IMC designs exist that reduce the power consumption and improve memory-bound task performance. Dong 2018 [7] implemented a novel SRAM structure for embedded searching and in-memory-computing applications, thus reducing power consumption without increasing the area overhead. A sparsely distributed memory scheme presented by Kang 2016 [8] employs a hierarchical binary decision (HBD) to reduce memory access delay and energy requirements. Kang 2016 [9] present a deep in-memory architecture (DIMA) that simultaneously accesses multiple rows of a standard 6T SRAM array per precharge. Shen 2015 [1] and Gao 2015 [6] present an in-memory weak/strong design that computes linear classifiers. By combining results of multiple weak classifiers, they create a strong classifier. Despite various energy and latency reduction techniques, power consumption remains a persistent constraint for the circuit designer. Each of the above research studies identified excessive power consumption as a key challenge in embedded ML devices. Considering the vital importance of accuracy for classifiers, and the necessity of low power design for portable devices, researching power efficient ML hardware and algorithms provides a critical challenge and opportunity. Thus, we conduct research in high speed, low power, and reliable linear classifier designs for portable ML applications.

Section II provides the details of the hybrid split-data-aware architecture, the trimodal architecture and the hierarchical tree structure employed in our design. Section III describes the simulation results of our proposed approach. Finally, we discuss the power-performance-reliability tradeoffs of the proposed architecture in Section IV.

II. ARCHITECTURE

Of the many algorithms ML researchers apply to classification problems, linear regression continues to play a prominent role. Linear regression methods perform admirably and their success hinges on the data model's linear separability [9]. When the target dataset exhibits linear separability, classification through linear regression provides an efficient alternative to costlier deep neural networks. However, these linear classifiers may also require input data normalization before use. In cases where the data has poor linear separability, projecting the data into a higher dimensional space improves the linear separability of the points at the expense of calculating the projection. Such projections are used to allow a linear classifier to work with poorly separable low-dimensional data. In our experimental results section, we discuss subsampling and show that our linear-regression successfully predicts handwritten digits from the MNIST database with a likelihood of approximately 90%. Comparing various linear classifiers, such as linear regression and logistic regression, the implementation of linear regression generally requires fewer electrical or silicon elements, as projections into higher dimensional space may increase storage requirements and force more operations to occur before the data reaches a usable state.

2380-6923/19/$31.00 ©2019 IEEE
DOI 10.1109/VLSID.2019.00046

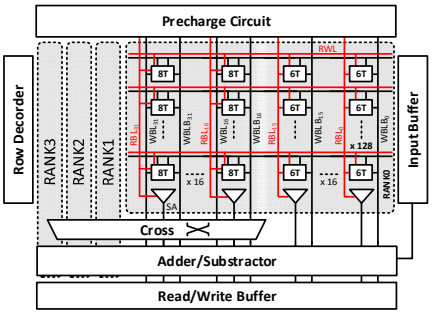


Fig. 1. In memory linear classifier

As per the standard definition of linear regression, we first determine the coefficients of a linear equation maximizing a goodness of fit score. We calculate the dot product of the coefficients with the input to arrive at a floating-point value y. We write y as a linear combination of real-valued weights, with each weight corresponding with an image pixel. We refer to these weights as pixel-weights, defined mathematically as $w_j \in \mathbb{R}$, with a given input pixel $x_j \in \mathbb{R}$, and an intercept $b \in \mathbb{R}$

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_{j=0}^{n-1} w_j x_j\right) = \sum_{j=0}^{n-1} w_j x_j + b \qquad (1)$$

The weight vector $\vec{w}$ is learned from a training set. In our in-memory classifier scheme, each row of the SRAM array represents a weight w_j. By default, each weight w_j is represented by 32 bits. An input buffer with 81 rows is designed to store the pixel values. We utilize the MNIST database of handwritten digits as our input dataset.

The linear classifier operates on a collection of pixels, e.g. an image. We formally use the binary field to define the collection of bits that form the pixel. We define the linear classifier as calculating the sum of the products $w_j\ x_{j,k}$, where k represents the index of the input bits. For an 8-bit gray scale image, the algorithm uses 8×81 multiplications per input if the multiplier operates at the bit level or 81 multiplications at the byte level. However, we perform the dot product using only $n = 8$ vector products by operating in the column space of the image matrix: we treat the 81 pixels as rows, each with 8 columns. As shown in Equation (2), for each value of k, each 81-bit column is multiplied by an 81-bit weight vector ($sum_k = w_{[0:81)} \cdot x_{[0:81),k}$). The summation of all bitwise products from each pixel, sum_k, is subject to a k-bit shift before accumulating into the buffer. Apart from the bit shifts and the dot products between w and x, only k multiplications are needed.

$$\begin{aligned} &for\ (k = 0; k < 8; ++k)\ \{ \\ &\quad sum_k = w_{[0:81)} \cdot x_{[0:81),k} \\ &\quad sum\ +=\ sum_k \times 2^k \\ &\} \\ &sum\ +=\ b \times 1 \end{aligned} \qquad (2)$$

As shown in Fig. 1, $x_{j,k}$ controls the value in each Read Word Line (RWL). When $x_{j,k} = 0$, accumulation is not performed. When $x_{j,k} = 1$, RBL is precharged to 1. Our scheme works as a conventional SRAM in READ mode and the value of the weight vector w_j is passed into the adder as $x_{j,k} = 1$. The bias b represents the intercept value. This is multiplied by the value 1 stored in row 82 and added to the final sum_k. We draw attention to the detail that calculating the vector product uses an adder outside the memory. However, calculating the vector product occurs entirely in-memory. Making the summation an in-memory operation without the read/write buffer scheme could be achieved via a carefully designed signal amplifier and a modification to the word-line read scheme. We have already begun investigating this to extend our current work.

From testing our scheme in MATLAB and Python, we have observed that classification accuracy and image resolution have a direct correlation. We obtained an accuracy of 92.95% using the 28 × 28 grayscale images from the MNIST database. By down sampling the image to a resolution of 9 × 9, the accuracy becomes 89.89% and down sampling the grey scale to a 1-bit black and white representation yields an accuracy of 84.83%. For circuit level simulation, we choose to down-sample to a resolution of 9 × 9 which has a fair trade-off between accuracy and complexity. Each pixel, given through WL, can be either 0 or 1, representing black or white respectively.

A. Hybrid split-data-aware (SDA) architecture to mitigate the effects of process, voltage and temperature variation.

Traditionally, one stores neural network weights in an array of conventional 6T SRAM cells. However, by carefully analyzing the effects of PVT variations on the stability of SRAM cells, including Negative-Bias Temperature Instability(NBTI) aging, we chose a 6T/8T hybrid SRAM array with better stability. Referring to related work in the register files domain [10], we apply the split-data-aware architecture with similar W/L ratios in our implementation to enhance reliability.

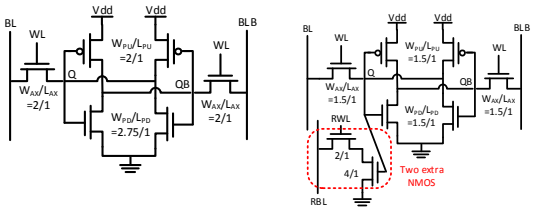


Fig. 2. Schematic of Conventional 6T SRAM (a) and 8T SRAM (b)

In a conventional SRAM cell, technology scaling seriously impacts a cell's lifespan and operational capability. Considering process variations, fast-NMOS slow-PMOS (FS) and slow-NMOS fast-PMOS (SF) are the worst process corners for read and write operations [11]. In our in-memory linear classifier scheme, we perform the read operation more frequently than the write operation. Thus, we adopt the Read Static Noise Margin (RSNM) as the reliability metric for this design.

$$\Delta V_{th} = K_v \cdot \beta^{0.25} \cdot T^{0.25} \cdot \left[\frac{1-(1-\sqrt{\frac{\eta(1-\beta)}{n}})^{2n}}{1-(1-\sqrt{\frac{\eta(1-\beta)}{n}})^{2}}\right]^{0.5} + \delta_v \qquad (3)$$

In addition to the process corners, NBTI effect is another significant factor which may lead to large-scale memory failures [12]. In a conventional 6T SRAM cell as shown in Fig. 2(a), when the pull-up PMOS transistor (P_L or P_R) is negatively biased, interface traps are generated. Consequently, normal

operation of the 6T SRAM array results in a small, but cumulative increase in the voltage threshold of the PMOS transistors. By using the Predictive Technology Model, we calculate the shift in the V_{th} value. When the Zero Bias Probabilities (ZBP) is 50%, the RSNM degradation in the 8T cell is negligible against the 15% degradation for the traditional 6T cell. The long term NBTI effects can be calculated from Equation (3) by using the Predictive Technology Model [13], where K_v and η are constants, β is the duty cycle, T is the clock period and n represents the number of cycles of stress and recovery [14]. Fig. 3 shows the RSNM comparison between 6T and 8T SRAM cells. Compared with the conventional 6T structure, the reliability of the 8T SRAM cell improves significantly.

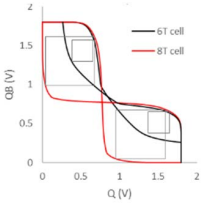


Fig. 3. SNM Buttefly Curve for 6T and 8T SRAM cell

In the standard IEEE 754 single (32-bit) precision, the importance of each bit varies. There are three components in this format: sign bit (1 bit), exponent (8 bits) and mantissa (23 bits). In order of precedence, errors in the sign bit and exponent width will affect the accuracy to a greater extent than the errors in the LSB part of the mantissa. This conception is illustrated in Fig. 4. The value in the example is 137.890625 which is represented in binary as $1.000100111101 \times 2^7$. Fig. 4 (b) and (c) show a read failure in bit 15 versus bit 31. Errors in bit 31 have greater impact on accuracy than errors in bit 15. Hence ensuring the reliability of the sign bit, exponent and MSBs of the mantissa are more significant than the LSBs of the mantissa.

Fig. 4. IEEE754 Single precision 32-bit (a) no fail
(b) fail in bit 15 (c) fail in bit 31

To achieve a fair trade-off between reliability of SRAM cells and area, we design a hybrid split-data-aware architecture as shown in Fig. 1. As we use a 32-bit floating point format to represent weights in our implementation, we ensure the reliability of the first 16 most significant bits. The sign bit, exponent and the first 7 bits of the mantissa are stored in the more reliable 8T SRAM cells (high stability), while the remaining bits of the mantissa are stored in 6T SRAM cells (low area and low power consumption). Our design operates on two accuracy modes: high accuracy mode, where Rank 0 and Rank1 are both ON and power efficient mode, where Rank 0 is ON and Rank 1 is DEAD. In the high accuracy mode, our scheme switches to 64-bit double precision format by turning ON Rank 1 and the cross circuit. The cross circuit as shown in Fig. 6, is added to ensure the sign bit, exponent and MSBs of the mantissa are always stored in 8T SRAM cells.

The hybrid split-data-aware scheme is efficient even when the weights are in fixed point format. A fixed-point number contains two parts, the integer and the fractional segments. The more significant bits (integer) are stored in 8T SRAM cells and less significant bits (fraction) are stored in 6T SRAM cells. The range of weights stored in the SRAM array depend greatly on the target application. We tested the design using the MNIST handwritten digit dataset. After training, the weights obtained are in the range 10^{-4} *to* 10^{9}, which are stored in Rank 0 and Rank 1 (64 bits) of the SRAM array. If a given application requires more bits to represent the trained weights, the design can be extended to accommodate Rank 2 and Rank 3, as shown in Fig. 1.

For efficient integration of 6T and 8T SRAM cells, we adopt the split-word line scheme from [11]. The schematic of a 32-bit hybrid-cell with one reading port and one writing port (1R1W) is shown in Fig. 5. An 8T SRAM cell has two separate word lines: RWL and WWL, for READ and WRITE operations, while a 6T SRAM cell has only one word line. The area overhead of using a hybrid 6T-8T array as opposed to a full 6T array is ~12%.

During the single-ended READ operation, Write Word Line (WWL) is disabled and Read Word Line (RWL) is enabled. During the WRITE operation, both WWL and RWL are enabled to overwrite the values in Q and QB.

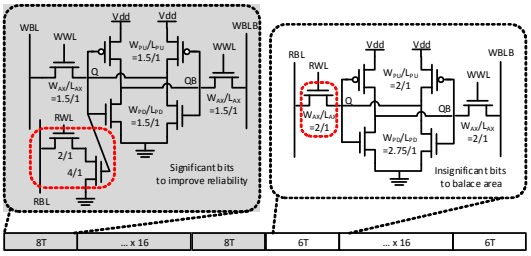


Fig. 5. Schematic of hybrid-cell

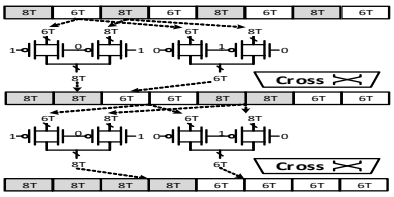
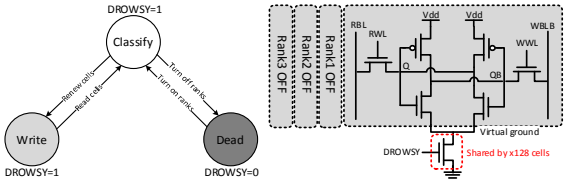


Fig. 6. Transmission gate cross circuit

B. *Trimodal architecture*

Our design has three different modes of operation as shown in Fig. 7: WRITE mode, where the SRAM array overwrites the existing values in Q and QB, CLASSIFY mode, where the SRAM array performs the binary classification using a regular READ operation and DEAD mode wherein the bit lines, word lines, precharge and discharge path are disconnected when a particular RANK is OFF. All SRAM cells in a single row of the 128x128 array share a drowsy transistor, which is enabled during the WRITE and CLASSIFY operations. The drowsy

transistor enables the discharge to *Gnd* thus resulting in 33.52% leakage current reduction with almost zero area overhead.



(a) (b)

Fig. 7. Trimodal architecture. (a) three modes (b) drowsy transistor

Depending on the target application, our scheme can switch among 32, 64 and 128-bit accuracy modes. In the 32-bit mode, only the Rank 0 is ON while all other ranks are in DEAD mode. Since the frequency of operation in CLASSIFY mode is much higher than that of the WRITE mode, the input data buffer is designed close to Rank 0 which is always ON**.**

C. Hierarchical Tree Structure

Zhang 2016 [15] present a scheme in which all 45 strong binary classifiers are activated simultaneously providing high performance classification, but this also results in high power consumption. To address this issue, we have designed an energy efficient hierarchical tree structure. There are three select signals to enable any specific classifier in a group. The select signal, en_group, selects a group in chronological order from group 0 through group 3. The classifier enable signal, en_num[0-9], enables a classifier in the desired group. We use a ten-bit status register to record the current possible labels for each group. In the beginning, we set all bits in the status register to 1, indicating each label as a possible candidate. After the classifications in each group, the possible labels reduce or remain the same, indicated by a change of the corresponding binary value in the status register. The classification performed in the final stage (group 3) results in a single labeling out of the 10 original labels. Fig. 8 illustrates an example hierarchical tree structure. All the bits in the status register are set to 1. The first group (group 0) is selected and en_num2, en_num3, en_num6 and en_num7 are loaded from the status register whose values are set to 1. After all the classifications in group 0 are completed, the status register values in bit positions 2, 3, 6 and 7 will be updated based on the results from classifier 2/3 and 6/7. If the classifier predicts the values 3 and 7, the status register values in bit positions 2 and 6 will be updated to 0. Now the status register values and select signal values are passed to group 1. Since en_num2 and en_num6 have been updated to value 0 it will disable the classifiers 2/4 and 5/6. Likewise, all the redundant classifiers are disabled in group 2 and group 3 depending on the values in the status register updated during each stage. In the final stage, after processing through all three groups, the status register will hold the value '1' in only one of the bit locations which denotes the final predicted value.

We test our design with down-sampled 9 × 9 grayscale images (each pixel is an 8-bit binary representation) from the MNIST database and obtained 89.79% prediction accuracy using the hierarchical tree structure. We also tested with a down-sampled 9 × 9 black and white image set (each pixel is a 1-bit binary representation), we achieved a prediction accuracy of 84.81%. We achieve similar prediction accuracy as obtained in [15], but also provide an 80% power savings.

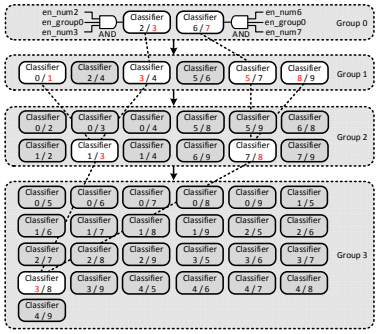


Fig. 8. Schematic of hierarchical tree structure

III. SIMULATION RESULTS

We implemented and test the design in the Cadence Virtuoso environment. We created a 128 × 128 SRAM array in a 1.8V TSMC 180nm CMOS process. We expect our scheme to scale well into lower technology nodes. We perform training with images from the handwritten number MNIST database and the weights are computed using Python and MATLAB.

A. Area Overhead

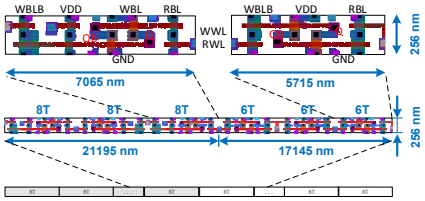
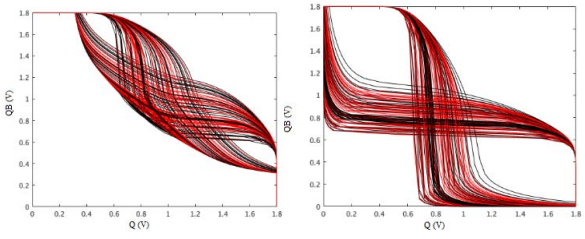


Fig. 9. Hybrid SDA SRAM cell layout

Fig. 9 shows the layout of the hybrid split-data-aware architecture. Compared with a complete 6T SRAM array, it reduces the overhead by ~12%

B. Static noise margin analysis



(a) (b)

Fig. 10. SNM considering NBTI effect (a) 6T SRAM (b) 8T SRAM

The effects of PBTI become more significant beyond 40nm process [16] and hence we analyze the variations in SNM caused by NBTI effects. Utilizing the calculated V_{th} shift due to NBTI effect as obtained in [17], Monte Carlo simulations are done for RSNM, as shown in Fig. 10. Compared to 6T SRAM cells, 8T SRAM cells have higher reliability and better resilience to temperature variations. As temperature increases, the RSNM of 6T SRAM cells decrease at a higher rate than 8T SRAM cells.

The SNM curves obtained by varying the temperature from -100°C to 100°C are shown in Fig. 11.

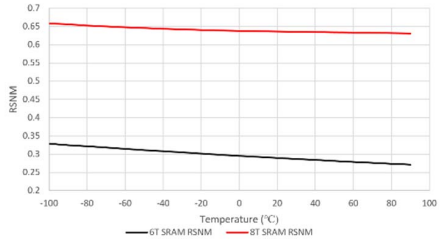


Fig. 11. SNM reduction considering temperature variation

C. Power Consumption and leakage current

Our design has an accuracy equivalent to the conventional von Neumann (CVN) architecture, while achieving ten-way classification at an energy requirement of 22.789 nJ per decision. The operating frequency is 62.5 MHz and we achieve upto 6.4 times reduction in energy consumption compared to CVN [15]. Significant power savings comes from the different techniques we have implemented. Energy consumption due to data access is limited by employing in-memory computing. Based on intense dataset analysis, the multi-rank architectures and hierarchical tree structure are introduced which further reduce power consumption.

Fig. 12 shows the power consumption in different cases. When all 128 bits are turned ON and when all the input image pixel values are equal to 1, the power consumption goes as high as 3.44mW. In comparison with high accuracy mode, the power consumption drops down to 1.97mW when only 32 bits are turned ON. A power saving of around 42.73% is achieved by using the multi-rank architecture. When input image pixels are equal to 0, the product $w_j\ x_j$ is always equal to 0 and hence all factors that cause unnecessary power dissipation including discharging, addition and buffer renew are avoided. This results in 90% power saving in the 32 bits mode while achieving 84.62% power saving in the 128 bits mode.

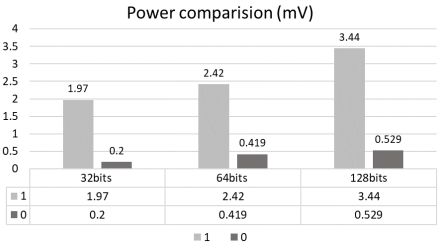
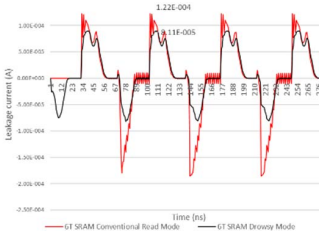
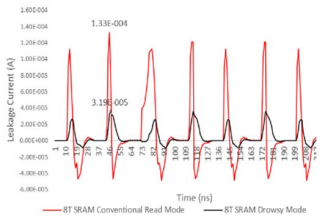


Fig. 12. Comparison of power consumption

We reduce leakage current by employing a drowsy transistor as shown in Fig. 712. While the hybrid split-data-aware architecture efficiently boosts the reliability. The effect of employing a drowsy transistor in a single 6T and a single 8T SRAM cell is shown in graphs in Fig. 13 (a) and (b). On observing the amount of current flowing through the leakage path of the SRAM cell, we conclude that 33.52% of leakage current reduction can be attained using the drowsy mode of operation.



(a)



(b)

Fig. 13. Leakage current analysis for drowsy transistor

D. Timing Diagram

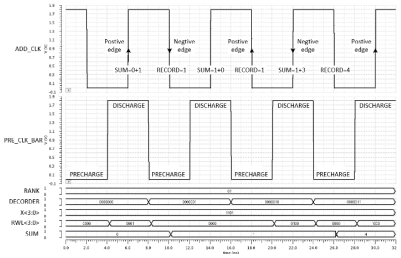


Fig. 14. Timing Diagram of 0 vs.1 classifier

Fig. shows the timing diagram for 0 vs. 1 classify operation in our proposed scheme. The first 4 bits of the down-sampled 9 × 9 image pixels with B/W representation (1-bit) are shown in this example. The weights are represented by a 64-bit integer value. Only Rank 0 and Rank 1 are turned ON in order to reduce the energy consumption. In our design, a PRE_CLK_BAR signal is used to control the bit line precharge, input image data read operation and ADD_and_MULTIPLY operation. For a B/W image, the ADDER is activated. First, the PRE_CLK_BAR is set to 0, which means the precharge circuit is ON and RBL is precharged to 1. Simultaneously, the input decoder selects the right image pixel value and the ADDER is disabled. In the second half period, PRE_CLK_BAR switched ON and the precharge operation is completed. The value in RWL is decided by the corresponding output of the decoder and the image pixel value X. The decoder selects a certain pixel of the image x_j and passes only this value to RWL_j, while all other RWLs are equal to 0. When RWL_j is 0, there is a discharge in RBL and ADD operation is disabled in order to eliminate unnecessary energy consumption. When RWL_j is 1, the SRAM array works as conventional SRAM in READ mode. The values in one row of the SRAM array are passed to ADDER. A second clock signal ADD_CLK with 1/4 period delay compared to PRE_CLK_BAR is used. At the rising edge of ADD_CLK, the

ADDER sums the values in the ALU Buffer Register and product of w_j and x_j from the SRAM array. After PERCHARGE_CLK_BAR switches back to 0, the D Latch array inside the ALU holds the values until they are read at the negative edge of ADD_CLK. The ADDER accumulates the values and stores the results in the ALU Buffer Register. After all 82 rows are read, the resulting value in the Buffer register is compared with the threshold of the particular classifier and the classification result is obtained.

IV. CONCLUSION

A novel in-memory computing linear classifier has been presented for portable ML devices. From the simulation results it can be verified that our scheme has attained 6.4 times energy savings and 54.67% improved reliability compared with the conventional von Neumann architecture [18]. The architecture we propose can be extended to include other machine learning techniques like random forest algorithm to be performed as an in memory computing scheme thus handling the I/O bottleneck faced in the modern processors.

V. ACKNOWLEDGEMENTS

This work was supported in part by NSF grant CNS-1422304

VI. REFERENCES

[1] F. Shen, C. Shen, W. Liu and H. Tao Shen, "Supervised discrete hashing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Boston, 2015.

[2] H.-Y. a. L. C.-J. Huang, "Linear and kernel classification: When to use which?," in *Proceedings of the 2016 SIAM International Conference on Data Mining*, 2016. [Online]. Available: https://www.csie.ntu.edu.tw/∼cjlin/papers/kernel-check/kcheck.pdf

[3] J. P. Cunningham and Z. Ghahramani, "Linear dimensionality reduction: Survey, insights, and generalizations," *The Journal of Machine Learning Research,* vol. 16, pp. 2859-2900, 2015.

[4] J. Steinhardt and J. Duchi, "Minimax rates for memory-bounded sparse linear regression," in Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015, pp. 1564–1587, 2015.

[5] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 10–14, 2014.

[6] M. Gao, G. Ayers and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *Parallel Architecture and Compilation (PACT), 2015 International Conference on*, pp. 113–124 2015.

[7] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, S. Miyoshi, M. Yasuda, D. Blaauw and D. Sylvester, "A 4+ 2T SRAM for Searching and In-Memory Computing With 0.3-V," *IEEE Journal of Solid-State Circuits,* vol. 53, pp. 1006-1015, 2018.

[8] M. Kang and N. R. Shanbhag, "In-Memory Computing Architectures for Sparse Distributed Memory.," *IEEE Trans. Biomed. Circuits and Systems,* vol. 10, no. 4, pp. 855-863, 2016.

[9] M. Kang, S. K. Gonugondla, A. Patil and N. R. Shanbhag, "A Multi-Functional In-Memory Inference Processor Using a Standard 6T SRAM Array," *IEEE Journal of Solid-State Circuits,* vol. 53, pp. 642-655, 2018.

[10] N. Gong, S. Jiang, J. Wang, B. Aravamudhan, K. Sekar and R. Sridhar, "Hybrid-cell register files design for improving NBTI reliability," *Microelectronics Reliability,* vol. 52, pp. 1865-1869, 2012.

[11] I. J. Chang, D. Mohapatra and K. Roy, "A priority-based 6T/8T hybrid SRAM architecture for aggressive voltage scaling in video applications," *IEEE transactions on circuits and systems for video technology,* vol. 21, pp. 101-112, 2011.

[12] S. P. Park, K. Kang and K. Roy, "Reliability implications of bias-temperature instability in digital ICs," *IEEE Design & Test of Computers,* vol. 26, no. 6, pp. 8–17, 2009.

[13] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao and S. Vrudhula, "Predictive modeling of the NBTI effect for reliable design," in *In Custom Integrated Circuits Conference*, pp 189–192, 2006.

[14] "PTM Model," [Online]. Available: http://www.eas.asu.edu/ ptm.

[15] J. Zhang, Z. Wang and N. Verma, "A machine-learning classifier implemented in a standard 6T SRAM array," in *VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on*, pp. 1–2, 2016.

[16] M. Yabuuchi and K. Kobayashi, "Circuit Characteristic Analysis Considering NBTI and PBTI-Induced Delay Degaration," in *IEEE International Meetig For Future of Electron Devices*, Kansai, pp. 1–2, 2012.

[17] W. Wang, V. Reddy, B. Yang, V. Balakrishnan, S. Krishnan and Y. Cao, "Statistical prediction of circuit aging under process variations," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pp. 13–16, 2008.

[18] J. Zhang, Z. Wang and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array.," *J. Solid-State Circuits,* vol. 52, pp. 915-924, 2017.

[19] K. Kang, H. Kufluoglu, K. Roy and M. A. Alam, "Impact of negative-bias temperature instability in nanoscale SRAM array: Modeling and analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 26, pp. 1770-1781, 2007.

[20] F. Ahmed and L. Milor, "NBTI resistant SRAM design," in *Advances in Sensors and Interfaces (IWASI), 2011 4th IEEE International Workshop on*, pp. 82 –87, 2011.

[21] L. Chang, R. K. Montoye, Y. Nakamura, K. A. Batson, R. J. Eickemeyer, R. H. Dennard, W. Haensch and D. Jamsek, "An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches," *IEEE Journal of Solid-State Circuits,* vol. 43, pp. 956-963, 2008.

[22] E. Glocker, D. Schmitt-Landsiedel and S. Drapatz, "Countermeasures against NBTI degradation on 6T-SRAM cells," *Advances in Radio Science,* vol. 9, pp. 255-261, 2011.

[23] S. K. Krishnappa and H. Mahmoodi, "Comparative BTI reliability analysis of SRAM cell designs in nano-scale CMOS technology," in *ISQED*, pp. 1–6, 2011.

[24] S. Kothawade, K. Chakraborty and S. Roy, "Analysis and mitigation of NBTI aging in register file: An end-to-end approach," in *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, pp. 1–7, 2011.

[25] J. Abella, X. Vera and A. Gonzalez, "Penelope: The NBTI-aware processor," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 85–96, 2007.

[26] S. . Chen, F. . Shen, Y. . Yang, X. . Xu and J. . Song, "Supervised hashing with adaptive discrete optimization for multimedia retrieval," *Neurocomputing,* vol. 253, no. , pp. 97-103, 2017.

[27] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks,* vol. 61, pp. 85-117, 2015.