1. Introduction

In this paper, we presented a detailed analysis of the pulse-train operation method for controlling multi-level conductive filament evolution in resistive random-access memory (ReRAM) devices. This approach addresses the challenge of achieving precise electrical manipulation of nanoscale defects, which is crucial for reliable multi-level ReRAM control [9]. We used pulse train for reset operation only as we need the maximum target resistance value to be accurate.

The research demonstrates that using pulse-train signals significantly improves the standard deviations of resistance levels, enhancing the control over the filament-rupture gap-formation process compared to single-pulse systems. It provides an in-depth analysis of enhancing pulsed Reset and SET functionality in resistive random-access memory (RRAM) devices for neuromorphic computing applications. The findings highlight the benefits of pulse-train operation in achieving more consistent multi-level switching, reducing resistance fluctuations, and improving storage capacity [15].

The research emphasizes the use of electrical pulses as a method for precise analog switching, maintaining area efficiency, and explores the optimization of programming pulses to mimic synaptic activity. By employing experiments and device simulations, the study examines the response of RRAM stacks to pulsed RESET operations and single pulse SET operation. The findings demonstrate the total write time required to write into the memory cells via number of RESET pulses and sinle SET pulse can achieve MLC functionality, making these RRAM cells viable for electronic synapses. This research provides valuable insights into the physical mechanisms driving pulsed RESET behavior and highlights the importance of optimizing pulse schemes to enhance the performance of RRAM devices in neuromorphic computing applications [9]. It also discusses the experimental setup and presents data on write time patterns, and variability improvements achieved through pulse-train operation. It will present how total write time affects different neural network models,and their intermediate layers, based on the experimental setup of 75% slow write and 25% slow write that is done for bandwidths of 32 bits, 16 bits and 64 bits accordingly. Fast write impulse will bring the resistance value close to the target and then the following impulses,that is, slow write impulse will fine tune the value slightly making it more accurate and precise. This is where deep neural network comes into play. Each of the intermediate layers contributes to get the target result at the end output. . [5]

2.Background

2.1 Resistive RAM switching mechanisms

In RRAM, High Resistance State (HRS) means “OFF” state i.e. “0” logic level and Low Resistance State(LRS) means “On” state i.e. “1” logic level. Resistive switching phenomenon is the reason behind the change of resistance values. The process of changing from HRS to LRS is enabled by SET voltage. Application of high voltage allows formation of conductive paths in the switching layers and RRAM is switched into LRS sate. This occurs due to breakdown of metal insulator metal(MIM). The voltage at which it occurs is called the forming volatage(Vf). To switch the RRAM from LRS to HRS the RESET voltage is applied(Vreset) and the process is called ‘RESET’ process. RRAM is called non-volatile because if I remove the applied voltage it will still retain their respective values. . [13]

Switching modes of RRAM:(I) unipolar switching (II) Bipolar switching. In case of unipolar switching, switching between set and reset can occur even if same polarity just change in voltage magnitude. Joule heating is understood to be the physical process that causes a conducting filament to burst during a reset operation in unipolar switching.. [6] In bipolar switching, the switching (set and reset process) of the device depends on the polarity of the applied voltage i.e., transition from HRS to LRS occurs at one polarity (either positive or negative) opposite polarity switches the RRAM back into the HRS(as shown in figure 1). Nevertheless, Joule heating still helps to speed up the movement of charged species, which is the primary cause of conductive filament disintegration in bipolar switching. To prevent permanent breakdown of the dielectric switching layer during the forming/set process in RRAM, a compliance current (Icc) is enforced. This is usually achieved with a cell selection device (transistor, diode, resistor) or a semiconductor parameter analyzer during off-chip testing.. [9]

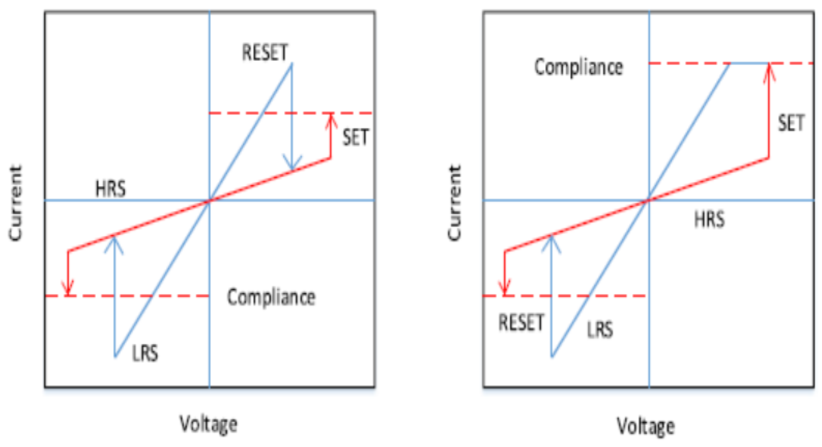


Figure 1: I-V curves for RRAM. (a) Unipolar switching and (b) bipolar switching. [9]

We have used bipolar switching mode of RRAM in our experiments.The switching mechanism in RRAM cells involves the formation of a conductive filament (CF) w ithin a dielectric, creating a nanometer-sized channel between the cell's electrodes. When the CF connects the electrodes, the cell is in a low resistance state (LRS); when the CF is disconnected, the cell is in a high resistance state (HRS). In a CBRAM memory cell, the Ag top electrode (TE) forms a conductive filament w ith t he i nert P t b ottom e lectrode (BE). . [6] When a positive voltage is applied to the Ag TE, Ag+ cations are generated and deposited into the dielectric layer. These cations are attracted to the Pt BE, where they are reduced to Ag atoms, forming a conductive bridge (LRS) in a process called 'SET'. Reversing the voltage polarity dissolves the filament, returning the device to a high resistance state (HRS) in a process called 'RESET'(as shown in figure 3).. [9]

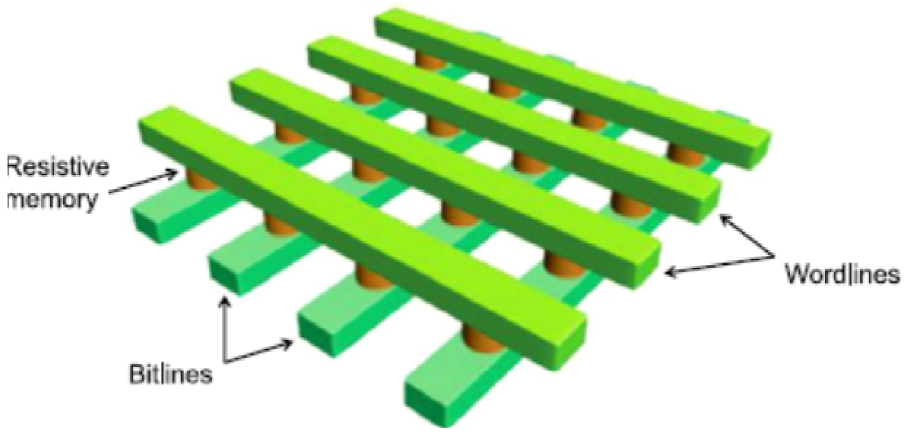


Figure 2: Crossbar architecture for resistive memories

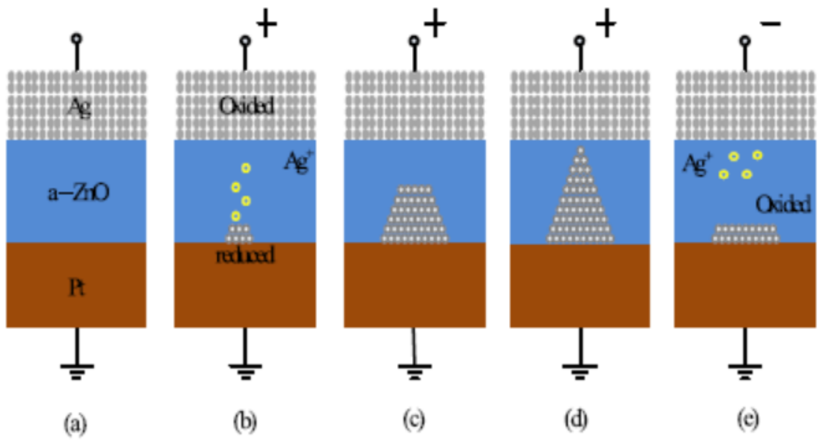


Figure 3: Schematic of the switching mechanism of conductive bridge RRAM. . [9]

Crossbar Architecture of RRAM:
Electrical pulses can alter resistance in response to a variety of physical processes. In an RRAM, resistance typically varies in response to the condition of the conductive filament (CF) inside the insulating oxide layer.
Crossbar structure in resistive memory applications:
Perpendicular conductive wordlines (rows) and bitlines (columns) are used to create the array, and a memory element is present at the junction of each row and column. By biasing the matching wordline and bitline, the memory element may be read and written to. . [10]The record low area of the single memory cell is 4F2, where F is the smallest lithographic feature size that the technology node specifies(as shown in figure 2).

2.2 Multi-level Cell

MLC RRAM (Multi-Level Cell Resistive RAM) is a type of resistive RAM that stores multiple bits per cell, similar to MLC NAND flash m emory. U nlike N AND fl ash, which

uses charge storage to represent data, RRAM uses resistance levels to store data. . [8]In MLC RRAM, multiple resistance states represent multiple bits per cell. For example, an MLC RRAM cell might use several resistance levels to represent 2 or more bits of data, enabling higher storage density. Key features of MLC RRAM include:
Higher Density: By storing more bits per cell, MLC RRAM can achieve higher storage densities compared to single-level cell (SLC) RRAM.
Non-Volatility: Like other types of RRAM, MLC RRAM retains data without power.
Endurance: RRAM generally has good endurance characteristics, but the specific endurance of MLC RRAM can vary depending on the technology and implementation.
Performance: MLC RRAM may have different performance characteristics compared to SLC RRAM, often with trade-offs between speed and density.
MLC RRAM aims to combine the advantages of high density and non-volatility while maintaining performance and endurance suitable for various applications.

2.3 In-memory DNN Processing

In-memory DNN processing refers to the execution of deep neural network (DNN) computations directly within the memory hardware, rather than transferring data back and forth between the memory and a separate processing unit (CPU/GPU). This approach leverages the concept of Processing-in-Memory (PIM). [17], aiming to overcome the limitations of traditional computing architectures by addressing power and memory bottlenecks. RRAM utilizes a crossbar structure, where memory cells at the intersections of horizontal and vertical wires change their resistance to store data. This structure supports parallel operations.RRAM cells can perform analog current-mode weighted summation, matching the multiplication-and-accumulation (MAC) operations in DNNs. The resistance value will be pushed toward the goal by the fast write impulse, and then it will be slightly adjusted by the slow write impulse to make it more exact and accurate. Deep neural networks are useful in this situation. Every intermediary layer helps the final product achieve the desired outcome.It comprises of the input layer, output layer and intermediate or hidden layers. . [2]

3. Motivation

Our research aims to demonstrate how different write processes affect the efficiency of resistance results in a neural network model. For example, when dealing with 32-bit data, altering the last 3 or 4 bits typically has minimal impact. This highlights the importance of slow write operations. In contrast, with fast write processes, both the least significant bits (LSBs) and the most significant bits (MSBs) are involved, leading to more significant changes due to the modification of MSB digits. For example: An 32-bit value can be represented as 10001010011110110011110000101101 (in binary). The maximum difference 32 bits can make is $2^{32} - 1 = 4294967295$ (from 0 to 4294967295). The range of values represented by the last 4 bits is from 0000 (0) to 1111 (15). Therefore, the maximum change that can occur due to changes in the last 4 bits is $15 - 0 = 15$. The percentage difference between the maximum difference that can be made with 32 bits and max difference due to changes in the last 4 bits for an 32-bit value is:

$$\begin{aligned} 4294967295 - 15 &= 4294967280 \\ \Rightarrow \frac{4294967280}{4294967295} &\approx 0.9999999965 \\ \times 100 &\approx 99.99999965\% \end{aligned}$$

It shows very negligible deviation from its original value,almost same, no change from the original value,i.e. does not impact the original resistance value.
Similarly, if we change the last 8 bits, the range of values represented by the last 8 bits is from 00000000 (0) to 11111111 (255). Therefore, the maximum change that can occur due to changes in the last 8 bits is $255 - 0 = 255$. The percentage difference between the maximum difference that can be made with 32 bits and max difference due to changes in the last 8 bits for an 32-bit value is:

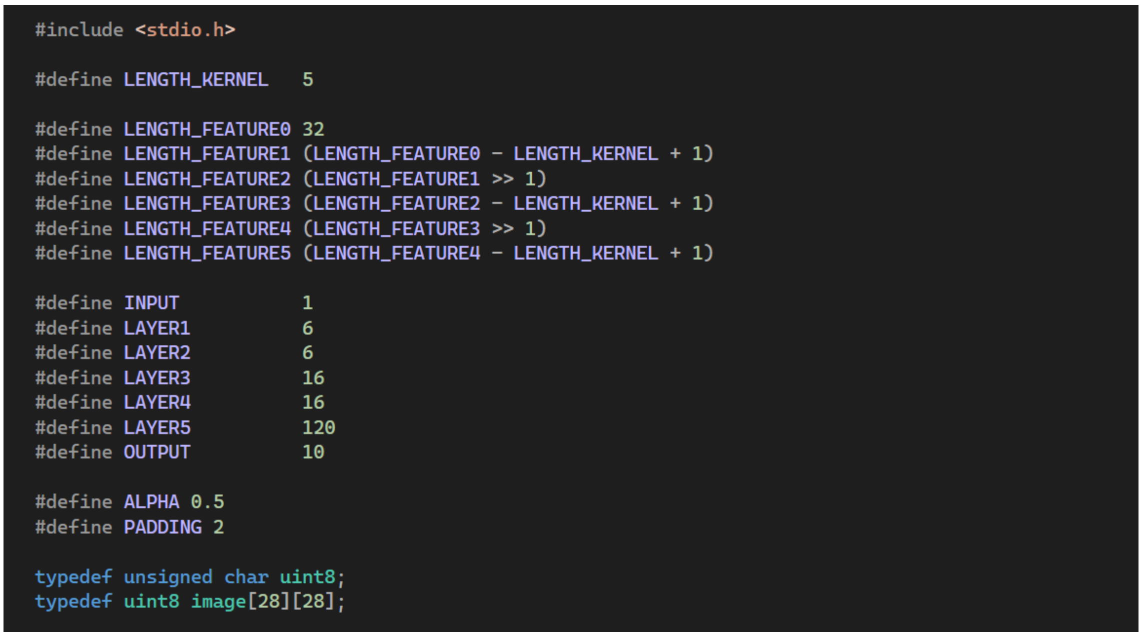
$$\begin{aligned} 4294967295 - 255 &= 4294967040 \\ \Rightarrow \frac{4294967040}{4294967295} &\approx 0.9999999406 \\ \times 100 &\approx 99.9999999406\% \end{aligned}$$

In this case also there is no impact or negligible impact on the original value on application of slow write.

4. Testing Accuracy and write time of Lenet-5

4.1 Experimental Setting

In the context of Deep Neural Networks (DNNs), the efficiency of memory operations can significantly impact overall performance. This case study explores the use of fast and slow write processes with Multi-Level Cell (MLC) memory, considering factors such as bitwidth and the structure of the DNN. Specific hardware and software settings are also discussed to provide a comprehensive understanding of the impact of these write processes.We have implemented it on Lenet-5 model.The LeNet-5 architecture starts with an input of a 28x28 grayscale image, followed by convolutional layers interleaved with pooling layers, and ends with fully connected layers.Final output layer with 10 classes (e.g., digits 0-9)



```c
#include <stdio.h>

#define LENGTH_KERNEL   5

#define LENGTH_FEATURE0 32
#define LENGTH_FEATURE1 (LENGTH_FEATURE0 - LENGTH_KERNEL + 1)
#define LENGTH_FEATURE2 (LENGTH_FEATURE1 >> 1)
#define LENGTH_FEATURE3 (LENGTH_FEATURE2 - LENGTH_KERNEL + 1)
#define LENGTH_FEATURE4 (LENGTH_FEATURE3 >> 1)
#define LENGTH_FEATURE5 (LENGTH_FEATURE4 - LENGTH_KERNEL + 1)

#define INPUT           1
#define LAYER1          6
#define LAYER2          6
#define LAYER3          16
#define LAYER4          16
#define LAYER5          120
#define OUTPUT          10

#define ALPHA 0.5
#define PADDING 2

typedef unsigned char uint8;
typedef uint8 image[28][28];
```

Figure 4: Lenet-5 NN dimention

The hardware and software setting of the DNN:The breakdown of how many pieces of data each layer has according to the Lenet-5 neural network model:
Lenet-5 is comprised of 3 convolutional layer and 3 fully connected layers and maxpooling layer after each convolutional layer.

LeNet-5 Architecture

Structure

- **Input Layer:**
  - Input size: 32×32 grayscale image.
  - MNIST 28×28 images are padded to 32×32.
- **Layer 1 - Convolution (Conv1):**
  - 6 filters of size 5×5.
  - Output size: 28×28.
  - Activation: Sigmoid.
- **Layer 2 - Subsampling (Pool1):**
  - Average pooling with 2×2 filters, stride 2.
  - Output size: 14×14.
  - Reduces spatial dimensions while preserving features.
- **Layer 3 - Convolution (Conv2):**

  - 16 filters of size 5×5, connected to subsets of S_2.
  - Output size: 10×10.
  - Captures more complex patterns.

- **Layer 4 - Subsampling (Pool2):**
  - Average pooling with 2×2 filters, stride 2.
  - Output size: 5×5.

- **Layer 5 - Fully Connected (FC1):**
  - Input: $5 \times 5 \times 16 = 400$ neurons.
  - Output: 120 neurons.
  - Dense connections map extracted features to abstract representations.

- **Layer 6 - Fully Connected (FC2):**
  - Input: 120 neurons.
  - Output: 84 neurons.
  - Acts as a classifier for high-level features.

- **Layer 7 - Output Layer:**
  - Input: 84 neurons.
  - Output: 10 neurons (for digit classification 0-9).
  - Uses softmax or Euclidean Radial Basis Function (RBF).

Output Dimension Calculations in LeNet-5

The output dimensions for each layer in LeNet-5 can be calculated using the following formulas:

1. Convolutional Layer

$$\text{Output Height (Hout)} = \frac{\text{Hin} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

$$\text{Output Width (Wout)} = \frac{\text{Win} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

Where:

- Hin, Win: Input height and width.
- **Kernel Size**: Size of the convolutional filter (assume square filters if not specified).
- **Padding**: Number of pixels added to each side of the input.
- **Stride**: Number of pixels by which the filter moves.

2. Pooling Layer

The output dimensions of a pooling layer are calculated similarly:

$$\text{Output Height (Hout)} = \frac{\text{Hin} - \text{Kernel Size}}{\text{Stride}} + 1$$

$$\text{Output Width (Wout)} = \frac{\text{Win} - \text{Kernel Size}}{\text{Stride}} + 1$$

Layer	**Kernel Size**	**Stride**	**Padding**	**Input Dimensions**	**Output Dimensions**
Input	-	-	-	$32 \times 32 \times 1$	$32 \times 32 \times 1$
Conv1	5×5	1	Valid	$32 \times 32 \times 1$	$28 \times 28 \times 6$
Pool1	2×2	2	None	$28 \times 28 \times 6$	$14 \times 14 \times 6$
Conv2	5×5	1	Valid	$14 \times 14 \times 6$	$10 \times 10 \times 16$
Pool2	2×2	2	None	$10 \times 10 \times 16$	$5 \times 5 \times 16$
FC1	-	-	None	$5 \times 5 \times 16$	$1 \times 1 \times 120$
FC2	-	-	None	$1 \times 1 \times 120$	$1 \times 1 \times 84$
Output Layer	-	-	None	$1 \times 1 \times 84$	$1 \times 1 \times 10$

Table 1: LeNet-5 Architecture

Dimensions: OUTPUT
Total data pieces: OUTPUT

Structure of the Neural Network:

Conv1: 6 * 28 * 28 = 4704 pieces of data
Conv2: 16 * 10 * 10 = 1600 pieces of data
FC1: 120 * 1 * 1 = 120 pieces of data
FC2: 84 * 1 * 1 = 84 pieces of data
Output Layer: 10 * 1 * 1 = 10 pieces of data

And as we have mentioned earlier that storing data in RRAM depends on the change in resistance levels so getting correct resistance value is essential for writing data on the 32bit data bus memory we are writing on.32 bit data bus corresponds to 8 memory cells. Each hex digit is of 16 bits so it needs 4 MLC memory all written simultaneously.

For Conv1, the minimum memory requirement = 4704*8=37632
For Conv2, the minimum memory requirement = 1600*8=12800
For FC1, the minimum memory requirement = 120*8=960
For FC2, the minimum memory requirement = 84*8=672

For Output Layer, the minimum memory requirement = 10*8=80

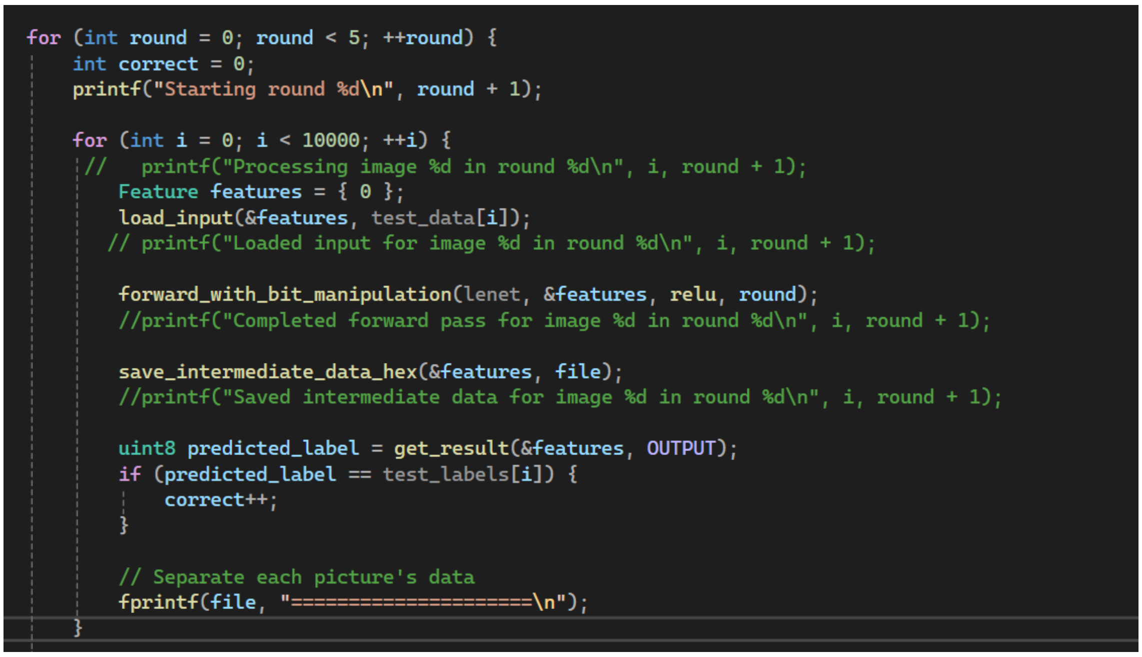
DNN Structure and Bitwidth: DNN Architecture: Consider a typical DNN architecture, such as a Convolutional Neural Network (CNN) used for image classification. Bitwidth: The precision of the weights and activations in the network. Common bitwidths include 8-bit and 16-bit, with lower bitwidths reducing memory usage and potentially increasing speed but at the risk of reduced accuracy.Here we have implemented on Lenet-5 model.

Multi-Level Cell (MLC) Memory: MLC memory stores multiple bits per cell by using different voltage levels. For instance, a 4-bit MLC can represent four states (0000, 0001, 0010, 0011 ....1111)-16 levels.

Write Processes: Fast Write: Fast write operations that may involve changing multiple bits simultaneously, including both the least significant bits (LSBs) and most significant bits (MSBs). This method is generally faster but can introduce more errors and instability, particularly in MLC memory. . [7] **Slow Write:** More controlled write operations that typically change fewer bits at a time, often focusing on the LSBs. This method is slower but provides higher reliability and accuracy.

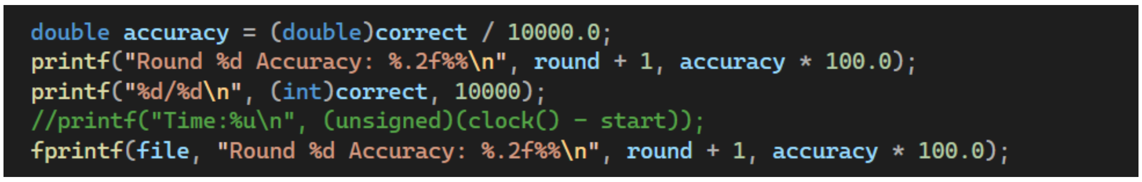
4.2 Experimental Results

As a metric for evaluation,we focused on the accuracy of the DNN.Here in this experiment we have showed the level of accuracy for 5 repetitions.



```c
for (int round = 0; round < 5; ++round) {
    int correct = 0;
    printf("Starting round %d\n", round + 1);

    for (int i = 0; i < 10000; ++i) {
//   printf("Processing image %d in round %d\n", i, round + 1);
        Feature features = { 0 };
        load_input(&features, test_data[i]);
       // printf("Loaded input for image %d in round %d\n", i, round + 1);

        forward_with_bit_manipulation(lenet, &features, relu, round);
        //printf("Completed forward pass for image %d in round %d\n", i, round + 1);

        save_intermediate_data_hex(&features, file);
        //printf("Saved intermediate data for image %d in round %d\n", i, round + 1);

        uint8 predicted_label = get_result(&features, OUTPUT);
        if (predicted_label == test_labels[i]) {
            correct++;
        }

        // Separate each picture's data
        fprintf(file, "====================\n");
    }
```

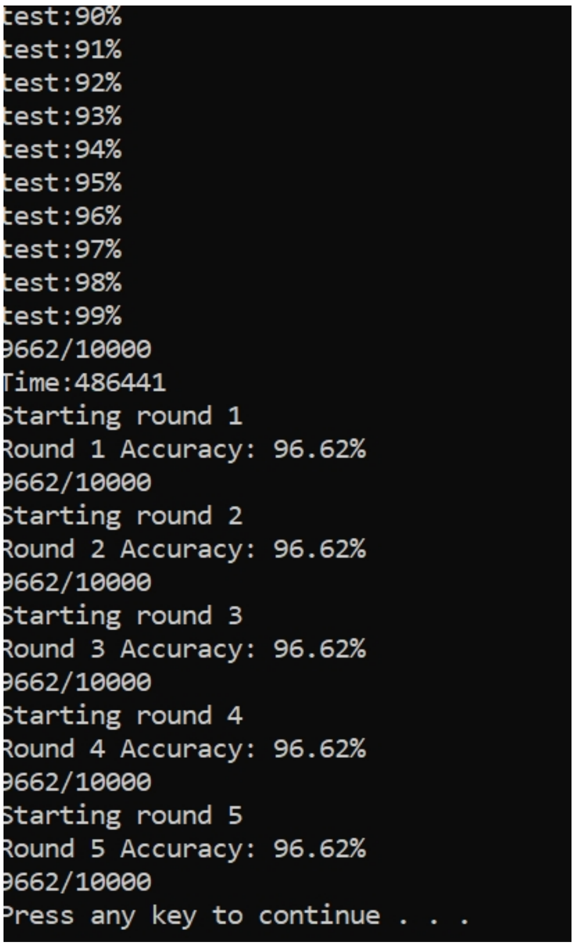
Figure 5: Accuracy of Lenet-5 neural network



```
double accuracy = (double)correct / 10000.0;
printf("Round %d Accuracy: %.2f%%\n", round + 1, accuracy * 100.0);
printf("%d/%d\n", (int)correct, 10000);
//printf("Time:%u\n", (unsigned)(clock() - start));
fprintf(file, "Round %d Accuracy: %.2f%%\n", round + 1, accuracy * 100.0);
```

Figure 6: Accuracy of Lenet-5 neural network

We get the same result every time as in 96.62 percent. This is the accuracy of the target resistance value after coming out of the layers of neural network model.Here Fast write is done on the first layer which itself gives you a significant effect that close to the target value later on the slow write impulse takes place in rest of the layers to fine tune it make it closer to the correct value.And the value of 96.62 means it gives us quite an accurate value of resistance out of the impulse hence the standard deviation of the range of resistance values is less.



```
test:90%
test:91%
test:92%
test:93%
test:94%
test:95%
test:96%
test:97%
test:98%
test:99%
9662/10000
Time:486441
Starting round 1
Round 1 Accuracy: 96.62%
9662/10000
Starting round 2
Round 2 Accuracy: 96.62%
9662/10000
Starting round 3
Round 3 Accuracy: 96.62%
9662/10000
Starting round 4
Round 4 Accuracy: 96.62%
9662/10000
Starting round 5
Round 5 Accuracy: 96.62%
9662/10000
Press any key to continue . . .
```

Figure 7: Showing Accuracy of the DNN

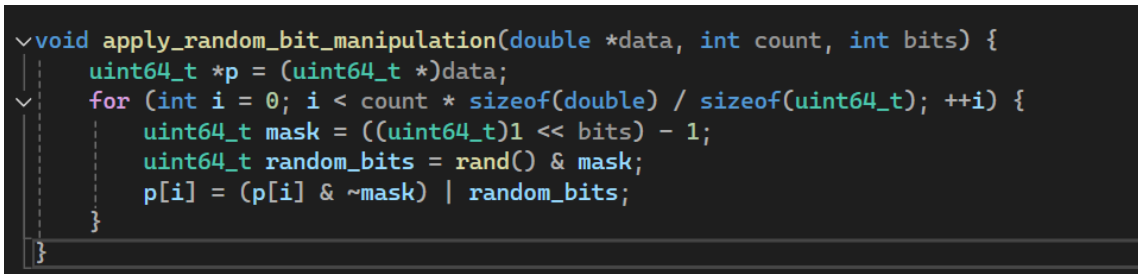
As another metric of evaluation we have included latency time i.e. the time required to write into the memory as shown in the table below:

Round	Time (ms)	Improvement (%)
First Round	6189.6	35.53
Second Round	6871.68	28.42
Third Round	9600	0
Fourth Round	9600	0
Fifth Round	9600	0

Table 2: Data for 16-bit Data Bus

Round	Time (ms)	Improvement (%)
First Round	8395.84	56.27
Second Round	12130.88	36.82
Third Round	15789.6	17.76
Fourth Round	16471.68	14.21
Fifth Round	19200	0

Table 3: Data for 8-bit Data Bus



```c
void apply_random_bit_manipulation(double *data, int count, int bits) {
    uint64_t *p = (uint64_t *)data;
    for (int i = 0; i < count * sizeof(double) / sizeof(uint64_t); ++i) {
        uint64_t mask = ((uint64_t)1 << bits) - 1;
        uint64_t random_bits = rand() & mask;
        p[i] = (p[i] & ~mask) | random_bits;
    }
}
```

Figure 8: Code changing the LSBs

The function apply_random_bit_manipulation alters the lowest bits of each double value in the array pointed to by data. The specific bits to modify are replaced with random values, while the rest of the double remains unchanged.

16-bit Data Bus:
The times for each round initially show a decreasing trend in improvements, with significant improvements seen from the first to the second round. The third to fifth rounds show no further improvement, indicating that the system has reached a point of diminishing returns or saturation in improvement with the fast write process.

8-bit Data Bus:
The times are initially higher, but improvements are more pronounced in the first two rounds. Improvement percentages decrease progressively from the first to the fifth round, reflecting reduced gains over time. The lower improvements in later rounds suggest that while fast writes initially boost performance, their effect diminishes over subsequent rounds.

Explanation of the Results:

Write Latency: Fast writes significantly reduce latency compared to slow writes. However, the improvement is more pronounced with lower bitwidths (e.g., 8-bit).
Accuracy: Using fast writes for MSBs may introduce errors, leading to a slight decrease in accuracy. Slow writes for MSBs maintain accuracy but at the cost of higher latency.
Power Consumption: Fast writes consume more power due to higher current requirements for changing multiple bits simultaneously.
Endurance: Slow writes improve the endurance of MLC memory by reducing the stress on cells. [15]

5. Analyzing the write time on DNN inference: Experimental Setting and Results

We use pulse train for reset operation to go from low to high resistant state i.e., the target resistance value. Every pulse fine-tunes towards the desired resistance value. For that, we use 100 identical pulses, each having a pulse width of 200ns after applying reset pulse train. Otherwise, without pulse train, the reset pulse width is 500ns at a reset voltage of -2.4V. Here we observe that when we apply pulse train, the pulse width shortens from 500ns to 200ns. [19]

However, for set operation, we use a single set pulse as it is meant to drop to the minimum value. As we know, the set operation is the transition from high to low resistance state. The pulse width for the set pulse is used as 5 ns [4], and the set voltage at which it occurs is 1.5V. [18] Here, the set voltage is positive, and the reset voltage is negative as it is of bipolar switching mode we are using. They must have opposite polarities.

To investigate the slow and fast write dual modes, we first take a 32-bit data bus. It means we transfer and write 32-bit data into the memory at once per cycle.

Considering the following experimental setting:

Slow write $\rightarrow$ set value to minimum and reset to maximum/target resistance (pulse train). Full reset wit

Fast write $\rightarrow$ set to minimum value and reset by 1 step (single pulse).

Considering 75% slow and 25% fast write and 100 sets of 32-bit data:

$$\text{Set} \rightarrow 5 \text{ ns}, \quad \text{Reset} \rightarrow 100 \times 500 \text{ ns} = 50000 \text{ ns}$$

$$\text{Time per slow write} = \text{set time} + 100 \text{ pulse reset time} = 5 \text{ ns} + 50000 \text{ ns}$$

$$\text{Time per fast write} = \text{set time} + 1 \text{ pulse reset time} = 5 \text{ ns} + 500 \text{ ns}$$

Write Strategy: First 24 bits use the slow write, and the last 8 bits use the fast write. Out of the 32 bits, 75% are written using slow write and 25% using fast write. Total data: We are writing 100 sets of 32-bit data values.

Weighted average time per 32-bit word:

$$= 0.75(5 \times 10^{-9} + 50000 \times 10^{-9}) + 0.25(5 \times 10^{-9} + 500 \times 10^{-9})$$

$$= 3.763 \times 10^{-5}\ \text{s} = 3.763 \times 10^{-5}\ \text{s} \times 100 = 3.763 \times 10^{-3} = 3.763\ \text{ms}$$

Therefore, the total time required to write 100 sets of 32-bit values into memory using the specified slow and fast write approach is 3.76 ms.

For a 32-bit data bus and 16-level multi-level cell, and if it is 4 bits per cell, then we need 8 memory cells to represent all the 32 bits.

- For 16 bits data: 200 data at a time - For 32 bits data: 100 data at a time - For 64 bits data: 50 data at a time

100 sets of 32-bit data is equivalent to 200 sets of 16-bit data per cycle because each 32-bit set can be split into 16-bit sets. Therefore, if we have 100 sets of 32-bit data, we can split each set into two parts resulting in $100 \times 2 = 200$ sets of 16 bits.

Similarly, it is also equivalent to 50 sets of 64-bit data per cycle because each 64-bit set can be broken down into two 32-bit sets. Therefore, if we have 50 sets of 64-bit data, we effectively have $50 \times 2 = 100$ sets of 32 bits.

$$32 \times 100 = 3200, \quad 16 \times 200 = 3200, \quad 64 \times 50 = 3200$$

Bitwidth = 16 bits, Number of data = 200: First 12 bits (75%) use slow write, and the last 4 bits (25%) use fast write. Weighted average time for 200 sets of 16-bit word:

$$= 3.763 \times 10^{-5}\ \text{s} \times 200 = 7.53\ \text{ms}$$

Bitwidth = 64 bits, Number of data = 50: First 48 bits (75%) use slow write, and the last 16 bits (25%) use fast write. Weighted average time for 200 sets of 64-bit word:

$$= 3.763 \times 10^{-5}\ \text{s} \times 50 = 1.88\ \text{ms}$$

All these represent the total time to write in the RRAM, which also corresponds to the time to write in each intermediate layer of a neural network model.

We will be applying these values to the following neural network model: 1. DanNet 2. AlexNet 3. VGG-16 4. ResNet-18

5.1. DanNet Architecture

In 2011, Jürgen Schmidhuber and Dan Ciresan unveiled DanNet, a deep convolutional neural network (CNN) architecture that achieved superhuman performance in visual pattern recognition tests, marking a key milestone in deep learning. As is common with early deep CNN designs, the network has eight layers, with two fully linked levels coming after five convolutional layers.

The layers and output dimensions of DanNet (or a comparable deep CNN architecture) depend on numerous factors, like the input size, filter sizes, strides, padding, and pooling employed in each layer. [16]

Layer	**Kernel Size**	**Stride**	**Padding**	**Input Dimensions**	**Output Dimensions**
Input	N/A	N/A	N/A	$32 \times 32 \times 3$	$32 \times 32 \times 3$
Conv1	3×3	1	0	$32 \times 32 \times 3$	$30 \times 30 \times 3$
Pool1	2×2	2	N/A	$30 \times 30 \times 3$	$15 \times 15 \times 32$
Conv2	3×3	1	0	$15 \times 15 \times 32$	$13 \times 13 \times 64$
Pool2	2×2	2	N/A	$13 \times 13 \times 64$	$6 \times 6 \times 64$
Conv3	3×3	1	0	$6 \times 6 \times 64$	$4 \times 4 \times 128$
Pool3	2×2	2	N/A	$4 \times 4 \times 128$	$2 \times 2 \times 128$
FC1	N/A	N/A	N/A	$2 \times 2 \times 128$	512
FC2	N/A	N/A	N/A	512	10

Table 4: Layer Information for the DanNet Architecture

Input dimensions: are typically height x weight x depth(channels) of the input image which is 32 x 32 x 3 for a color image. Kernel size refers to to the size of the convolutional filter(3 x 3 matrix). Stride is the number of pixels the filter moves across the image. A stride of 1 moves the filter by 1 pixel at a time. Padding:Same padding keeps the spatial dimensions the same, while valid padding reduces the size. Output Dimensions: It is calculated based on the input size, kernel size,stride and padding. For example with a stride of 1, 'same' padding and a 3 x 3 filter, the output size stays the same as the input size for that layer.

Step by step calculation of the output dimensions for each layer are following:
1. Input layer:

- Input Dimensions:32 x 32 x 3(height, width, depth)
- The image has 32 pixels in height and width with 3 color channels(RGB)

2. Conv1(Convolutional Layer 1):

- Kernel Size:3 x 3
- Stride:1
- Padding:0(valid padding)

For each dimension(height and weight):
H_out=((32-3)/1)+1=30 W_out=((32-3)/1)+1=30 hence the output dimension of Conv1 is 30 x 30. The depth is the number of filters which is 32 in this case. 3.Pool1(Pooling Layer 1):

- Kernel Size= 2 x 2
- Stride =2
- Padding =0

For each of the dimensions(height and weight): H_out=((30-2)/1)+1=15 W_out=((30-2)/1)+1=15 So the output dimension of Pool 1 layer is 15 x 15 x 32. The depth remains the same as in the previous layer. 4. Conv2(Convolutional Layer 2):

- Kernel Size: 3 x 3
- Stride: 1
- Padding: 0

For each dimension: H_out =((15-3)/1)+1=13 W_out =((15-3)/1)+1=13

So the output dimension of conv 2 layer is 13 x 13 x 64. Depth is now increased to 64 filters.

5.Pool 2(Pooling Layer 2):

- Kernel Size: 2 x 2
- Stride:2
- Padding:0

For each dimensions: H_out =((13-2)/2)+1=6 W_out =((15-3)/2)+1=6

So the output dimension of Pool 2 layer is 6 x 6x 64. Depth remains 64 filters.

6. Conv3(Convolutional Layer 3)

- Kernel Size:3 x 3
- Stride:1
- Padding:0

For each dimension: H_out =((6-2)/1)+1=4 W_out =((6-2)/1)+1=4

So the output dimension of Pool 2 layer is 4 x 4x 128. Depth increased to 128 filters.

7. Pool3(Pooling Layer 3)

- Kernel Size:2 x 2
- Stride:2
- Padding:0

For each dimension: H_out =((4-2)/2)+1=2 W_out =((4-2)/1)+1=2

So the output dimension of Pool 2 layer is 2 x 2x 128. Depth remains 128 filters.

8. Flattening the 2 x 2 x 128 output is flattened into a 1 D vector of size 2 x 2 x 128=512(1D vector)

9. FC1(Fully Connected layer 1): The flattened vector of size 512 is connected to 512 neurons.

10. FC2(Fully Connected Layer 2-Output Layer): The final fully connected layer produces the output with a size corresponding to the number of classes(e.g. 10 for MNIST)

Calculating the write times in each of these layer using the following: Using this, the write time per layer can be calculated as follows:

1. **32-bit write time per layer:**

$$\text{Write time} = \left(\frac{\text{Total 32-bit data points}}{100}\right) \times 3.763\,\text{ms}$$

2. **16-bit write time per layer:**

$$\text{Write time} = \left(\frac{\text{Total 16-bit data points}}{200}\right) \times 7.53\,\text{ms}$$

3. **64-bit write time per layer:**

$$\text{Write time} = \left(\frac{\text{Total 64-bit data points}}{50}\right) \times 1.88\,\text{ms}$$

Layer	**Output Dimensions**	**32 bit data points**	**32 bit write time(ms)**	**16 bit data points**	**16 bit write time(ms)**	**64 bit data points**	**64 bit write time(ms)**
Input	$32 \times 32 \times 3$	3072	115.6	6144	231.32	1536	57.75
Conv1	$30 \times 30 \times 3$	28800	1083.74	57600	2168.64	14400	541.44
MaxPool1	$15 \times 15 \times 32$	7200	270.94	14400	542.16	3600	135.36
Conv2	$13 \times 13 \times 64$	10816	407.006	21632	814.44	5408	203.34
Pool2	$6 \times 6 \times 64$	2304	86.7	4608	173.5	1152	43.32
Conv3	$4 \times 4 \times 128$	2048	77.06	4096	154.21	1024	38.5
Pool3	$2 \times 2 \times 128$	512	19.26	1024	38.55	256	9.63
Flatten	512	512	19.26	1024	38.55	256	9.63
FC1	512	512	19.26	1024	38.55	256	9.63
FC2	10	10	0.37	20	0.75	5	0.188
Total	-	-	2099.196	-	4162.12	-	1039.18

Table 5: Layer Information for the DanNet Architecture

Total write time in DanNet for a 32 bit data is 2099.196ms.Total write time in DanNet for a 16 bit data is 4162.12ms.Total write time in DanNet for a 64 bit data is 1039.18ms.

5.2 AlexNet Architecture

AlexNet is a deep CNN architecture that revolutionized image classification by winning the ImageNet 2012 challenge. It introduced innovations like ReLU activation for faster training, dropout regularization to prevent overfitting, and parallel GPU training for scalability. Key features include:

Input: 227×227 RGB images. Convolutional Layers: 5 layers with varying filter sizes (11×11,5×5, 3×3). Includes Local Response Normalization (LRN) and max-pooling after early layers. [14] Fully Connected Layers: 3 layers, with 4096 neurons in the first two and 1000 output neurons for classification. Dropout applied to reduce overfitting. Output: 1000-class probabilities using a softmax layer. [3]

Layer	Output Dimension	Number of 32-bit data points or words	Equivalent 16-bit words	Equivalent 64-bit words (%)
Input	227×227×3	154587	309174	77294
Conv1	55 × 55 × 96	290400	580800	145200
MaxPool1	27 × 27 × 96	69984	139968	34992
Conv2	27×27×256	186624	373248	93312
MaxPool2	13×13×256	43264	86528	21632
Conv3	13×13×384	64896	129792	32448
Conv4	13×13×384	64896	129792	32448
Conv5	13×13×256	43264	86528	21632
MaxPool3	6 × 6 × 256	9216	18432	4608
FC1	4096	4096	8192	2048
FC2	4096	4096	8192	2048
FC3	1000	1000	2000	500

Table 6: AlexNet architecture and number of words in each intermediate layers for 3 different bandwidth

Calculating Total Write Time Per Layer

For each bitwidth, the total write time is the product of the number of words and the weighted average write time per set.

- **32-bit data:** Weighted average write time = 3.763 ms per 100 sets of 32 bits of data.
- **16-bit data:** Weighted average write time = 7.53 ms per 200 sets of 16 bits of data.
- **64-bit data:** Weighted average write time = 1.88 ms.

Using this, the write time per layer can be calculated as follows:

1. **32-bit write time per layer:**

$$\text{Write time} = \left(\frac{\text{Total 32-bit data points}}{100}\right) \times 3.763\,\text{ms}$$

2. **16-bit write time per layer:**

$$\text{Write time} = \left(\frac{\text{Total 16-bit data points}}{200}\right) \times 7.53\,\text{ms}$$

3. **64-bit write time per layer:**

$$\text{Write time} = \left(\frac{\text{Total 64-bit data points}}{50}\right) \times 1.88\,\text{ms}$$

Total write time for each layer of AlexNet(based on the number of words and bitwidth):

Layer	**32-bit data points**	**32-bit write time (ms)**	**16-bit data points**	**16-bit write time (ms)**	**64-bit data points**	**64-bit write time (ms)**
Input	154587	877.68	309174	11640.4	77294	2906.25
Conv1	290400	10927.75	580800	21867.12	145200	5459.52
MaxPool1	69984	2633.49	139968	5269.79	34992	1315.69
Conv2	186624	7022.66	373248	14052.78	93312	3508.53
MaxPool2	43264	1628.02	86528	3257.78	21632	813.36
Conv3	64896	2442.04	129792	4886.67	32448	1220.04
Conv4	64896	2442.04	129792	4886.67	32448	1220.04
Conv5	43264	1628.02	86528	3257.78	21632	813.36
MaxPool3	9216	346.79	18432	693.96	4608	173.26
FC1	4096	154.13	8192	308.48	2048	77.00
FC2	4096	154.13	8192	308.48	2048	77.00
FC3	1000	37.63	2000	75.3	500	18.8
-	-	30294.38	-	70738.87	-	17525.85

Table 7: Write Time per Layer for Different Data Widths

Total write time in AlexNet for a 32 bit data is 30294.38 ms.Total write time in AlexNet for a 16 bit data is 70738.87 ms.Total write time in AlexNet for a 64 bit data is 17525.85 ms.

5.3 VGG-16 architecture

VGG-16 consists of 16 layers, including 13 convolutional layers and 3 fully connected (FC) layers.The network starts with a series of convolutional layers with small 3×3 filters and a stride of 1, which makes the network deep but relatively simple.Max-pooling layers follow some convolutional layers to reduce the spatial dimensions and allow for more abstract feature extraction.The final layers are fully connected, which help in classification tasks. The architecture culminates with a softmax layer to provide class probabilities. [11]

Layer	**Output Dimensions**	**32-bit data points**	**32-bit write time (ms)**	**16-bit data points**	**16-bit write time (ms)**	**64-bit data points**	**64-bit write time (ms)**
Input	224×224×3	150528	5664.36	301056	11334.76	75264	2829.93
Conv1_1	224 × 224 × 64	3211264	120839.86	6422528	241808.18	1605632	60371.76
Conv1_2	224 × 224 × 64	3211264	120839.86	6422528	241808.18	1605632	60371.76
MaxPool1	112 × 112 × 64	802816	30209.86	1605632	60452.05	401408	15092.94
Conv2_1	112 × 112 × 128	1605632	60419.93	3211264	120904.09	802816	30185.88
Conv2_2	112 × 112 × 128	1605632	60419.93	3211264	120904.09	802816	30185.88
MaxPool2	56×56×128	401408	15104.98	802816	30226.02	200704	7546.47
Conv3_1	56×56×256	802816	30209.96	1605632	60452.05	401408	15080.6
Conv3_2	56×56×256	802816	30209.96	1605632	60452.05	401408	15080.6
Conv3_3	56×56×256	802816	30209.96	1605632	60452.05	401408	15080.6
MaxPool3	28×28×256	200704	7552.5	401408	15113.01	100352	3773.24
Conv4_1	28×28×512	401408	15104.98	802816	30226.02	200704	7546.47
Conv4_2	28×28×512	401408	15104.98	802816	30226.02	200704	7546.47
Conv4_3	28×28×512	401408	15104.98	802816	30226.02	200704	7546.47
MaxPool4	14×14×512	100352	3776.25	200704	7556.5	50176	1886.62
Conv5_1	14×14×512	100352	3776.25	200704	7556.5	50176	1886.62
Conv5_2	14×14×512	100352	3776.25	200704	7556.5	50176	1886.62
Conv5_3	14×14×512	100352	3776.25	200704	7556.5	50176	1886.62
MaxPool5	7 × 7 × 512	25088	944.06	50176	1889.13	3136	117.91
FC1	-	4096	154.13	8192	308.48	2048	77.00
FC2	-	4096	154.13	8192	308.48	2048	77.00
FC3	-	1000	5.68	2000	75.3	500	18.8
Total	-	437414.36	-	4359307.18	-	245358.64	-

Table 8: Write Time per Layer for Different Data Widths VGG-16

Total write time in VGG-16 for a 32 bit data is 437414.36ms.Total write time in VGG-16 for a 16 bit data is 4359307.18ms.Total write time in VGG-16 for a 64 bit data is 245358.64ms.

5.4 ResNet-18 Architecture

The ResNet-18 (Residual Network with 18 layers) architecture is a widely used deep convolutional neural network that introduced the concept of residual learning to address the vanishing gradient problem in deep networks. It uses skip connections to allow gradients to flow t hrough t he n etwork m ore e ffectively. [12]

Main Stages of ResNet:

1. **Initial Convolution and Max Pooling Layers:** The network begins with a convolutional layer followed by a max pooling layer to downsample the input.

2. **Four Stages of Residual Blocks:** Each stage comprises residual blocks with increasing filter sizes:

   - **Stage 1:** Two 3×3 convolutional layers with 64 filters.
   - **Stage 2:** Two 3×3 convolutional layers with 128 filters.
   - **Stage 3:** Two 3×3 convolutional layers with 256 filters.
   - **Stage 4:** Two 3×3 convolutional layers with 512 filters.

3. **Average Pooling and Fully Connected (FC) Layer:** After the residual blocks, global average pooling is used to reduce the spatial dimensions, followed by a fully connected layer to produce the final output.

Layer	**Output Dimensions**	**32-bit data points**	**32-bit write time (ms)**	**16-bit data points**	**16-bit write time (ms)**	**64-bit data points**	**64-bit write time (ms)**
Input	224×224×3	150528	22729.73	301056	45459.46	75264	2829.93
Conv1_1	112 × 112 × 64	802816	30209.96	160432	6040.26	401408	15092.94
MaxPool1	56 × 56 × 64	200704	7552.50	401408	15223.011	100352	3773.24
Conv2_1_1	56 × 56 × 64	200704	7552.50	401408	15223.011	100352	3773.24
Conv2_1_2	56 × 56 × 64	200704	7552.50	401408	15223.011	100352	3773.24
Conv2_2_1	56 × 56 × 64	200704	7552.50	401408	15223.011	100352	3773.24
Conv2_2_2	56 × 56 × 64	200704	7552.50	401408	15223.011	100352	3773.24
Conv3_1_1	28×28×128	100352	3776.25	200704	7556.51	50176	1886.62
Conv3_1_2	28×28×128	100352	3776.25	200704	7556.51	50176	1886.62
Conv3_2_1	28×28×128	100352	3776.25	200704	7556.51	50176	1886.62
Conv3_2_2	28×28×128	100352	3776.25	200704	7556.51	50176	1886.62
Conv4_1_1	14×14×256	50176	1888.12	100352	3778.25	25088	943.31
Conv4_1_2	14×14×256	50176	1888.12	100352	3778.25	25088	943.31
Conv4_2_1	14×14×256	50176	1888.12	100352	3778.25	25088	943.31
Conv4_2_2	14×14×256	50176	1888.12	100352	3778.25	25088	943.31
Conv5_1_1	7 × 7 × 512	25088	944.06	12544	472.28	50176	188.62
Conv5_1_2	7 × 7 × 512	25088	944.06	12544	472.28	50176	188.62
Conv5_2_1	7 × 7 × 512	25088	944.06	12544	472.28	50176	188.62
Conv5_2_2	7 × 7 × 512	25088	944.06	12544	472.28	50176	188.62
Global Average Pooling	1 × 1 × 512	512	19.26	256	9.64	1024	38.5
Global Average Pooling	1 × 1 × 1000	1000	37.63	500	18.83	2000	75.2
Total	-	-	109640.3	-	174321.41	-	51995.73

Table 9: Write Time per Layer for Different Data Widths

Total write time in ResNet-18 for a 32 bit data is 109640.3 ms.Total write time in ResNet-18for a 16 bit data is 174321.41 ms.Total write time in ResNet-18 for a 64 bit data is 51995.73 ms.

5.5 Write Time Analysis for DanNet, AlexNet,VGG-16 and ResNet-18

Neural Network Model	**Total write time for 32 bit data(ms)**	**Total write time for 16 bit data(ms)**	**Total write time for 64 bit data(ms)**
DanNet	2099.196	4162.12	1039.18
AlexNet	30294.38	70738.87	17525.85
VGG-16	437414.36	4359307.18	245358.64
ResNet-18	109640.3	174321.41	51995.73

Table 10: Summary of all the neural network model and its total write times for 32 bit, 16 bit and 64 bit data.

Discussion on the above results:

The 64-bit write time for DanNet is the shortest, indicating that more efficient memory writes might result from wider data widths (64-bit). This is likely because larger data types reduce overhead in memory management and data transport. However, the 16-bit write time is significantly longer than the 32-bit write time, suggesting that smaller data widths introduce more complicated data processing. This could be due to increased fragmentation or the requirement for additional operations to handle smaller data chunks.

A similar pattern can be observed in AlexNet, where the 64-bit write time is noticeably faster than the 32-bit and 16-bit write timings. The difference is particularly pronounced for the 16-bit write time, indicating that smaller data types (16-bit) are much less efficient for AlexNet. The longer write time for 16-bit could be attributed to additional operations required to handle smaller data chunks or to control accuracy loss in computations when using reduced bit widths.

Although VGG-16 exhibits a similar pattern, the write time difference is significantly more noticeable, especially when comparing 16-bit and 32-bit. This suggests that VGG-16 is highly sensitive to the employed bit width, with a drastic increase in write time observed when the data width is reduced to 16 bits. This behavior may result from the large number of parameters in VGG-16, where processing smaller data types requires more memory writes. As with the other models, the 64-bit write time is significantly faster, demonstrating that larger data types can mitigate inefficiencies caused by smaller data lengths.

For ResNet-18, the 64-bit write time is again the most efficient, and the 16-bit write time is much higher, which is consistent with the trend seen in the other models. The 32-bit write time is noticeably better than 16-bit but not as good as 64-bit, suggesting that while reducing bit width to 16-bit can save memory and computational resources, it comes with a significant penalty in terms of data handling efficiency.

5.6 Implications of Write Time on Neural Networks

Efficiency: The to tal wr ite ti me di rectly im pacts th e th roughput of th e mo del during training and inference. Lower write times allow for faster model execution, making larger bit widths (e.g., 64-bit) more efficient in te rms of me mory ba ndwidth, es pecially in large models like VGG-16. [1]

Data Precision: Reducing the bit width (e.g., using 16-bit instead of 32-bit) saves memory and computational resources but may increase the write time and reduce precision. This trade-off b etween s peed a nd p recision i s i mportant i n n eural n etwork optimizations, especially for large models with many parameters. However, the substantial increase in write time for 16-bit precision suggests that the reduction in data width is not always the most optimal choice, depending on the network architecture. [1]

Memory Access Patterns: Models with more complex architectures (e.g., VGG-16) may suffer m ore f rom r educed b it w idths, a s t heir m emory a ccess p atterns a nd n eed for frequent data writes increase. As a result, they experience higher penalties in write time when switching to smaller bit widths.

Impact on Performance: The overall performance of a neural network can be significantly impacted by how data is written to and read from memory. For some architectures, like VGG-16, the larger data widths (64-bit) offer better memory handling, while for others (like DanNet), 32-bit or 64-bit widths seem to perform better compared to 16-bit due to the increased overhead when using smaller bit widths. [1]

Final Remarks: Bit width plays a critical role in the write time of neural networks. While reducing bit width can save memory, it often results in higher write times, especially for complex models. Larger bit widths (64-bit) tend to reduce the overall write time, improving memory handling efficiency. Sm aller bi t wi dths (1 6-bit) ma y le ad to longer write times, which could be counterproductive, particularly for larger models with many layers and parameters. In practice, finding t he r ight b alance b etween b it w idth, model complexity, and hardware limitations is key to optimizing neural network performance.

6. Conclusion

In conclusion, we have thoroughly examined how the fast write ans slow write approaches affects t he D NN i ntermediate l ayers a nd fi nding to tal wr ite ti me,compared fo r 3 different bitwidths. It signifies t he p ulse-train o peration m ethod p articularly f or r eset operation for controlling multi-level conductive filament e volution R eRAM d evices, demonstrating its effectiveness i n a chieving p recise a nd r eliable m ulti-level c ontrol.The r esearch provides critical insights into improving neuromorphic computing through better analog switching and multi-level switching consistency. . [9] The findings underscore the potential of RRAM devices in electronic synapse applications, particularly when using deep neural network-inspired approaches for fine-tuning r esistance values.

7. References

[1] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. Journal_of_Big Data, 8(1):53, 2021.

[2] S. Angizi, Z. He, D. Reis, X. S. Hu, W. Tsai, S. J. Lin, and D. Fan. Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach? In 2019_IEEE_Computer_Society_Annual_Symposium_on_VLSI_(ISVLSI), pages 197–202, 2019.

[3] S. B. Alexnet architecture explained, 2024. Accessed: 2024-11-21.

[4] I. Baek, D. Kim, M. Lee, H.-J. Kim, E. Yim, M. Lee, J. Lee, S. Ahn, S. Seo, J. Lee, J. Park, Y. Cha, S. Park, H. Kim, I. Yoo, U. Chung, J. Moon, and B. Ryu. Multi-layer cross-point binary oxide resistive memory (oxrram) for post-nand storage application. In IEEE_InternationalElectron_Devices_Meeting,_2005._IEDM_Technical_Digest., pages 750–753, 2005.

[5] A. Bagheri-Soulla and M. Ghaznavi-Ghoushchi. An rram-based mlc design approach. Microelectronics_Journal, 64:9–18, 2017.

[6] C. Cagli, D. Ielmini, F. Nardi, and A. L. Lacaita. Evidence for threshold switching in the set process of nio-based rram and physical modeling for set, reset, retention and disturb prediction. In 2008_IEEE_International_Electron_Devices_Meeting, pages 1–4, 2008.

[7] C. Chen, L. Goux, A. Fantini, R. Degraeve, A. Redolfi, G. Groeseneken, and M. Jurczak. Stack optimization of oxide-based rram for fast write speed ($<$1s) at low operating current ($<$10a). Solid-State_Electronics, 125:198–203, 2016. Extended papers selected from ESSDERC 2015.

[8] F. A. K. Furqan Zahoor, Tun Zainal Azni Zulkifli. Multi-level control of conductive nano-filament evolution in hfo2 reram by pulse-train operations. SPRINGER_LINK, 6, 2014.

[9] F. A. K. Furqan Zahoor, Tun Zainal Azni Zulkifli. Resistive random access memory (rram): an overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications. SPRINGER_LINK, 15(90):672–675, 2020.

[10] D. Ielmini. Resistive switching memories based on metal oxides: mechanisms, reliability and scaling. In IOPScience_Semiconductor_Science_and_Technology, volume 31, 2016.

[11] A. H. M. Linkon, M. M. Labib, T. Hasan, M. Hossain, and Marium-E-Jannat. Deep learning in prostate cancer diagnosis and gleason grading in histopathology images: An extensive study. Informatics_in_Medicine_Unlocked, 24:100582, 2021.

[12] P. Napoletano, F. Piccoli, and R. Schettini. Anomaly detection in nanofibrous materials by cnn-based self-similarity. Sensors_(Basel,_Switzerland), 18, 01 2018.

[13] A. Padovani, J. Woo, H. Hwang, and L. Larcher. Understanding and optimization of pulsed set operation in hfox-based rram devices for neuromorphic computing applications. IEEE_Electron_Device_Letters, 39(5):672–675, 2018.

[14] B. Prasad. Cnn architectures: Alex net, le net, vgg, google net, res net. ResearchGate, 2022.

[15] G. Sassine, C. Nail, L. Tillie, D. A. Robayo, A. Levisse, C. Cagli, K. E. Hajjam, J.-F. Nodin, E. Vianello, M. Bernard, G. Molas, and E. Nowak. Sub-pj consumption and short latency time in rram arrays for high endurance applications. In 2018_IEEE International_Reliability_Physics_Symposium_(IRPS), pages P–MY.2–1–P–MY.2–5, 2018.

[16] J. Schmidhuber. Dannet: Triggering the deep convolutional neural network revolution. In Proceedings_of_the_International_Joint_Conference_on_Neural_Networks (IJCNN), 2011. Detailed overview available at Schmidhuber's annotated history of AI: https://people.idsia.ch/ juergen/DanNet-triggers-deep-CNN-revolution-2011.html.

[17] Z. A. K. S. K. S. Weijian Chen, Zhi Qi. Resistive-ram-based in-memory computing for neural network: A review. In Electronics, volume 11, 2022.

[18] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai. Metal–oxide rram. Proceedings_of_the_IEEE, 100(6):1951–1970, 2012.

[19] L. Zhao, H. Y. Chen, S. C. Wu, Z. Jiang, S. Yu, T. H. Hou, H. S. Wong, and Y. Nishi. Multi-level control of conductive nano-filament evolution in hfo2 reram by pulse-train operations. Nanoscale, 6(11):5698–5702, Jun 2014. Epub 2014 Apr 28.