# DERIVED ECC FOR PROTECTION OF ON-DEMAND DATA

by

**Sebastien Ollivier**

Submitted to the Graduate Faculty of
the Swanson School of Engineering in partial fulfillment
of the requirements for the degree of
**Master of Science**

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Sebastien Ollivier

It was defended on

November 28, 2018

and approved by

Alex K. Jones, Ph.D., Professor
Department of Electrical and Computer Engineering

Jingtong Hu, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Samuel J. Dickerson, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Thesis Advisor: Alex K. Jones, Ph.D., Professor
Department of Electrical and Computer Engineering

**DERIVED ECC FOR PROTECTION OF ON-DEMAND DATA**

Sebastien Ollivier, M.S.

University of Pittsburgh, 2018

Traditional error correction coding (ECC) methodologies store data parity bits along with data they protect. Subsequently, upon accessing the combined data and parity bits it is possible discover and correct faults in either the data or correction bits. In domain-wall memories (DWMs), a type of sequential access non-volatile memory related to spin-transfer torque magnetic memory (STT-MRAM), it is not convenient or efficient to store data to protect access errors data shifting during sequential access. To solve this problem, we propose a new technique called *derived error correction* (DEC) for such cases where it is intractable to record metadata for error correction. Instead, we rebuild the metadata *on-demand* and store only the parity bits that protect the metadata. The DWM metadata is constructed using a novel *transverse read* (TR). TR reads in an orthogonal direction of a traditional DWM access point and can be used to calculate number of ones in a DWM. Faults in the metadata correspond to *shift-errors* in the DWM. Using ECC, we can correct these faults in the metadata and use these corrections to repair the shifting state of the DWM to ensure correct operation. Through these techniques, we propose a shift-aware error correction code that provides a lifetime over 10 years while reducing area by 2.6 and 3.7 times against state of the art technique.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# 1.0 INTRODUCTION

The progression of process scaling has enabled increasingly faster and higher-capacity caches, which in turn along with parallelization and networks-on-chips has enabled the continued improvements of computational performance. While the scaling of traditional memory technology used for caches, SRAM, has enabled this progression, current SRAM caches now dominate both chip-area and power consumption of the device. In order to reduce the substantial static energy consumption of SRAM, non-volatile memories such as STT-MRAM have been considered as replacements for SRAM [3]. STT-MRAM preserves the performance of SRAM while being low powered. That's why people as Tom Coughlin expects this technology revenues to reach 3.3 billion dollars by 2028.

While STT-MRAM can yield considerable savings versus SRAM, an improvement to SRAM based on similar technology, Domain-Wall Memory, can potentially yield substantial savings to both area and power.

Made of tapes (ferromagnetic nanowire) separated in domains, DWM conserves the advantages of STT-MRAM while increasing the storage density up to 10x according to previous study [4]. To read or write a value inside a domain, it has to be align with a read/write head (similar to a STT-RAM cell). In order to make that possible, a current is either send through the head or the tail of the nanowire.

However, due to imprecision in the fabrication of these highly-scaled domain-wall memory tapes [5], slight variations in current combined with these imperfections can cause errors in the shifting process. These errors include over-shifting, under-shifting, and pinning errors (where only part of the tape shifts). While all of these errors, especially pinnning, should become more infrequent as the lithography process matures, recent modeling has demon-

strated that the current error rates of under-shifting and over-shifting is significant [1], and must be addressed to feasibly consider using domain-wall memory in commercial systems.

Classical ECC code will interact with the data under the different read head but won't be able to interact with the position of the domains. Due to that, previous works [1, 6] develop interesting new algorithms. However, all this algorithms need to sacrifice advantages of the racetrack to minimize the impact of the shifting errors, either by degrading the performance or by using extra area (high number of additional read/write heads and/or additional domains).

In this paper, we propose DEC (Derive error correction), a method to correct errors which doesn't interact directly with the data, but with information created from the data. By interacting with these elements, DEC enables the correction of misaligned racetrack. The correction code applied on the data is a combination of hamming code and additional parity bits. In this paper, we also demonstrate the new concept of transverse read, which gives the number of ones between two ports (read head to the head or read head to the tail) of the racetrack. Applying these two new concepts to fused cache [2] enable the correction of a misalignment by any number. In addition, while being as reliable as previous scheme it decrease the area overhead.

In this paper we provide the following contributions:

- Transverse Read
- Arbitrary Shift Error Correction
- Reduced Area Per Tape
- We use overhead bits like p-ECC-O, we can arbitrarily shift like p-ECC, and we can correct arbitrary shift errors

## 2.0 BACKGROUND

In this section, we will present basic information about racetrack memory as well as a short description of the previous algorithms from [1] developed to resolve the shifting errors.

### 2.1 RACETRACK MEMORY

Domain wall memory (DWM) is an array of nanowires[5]. Each of them is composed of domains separated by domain walls. The value of a domain is determined by its polarization. As STT-MRAM, DWM is a spintronic memory. When the domain is under a magnetoresistance also call read head if their polarization are in the same direction the resistance will be minimal and inversely if they are in opposite directions the resistance will be maximal (same principle as STT-MRAM). Read and write heads are fix elements. Thus, a domain needs to be shifted in order to be aligned with the head to be read or written. Depending on the direction needed, a pulse is send either at the head or the tail of the nanowire. Extra-domains are on each side of the data to prevent any lost while shifting to an extremity. Once a domain is under the head another current is used to read or write the domain. On figure 1(a) we can see the racetrack composition in different position. Data domains are represented in blue and the one in dark blue is under the head. Domain walls are in green and the extra-domains in grey. Several nanowires receive a pulse at the same time to allow the system to read/write a word or a cache line depending on the usage of the DWM.

This technology has been proposed for utilization in a variety of positions in the memory hierarchy, including in network-on-chips [7], as part of the cache hierarchy representing the last-level cache [8] and multiple cache levels [2], and as a fast main-memory technology [9].

Table 1: Shift error probabilities, for k=1 and k=2 bit shifting offsets, from [1].

| Shifting Distance | ±k Step Error Rate | |
|---|---|---|
| | k=1 | k=2 |
| 1 | $4.55 \times 10^{-5}$ | $1.37 \times 10^{-21}$ |
| 2 | $9.95 \times 10^{-5}$ | $1.19 \times 10^{-20}$ |
| 3 | $2.07 \times 10^{-4}$ | $5.59 \times 10^{-20}$ |
| 4 | $3.76 \times 10^{-4}$ | $1.80 \times 10^{-19}$ |
| 5 | $5.94 \times 10^{-4}$ | $4.47 \times 10^{-19}$ |
| 6 | $8.43 \times 10^{-4}$ | $9.96 \times 10^{-18}$ |
| 7 | $1.10 \times 10^{-3}$ | $7.57 \times 10^{-15}$ |

In this work, we will assume the technology is utilized at the cache level, using the Fused-Cache technique [2]. While using the nanowires as cache, they can over or under-shift, the probabilities of that happening are given in table 1. If a shifting error occurs, this special nanowire will never be correctly align with the others which results in several faults appearing. Hence, the importance to detect and correct a misalignment as soon as it appears.

A previous work proposed to add read heads on each nanowire and a specific pattern on the extra domains to detect and correct this issue [1]. We propose a solution coupling DWM with STT-MRAM which minimize the space overhead while enabling the correction of an arbitrary shift error.

### 2.2 FUSED CACHE

The principle is to use blocks of racetrack as L1 and L2 cache. The domains under the head are considered as L1 and all the other domains are L2. To read a cache line of 512 bits, 512 racetracks will read the values under their head in L1. To demote or promote a cache line from L2 to L1 a block of racetrack needs to be shifted. Using a fused cache made of

racetrack, enables a performance gain of seven percent and a 69 percent reduction in cache energy compared with an iso-capacity two level SRAM cache. In addition, DWM consumes less area per bits, so for the same space more data can be stored.

## 2.3 INSERTION WHILE OVER-SHIFTING/UNDER-SHIFTING

While shifting the racetrack, not only one but all the domains are shifting, as a consequence extra domains are needed to conserve the main data. The number of head (min(read head, write head)) and their distances to useful data determine the length of the extra domains. A racetrack needs to be able to read or write any data domains without losing data.

While adding heads decreases the number of shift needed to access a specific domains and so improve the performance, it is adding an area overhead. It's a trade off between area and performance.

If needed, a special value ('0' or '1') can be inserted in the domain when shifting in. However, when a domain over-shift or under-shift it impacts the insertion.

Let's say we want a pattern that alternately inserts one and zero. If at one point the racetrack over-shift then it will insert twice the same value, and if it under-shift then the racetrack considers that a value have been inserted and go to the next one. In both case, two times in a row the same value will be inserted, which would break the pattern.

## 2.4 ECC FOR DWM CACHES

A natural consideration for any error-prone memory is to consider the use of ECC codes to protect its stored data. However, the nature of the over and under-shifting errors makes the simple application of ECC inadequate and ill-advised. "Correcting" a bit caused by a shifting error does not solve the underlying problem: that all bits in that tape are out of position, and that they will remain out of position for accesses to other words in that cache block. Future word accesses will have one tape already out of position, greatly reducing

the effectiveness of the ECC. Further, because of the organization of racetrack memory, an uncorrectable ECC error would result in the entire cache block having to be scrubbed, not just an individual word, resulting in degradation to performance. As a consequence, alternative methods of error correction have been proposed which overcome these limitations. In the next paragraph, we will present several such techniques.

### 2.5 P-ECC, P-ECC-O

Several techniques were developed by [1] in order to provide superior correction to ECC for domain-wall memory shift errors.

To accomplish this superior correction, both p-ECC and p-ECC-O took advantage of the fact that both under-shifting and over-shifting is observed over the entire length of the tape. Therefore, adding additional read/write heads and domains on either side of the tapes, known patterns can be used to detect when a misalignment has occurred. Thus, the idea is to read a pattern that will have a different output if an error occurred.

Their goal being to correct a misalignment by one they decided to alternate '00' and '11'. Using two additional heads, if for instance they read '01' while they should have read '00', they detect the error and they can determine if it's an under or over-shift. The pattern location differentiate p-ECC from p-ECC-O. For the first one, additional domains are added to store the pattern. The number of domain added depends on the maximum movement needed to reach every data. In the other hand, p-ECC-O uses the extra-domains that are already necessary for the racetrack to shift without losing any data. P-ECC will privilege a small latency overhead over area overhead, at the opposite p-ECC-O sacrifices less area but loses in performance. P-ECC-O can only shift by one domain at a time in order to implement the pattern inside the extra-domains. Their scheme is scalable to detect and correct more errors.

## 3.0 TRANSVERSE READ

While the traditional current-based read of STT-MRAM under a single domain is the obvious mode of reading, the nanowire implementation of domain-wall memory enables new possibility. Our simulations demonstrate that an alternative method of reading along the racetrack is possible. We call this alternative mode of reading transverse read (TR).

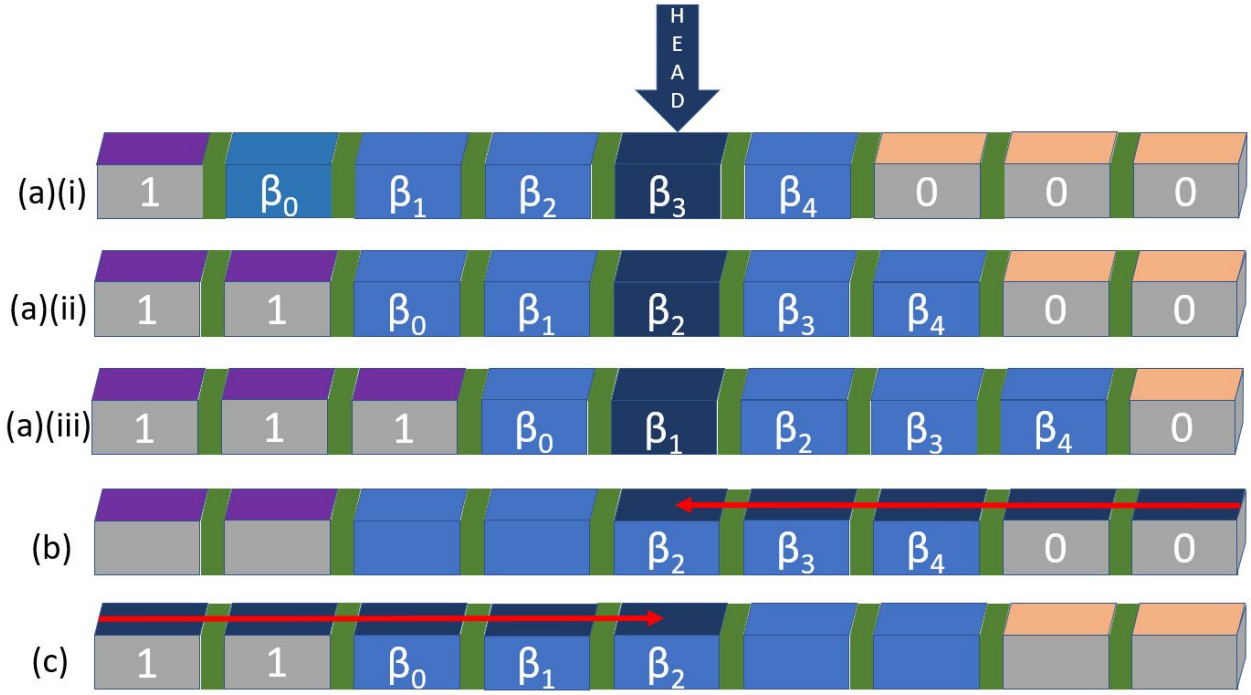


Figure 1: (a) Shows the traditional MTJ form of reading the domain under the read head. (b) Shows a transverse read along the right of the read head (c) Shows a transverse read along the left of the read head

### 3.1 TRANSVERSE READ BEHAVIOR

In contrast to the traditional method of reading discussed in the preliminaries, transverse reads provide information through multiple domains. To be precise it gives the number of one between an extremity and the head without knowing their position. To get this result a current is sent from one extremity while the other extremity behave as an open circuit.

Figure 1(b) and (c) represent respectively a transverse read performs from each extremities. As precised by the arrow on the figures, even the domain under the read head is include in the transverse read. In (b) the transverse read produce $\beta_2$+$\beta_3$+$\beta_4$+0+0, and in (c): 1+1+$\beta_0$+$\beta_1$+$\beta_2$. Here, plus represents a traditional addition. the result from (b) can be either zero, one, two or three. The result from (c) is between two and five inclusive.

### 3.2 TRANSVERSE READ UTILIZATION

In this paper, the ultimate goal of introducing the transverse read is to enable efficient correction of domain-wall memory shift errors. To accomplish this protection, TR will first provide the total number of ones inside a racetrack line. Due to a special pattern located in the extra-domains, by computing the number of ones our algorithm determines the racetrack position. Thus, If an undershift or overshift error occurs, the calculated number of ones will differ from the expected value. Using the difference from the expected value, we can then correct the error.

First, to calculate the total number of ones per line two transverse reads and a normal read are required. After performing the transverse reads shown in figure 1(b) and (c) and adding both values the result is: 1+1+$\beta_0$+$\beta_1$+$\beta_2$ + $\beta_2$+$\beta_3$+$\beta_4$+0+0. To remove the extra value of $\beta_2$, we perform a normal read, and use the value of $\beta_2$ to either subtract one from the total (in the case $\beta_2$ = '1') or keep the calculated value (in the case $\beta_2$ = '0'). The resulting sum gives us the number of ones-per-line (OPL). The classical read is performed first in order to, wherever possible, hide the latency of the transverse read verification.

As seen in Figure 1 the pattern used in the extra domains consists of ones for the extra-domains on one side (purple domains) and zeroes for those on the other side (beige domains). This pattern has been chosen for two reasons: (1) every possible alignment has a unique number of ones in the extra-domains, (2) while over-shifting or under-shifting the pattern isn't disturbed. If we look at Figure 1(a)(ii) where $\beta_2$ is aligned with the read head, there will always be a total of two ones in the extra-domains. While it's $\beta_1$ under the head (a)(iii), three ones are in the extra-domains, even if we arrived in this position due to an under-shift or over-shift. Therefore, the number of ones in the extra domains directly corresponds to the position of the racetrack.

Now, we will explain how after any shift operation, we can calculate the number of ones in the extra domains, and in doing so, ascertain the current position of the racetrack. During the initialization, we know the current position of the racetrack. As a consequence the number of ones on the side is known. By subtracting this value to the result from two paragraphs above (OPL), the Hamming weight to zero (hw) of the main data is obtained. This information will be used in DEC.

After any shift, OPL is read then OPL-hw is computed which is equal to (one-on-left-side+$\beta_0$+$\beta_1$+$\beta_2$+$\beta_3$+$\beta_4$+zeros) - ($\beta_0$+$\beta_1$+$\beta_2$+$\beta_3$+$\beta_4$) = one on left side + zeros. In other words, the position of the racetrack.

Keeping the nanowire as it appears in figure 1, we consider for the example that: $\beta_0$=$\beta_1$=$\beta_2$=$\beta_3$=0 and $\beta_4$=1. After initialization hw=1. Considering the initial position to be $\beta$3 and $\beta$2 the destination. The DWM should shift by one. If shifted correctly, opl=3 and hw=1. So opl-hw=2 which is the number of one in the extra-domains on the left side. In the case of an over-shift error, opl=4, and hw=1, which gives Sb=3$\neq$2. An error is detected and the correction that needs to be applied is known.

## 4.0 DERIVE ECC

The primary contribution of this work is Derived Error Correction (DEC). While, in some cases, as for example DWM, ECC is not effective nor efficient, the concept of DEC is to perform a more efficient correction on a derived representation of the data instead of the data itself. The primary benefit of using DEC is the area considerable saving versus performing error correction directly on the data. By accessing the data on demand, the derived elements can also be created on demand. Therefore, it's unnecessary to store the derived elements. That's why, in DEC we only store parity bits. When an error occurs the derived elements will generate parity bits those won't be as expected and when comparing with the stored bits an error will be detected and corrected. This method can be used when the main data can be read on demand. If our elements that are needed to protect the data are of length n then the hamming code needs log(n)+2 bits in order to protect it. The second extra bit is to guaranty the detection of a second error. Using the classical ECC(1) code n+log(n)+2 bits would be stored (variable + parity bits), when using the derive ECC only log(n)+2 bits has to be stored (parity bits).

## 4.1 DEC FOR DWM CACHES

In the case of position error in racetrack memory, we will use DEC to protect the number of one per racetrack.

As seen in section III-B, if the number of data-one per racetrack is known we can correct any position error. However, storing log(n) bits per racetrack (where n is the number of data bits per racetrack) would consume too much space and accessing them would also consume

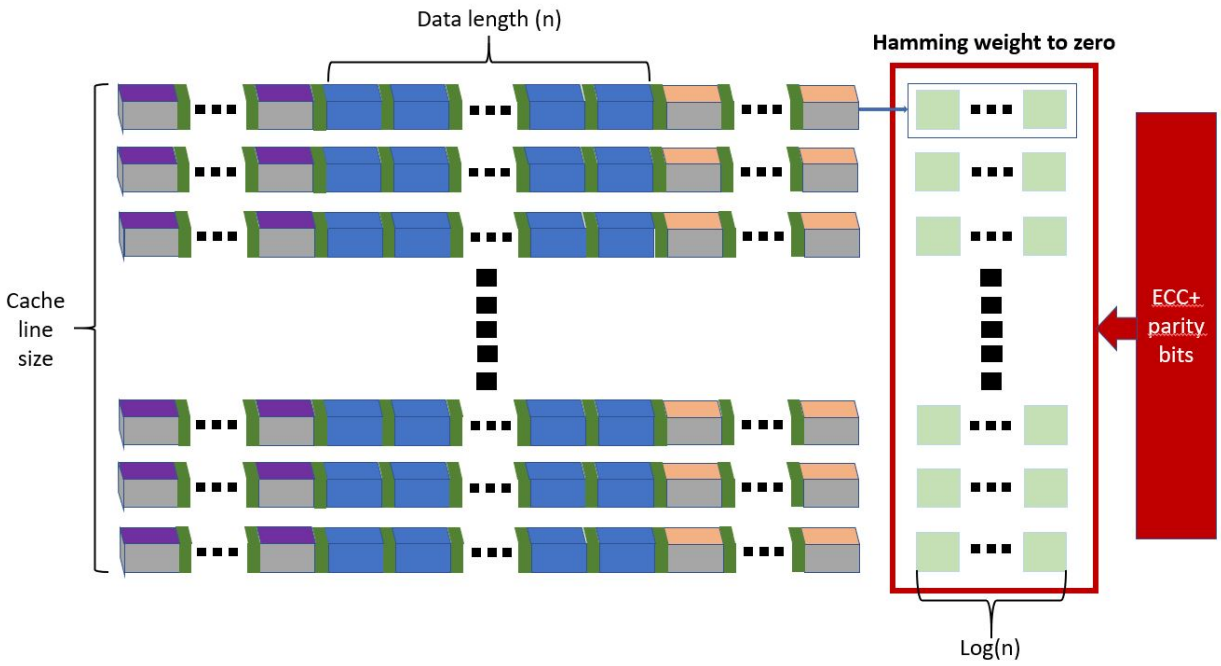


Figure 2: DEC for DWM cache

a large amount of energy. In spite, racetrack are grouped in blocks, and for each of these blocks, DEC will be applied. The bits generated by DEC are stored in STT-MRAM cells, which are considered as secured memories, thus, fault free.

First, hamming weight to zero (HW0) is generated (using TR) for each racetrack, figure 2. Then the protection code is applied on these elements. In our case, the protection code is composed of two elements. HW0 parity per line (one bit per line), and an ECC1 block per column (log(column length)+2 * number of columns) as represented in Figure 3.

As seen in the background section, the racetrack is used as fused cache, in such manner, we consider the length of a racetrack block equal to the length of the cache line (LCL). Figure 2 is composed of LCL nanowires, of length n (number of domains with important data, blue domains). The maximum number of ones visible in a line of length n is $\log_2(n)$. So, the number of column is $\log_2(n)$. In order to take an easier example, we consider only for now, that LCL=512, n=8 and $\log_2(n)$=3.

In the initialization, three bits $\alpha 0$, $\alpha 1$, $\alpha 2$ are computed per racetrack. These are the HW0 line on Figure 2. If for instance, in the first nanowire, $\beta 10{=}\beta 11{=}\beta 12{=}\beta 13{=}0$ and $\beta 14{=}1$ then $\alpha 10{=}\alpha 11{=}0$ and $\alpha 12{=}1$. if $\beta 10{=}\beta 11{=}\beta 12{=}\beta 13{=}\beta 14{=}1$ then $\alpha 10{=}\alpha 12{=}1$ and $\alpha 11{=}0$ because five ones are seen in the beta.

Afterwards, the line parity and the column ECC are created. Each line parity bit is the parity of each HW0 line. The column ECC (Hamming code), as represented in figure 3 protects a column of HW0. In our example, 512 line parity bits (one for each nanowire) and three column ECC blocks ($\log_2(n){=}3$) are created. All these bits are stored, they are modified only if a cache line is written.

Once they are initialized, after a shift, all these bits are computed again, however, this time they are only compared with the one stored. If they are not identical then an error occurred.

If only one error occur the column ECC are enough to detect and correct the error. The line parity bits are really useful when more than one error is to occurred because then, they point to the lines with the faults. In the case of multiple errors, the column ECC will detect the faults but won't be able to do anything or even worse the ECC block could point to the wrong racetrack. However when combined with the line parity it's possible to know from which HW0 column the errors are and the other parity bits will precise which lines are wrong. This will be further discussed in section 6-1.

### 4.2 BLOCK OF NANOWIRES

The process described in the previous section (4-1) is effective to correct a misalignment by any number, or up to three misalignment by one, however, if four position errors in a block were to occur in a certain pattern then DEC could only detect the errors without correcting them. Our goal here, is not to be able to detect and correct everything but to guaranty a certain reliability.

In order to do that, we separated the HW0 block in smaller blocks, each of them protected as demonstrated in the previous paragraph. In our example, the cache line size is 512 bits,

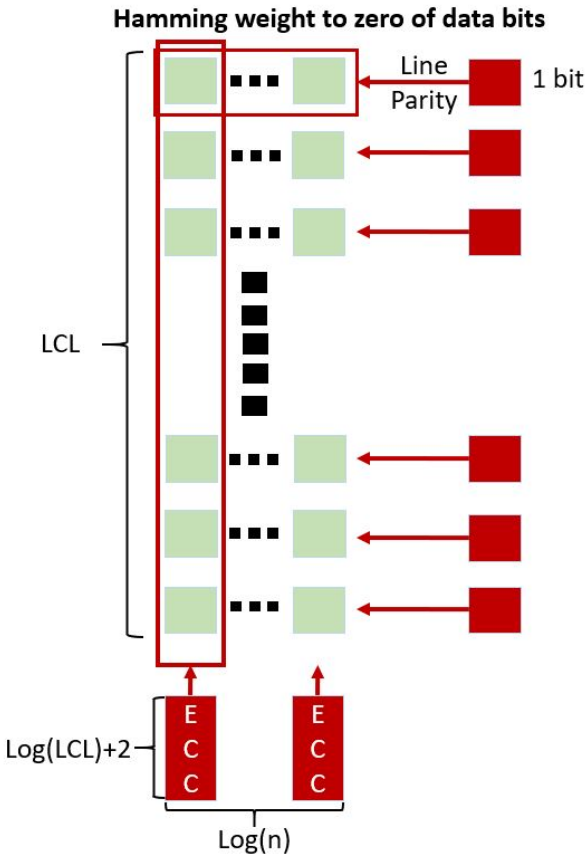


Figure 3: Hamming weight protection

before we had three column ECC blocks each of them protecting 512 bits. After dividing the blocks, in spite of one HW0 block of 512x3 bits for which 545 parity bits are stored (512+3*11), there will be two blocks of 256x3 bits each for which a total of 572 parity bits are stored (512+3*10+3*10). As a consequence there are now six column ECC blocks, each of them protecting 256 bits. The line parity bits are unchanged. It's possible to divide HW0 in more than two blocks, however more column ECC bits are needed. In the result section our scheme divides HW0 in eight blocks of 64x5 bits (n=32 => $\log_2(n)$=5). The goal behind this subdivision is to reduce the probability to get errors in the same block. The subdivision is a trade-off between area consumption and reliability.

### 4.3 GREY CODE FOR ONES COUNTING

According to table 1, while shifting the domains, if an error were to occur, most of the time it will be a misalignment by one domain in a way or the other (over or under-shift). Which implies, when counting the ones, the sum will be either one up or one down. Using normal binary counting this kind of error can modify several bits in HW0. That's why while counting ones, Gray code was used. It guaranties if an error by one domain occurs only one bit of HW0 will be modified. As a consequence only one column ECC will be affected by the error and the parity line will be able to detect the line. If two bits were to be modified then the line parity wouldn't detect it and two columns ECC would be affected. By decreasing the number of bits change it improves the reliability. Whatever the number of line misaligned by one domains, since only one bit per line is changed, the line parity will always be able to detect it.

### 4.4 LIMITED SHIFT

As seen in table 1, the probability of getting an error while shifting by seven is too high. A way to limit the number of errors without drastically hurting the performance is to limit the maximum number of shift effected at once. For instance, in spite of directly order a shift by six, we do two shifts by three. By sending two smaller currents, the probability of error is decreased. Moreover, even if an order of six shifts is made, as it will be described in the result section, only three shift can be done in a cycle. Therefore, two cycles will be needed. The same time is needed while doing two shifts of three if no checked is ordered in the intermediate state. However, if we wanted to reduce even more the probability of error and limit the shifting to two shift. Then in this example our performance would be affected because now, three cycles are needed to shift by six. Solution that we try to avoid.

Consequently, we used a shift limited policy, where for each shift, we try to not decrease the performance while decreasing the probability of error.

# 5.0 EXPERIMENTAL SETUP FOR TRANSVERSE READ

## 5.1 TRANSVERSE READ

The existence of a domain wall in a ferromagnetic nanowire can increase the resistance of it significantly when the difference between the spin-up and spin-down density of states is considerable. Therefore, a portion of the spin polarized electrons from one domain trying to pass through the next domain get reflected from the domain wall. Consequently, the resistance increases even though the reflection by the wall is minimal. Cabrera and Falicov [10] calculated the effect of back scattering due to a domain wall on the resistance by measuring reflection coefficient. However, the transmitted electrons will experience two different phenomena from the resistivity perspective : (i) in the wall region, the direction of the electron flow bends; and (ii) in the next domain, spin transfer torque (STT) evolves due to magnetization mismatch.

## 5.2 RESISTANCE PROFILE

We read the different bit patterns stored in magnetic nanostrip by considering the fact that the walls are static. Berger demonstrated that when a current passes through the wall, it exerts a force on the wall to move [11]. However, due to the presence of local strains, inhomogeneities and shape anisotropy, there exists a pining force ($F_p$) on the wall and can be calculated from :

$$\mid F_x^p \mid \leq 2H_c M_s A \tag{5.1}$$

For a static wall, where, $H_c$, $M_s$ and A are the coercive field, saturation magnetization and area of the wall. To ensure that the walls are in zero velocity, the current passing through it should be less than the critical current ($J_c$) necessary to move the wall.

$$J_c = \frac{2H_c}{a\beta} \tag{5.2}$$

$$\beta = \frac{M_s R_1}{\rho} \tag{5.3}$$

Where $a$ and $\beta$ are the wall spacing and the tangent of the Hall angle of the material. Assuming that the current flow is in +x direction while the magnetic domains are out of plane (+z or -z), the resistivity tensor can be expressed [11] as

$$\rho_1 = \begin{vmatrix} \rho & -\beta\rho\frac{M_{z1}}{M_s} & 0 \\ \beta\rho\frac{M_{z1}}{M_s} & \rho & 0 \\ 0 & 0 & \rho \end{vmatrix} \tag{5.4}$$

$$\rho_2 = \begin{vmatrix} \rho & \beta\rho\frac{M_{z2}}{M_s} & 0 \\ -\beta\rho\frac{M_{z2}}{M_s} & \rho & 0 \\ 0 & 0 & \rho \end{vmatrix} \tag{5.5}$$

where $\rho$, $M_{z1}$, $M_{z2}$ are the resistivity of the material and the magnetization of domains next to the wall. The resistance of the nanostrip due to the presence of domain walls is calculated as

$$R = R_0 + \Delta R = R_0(1 + \frac{K_c N_w t \beta^2}{L}) \tag{5.6}$$

where $K_c$ is material dependent, $L$,$t$ and $N_w$ are the length, thickness of the strip and the number of walls.

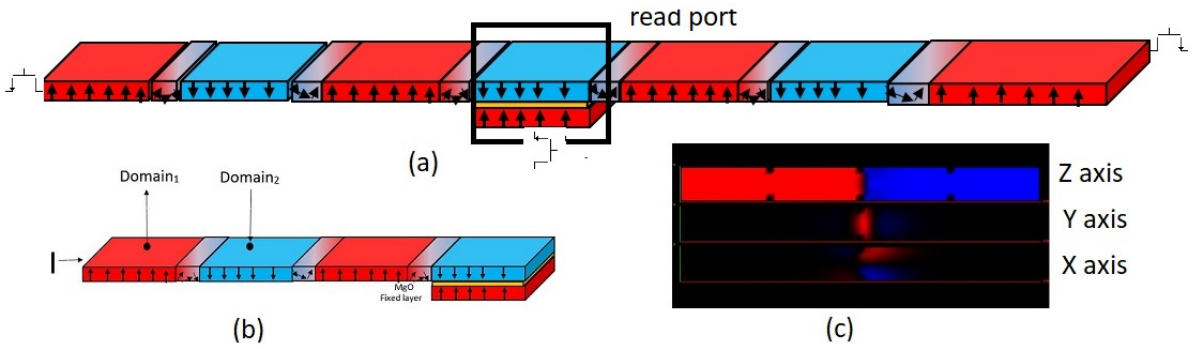


Figure 4: (a) Structure of the strip containing 7 bits ; (b) half of the strips used for simulation;(c) '1100' bit patterns stored in the strip.

### 5.3 SIMULATION FRAMEWORK

We considered a nanowire with perpendicular magnetic anisotropy (PMA) for high density and faster operation. The structure of the nanowire is depicted in Fig. 4a. This strip is symmetric in a sense that the read port is in the middle of the nanostrip and it can store 7 bits.

The reading process starts by turning the rightmost transistor off, and turning the leftmost transistor and the transistor connected to the read port on to ensure that the current flows from the left most domain to the read port. Fig. 4(b) shows the half of the strip used in the simulation for measuring resistances. We applied a read current I less than the critical current (see Eq. 5.2) for domain wall motion, and collected the current in the fixed layer of the read port. We calculated the resistance of the nanostrip for bit patterns by measuring the voltage difference between the leftmost domain and the fixed layer of the read port. Due to the symmetry, achieved similar results while reading from the right. We carried out the simulations by using LLG micromagnetic simulator, and used the parameters from [12] for a nanostrip with PMA. The resistance values are listed on average in table 2. These results are for a length of four bits, in this case the average is for the same number of one but in different configuration as: 1100 and 0101 both have two ones but their resistance isn't equal.

Table 2: Resistance during a transverse read of four racetrack bits for all data combinations.

| $X_3X_2X_1X_0$ | Resistance (Ω) |
|---|---|
| 0000 | 5.3k |
| 0001 | 6k |
| 0010 | 5.85k |
| 0100 | 5.8k |
| 1000 | 5.8k |
| 0011 | 6.6k |
| 0101 | 6.5k |
| 0110 | 6.4k |
| 1001 | 6.45k |
| 1010 | 6.35k |
| 1100 | 6.25k |
| 0111 | 7.2k |
| 1011 | 7.2k |
| 1101 | 7.15k |
| 1110 | 7.05k |
| 1111 | 7.55k |

## 6.0 ERROR PROBABILITY

In previous work, the correction code is applied at the granularity of one racetrack. In our case, we apply a correction code to an entire block of racetracks. In this section we will consider the probability of several racetracks simultaneously suffering from a position error. The probabilities are calculated for a cache line length (LCL) of 512 and data length per racetrack n=32.

## 6.1 ERROR PROBABILITY IN A BLOCK

When shifting an entire block, using the numbers from table 1, the probability of having two or more misalignments by one domain is significant enough that it must be considered. First, we calculated the likelihood of transitioning from a fault-free state to a faulty state in terms of the number of erroneous shifts. The equation used is:

$$P(o1, o2) = \binom{o1 + o2}{o2} * \binom{nn}{o1 + o2} * P_k(off1)^{o1} * P_k(off2)^{o2} * P_k(nf)^{(nn-o1-o2)} \quad (6.1)$$

Where o1 represents the number of racetrack misaligned by one domain, o2 the number of racetrack misaligned by 2 domains, nn the number of nanowires per block (nn=LCL if the HW0 block is not divided), $P_k$(off1) the probability of being misaligned by one domain while shifting by k, and $P_k$(off2) the probability of being misaligned by two domains while shifting by k.

The probability of a misalignment by three domains is too small to be considered. $P_k$(nf) is

the probability of having no faults while shifting by k. $P_k$(off2) and $P_k$(off1) are as described in table 1.

$$P_k(nf) = 1 - P_k(off1) - P_k(off2) \tag{6.2}$$

Presuming we are in one of these error states, we calculated the probability of detection and correction. These numbers are tied to the racetrack length; the bigger n is, the more column ECC blocks are used. Thus, more errors can be corrected. If in DEC only vertical ECC and Gray code were used, then the maximum number of errors that could be corrected would be $\log_2$(n) misalignments by one (n is the data length of a racetrack), or $\log_2$(n)/2 misalignments by two. This result is explained by the fact that using the grey code, only one bit will change (if misaligned by one), thus each vertical ECC can correct one error, in the case where each error appears in a different column of HW0. Using only column ECC (CECC), if more than two errors occur in the same column, then the line parity would be necessary to correct it. CECC alone would detect only a single error in this case, and attribute it to the wrong position. For this reason, the line parity bit must be included to provide the reliability as described in section 4.

### 6.2 HW0 DIVISION

If four errors by one occur in a block, only certain patterns are guaranteed to be correctable. If the errors occur in different lines, then the errors can be corrected as described above. If two errors occur in one column and two in another column, because the line parity bits are shared, it is impossible to determine precisely which bit in HW0 is wrong and so it's impossible to characterize the errors as over-shift or under-shift. We address this by cutting in half the hamming weight to zero block (HW0) and doubling the number of CECC blocks. This will decrease the probability of errors in the same block, because we keep the same overall fault rate, but the number of blocks increases.

Figure 5 shows the additional ECC bits needed when dividing the HW0 block. In the results section, we decided to divide the HW0 block into eight sub-blocks. DWM is a new

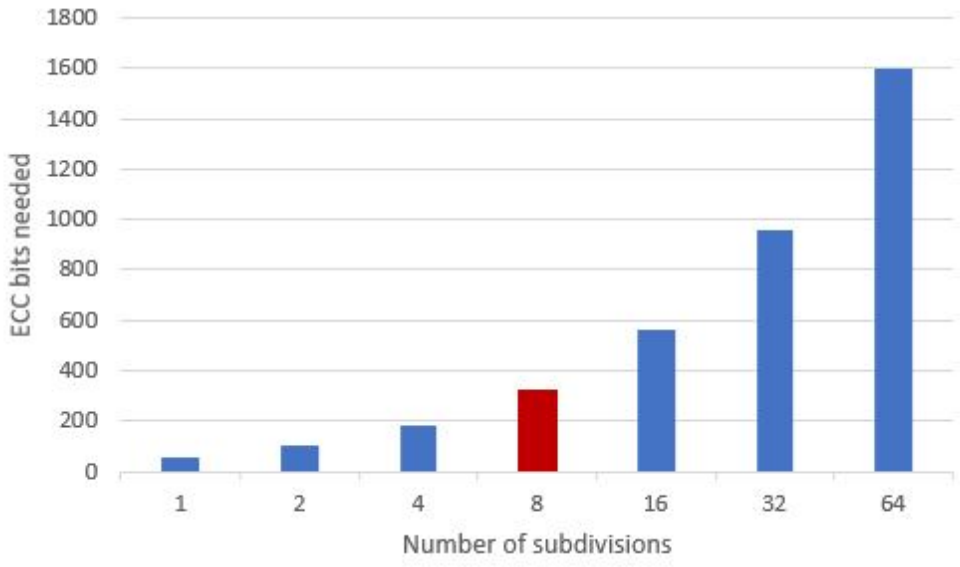


Figure 5: ECC bits needed when subdividing HW0 into different sizes

technology and in the future we can hope that IBM improves this technology, reducing the probability of position errors. If the probability of shifting drops, it's possible to consider no division of HW0, which would reduce the number of cells needed to store the CECC bits by approximately six times.

# 7.0 RESULTS

In this section, four characteristics will be evaluated: reliability, area overhead, power consumption, and finally performance. The simulations are realized using the simulator Sniper. All the benchmarks used to profile the performances are workloads from SPECCPU2006.

All the simulations were launched with the parameters describe in table 3. Furthermore, we simulated only one read/write head per line.

Table 3: Latency and Energy Parameters by [2]

| | Fused cache |
|---|---|
| Tag access latency (ns) | 0.28 |
| Data read latency (ns) | 0.98 |
| Data write latency (ns) | 0.65 |
| Read energy (pJ/access*) | 19.08 |
| Write energy (pJ/access) | 7.94 |
| Leakage power (mW) | 19.40 |
| Tag area ($mm^2$) | 0.045 |
| Data area ($mm^2$) | 0.0491 |
| Total area ($mm^2$) | 0.0941 |
| Shift latency (ns) | 0.322 |
| Shift energy (pJ/cell) | 0.0617 |

*Each access contains one line, or 512 bits/cells

### 7.1 RELIABILITY

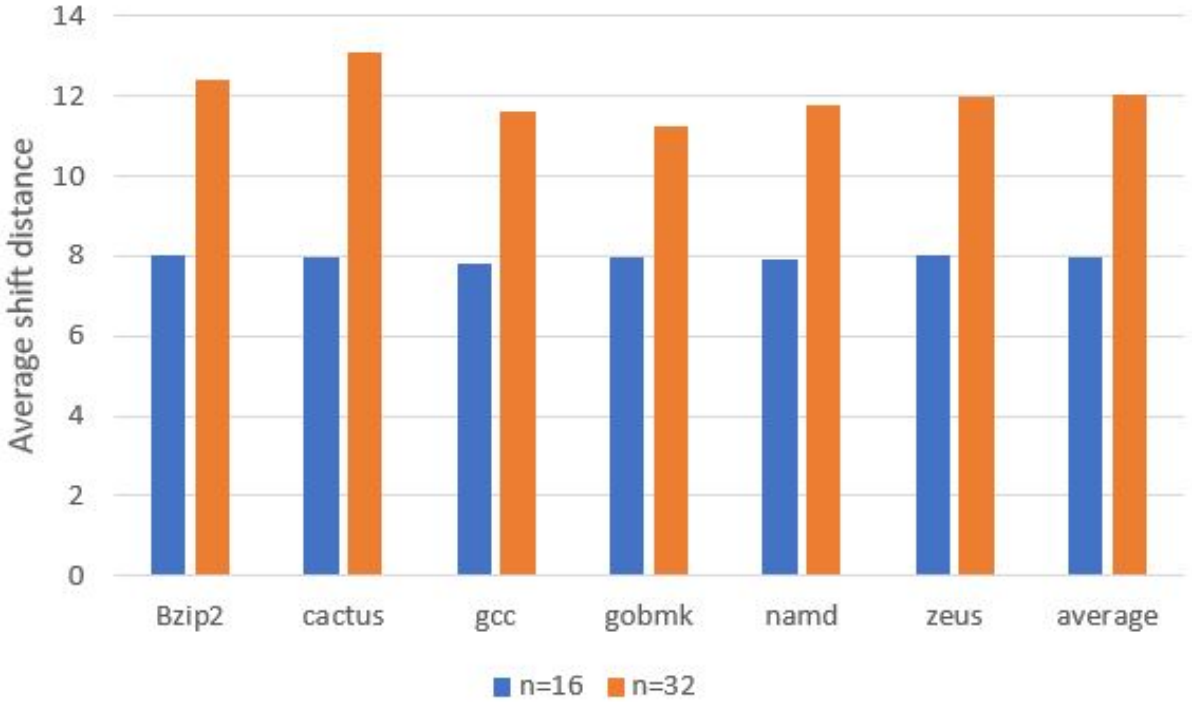


Figure 6: Average Shift Distance in L2 for Different SPEC 2006 benchmarks.

Figure 6 represents the average shift distance in L2 cache for different lengths of racetrack (n=16 and n=32). According to this figure, to get the best performance the system should be allowed to shift by distances greater than eight. However, according to table 1 shifting twice by two is not equivalent to shifting once by four from a reliability perspective.

In Figure 7 we shows the detectable, unrecoverable errors mean time to failure (DUE MTTF) of our scheme using various shift lengths. In order to meet the DUE MTTF target fixed by previous research [**?**] at 10-years, we need to make a trade-off between reliability and performance. Each DEC-i represents the MTTF if the system can only shift in increments of i domains. The horizontal line represents the 10 years goal. As a consequence, in order to meet this goal, the system can't shift by more than three domains at once. Under this condition, the different MTTFs are: 1780, 160 and 13 years on average across the benchmarks.

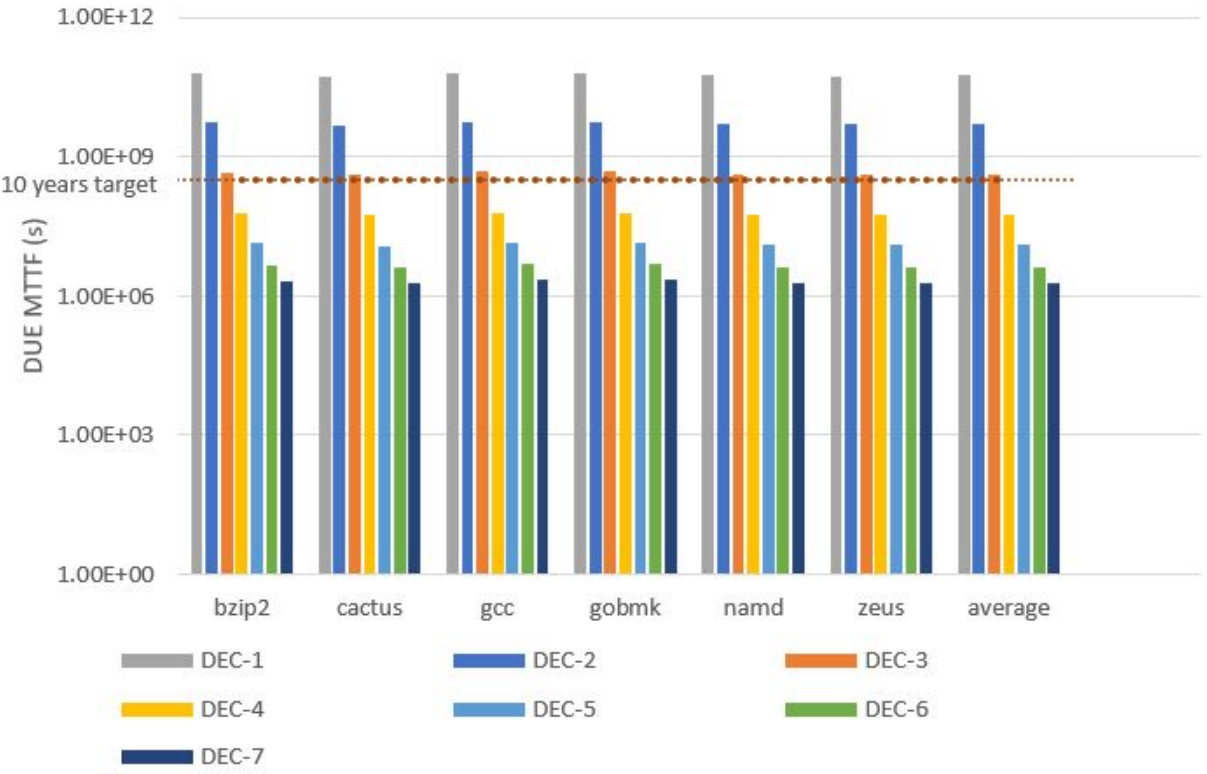


Figure 7: DUE MTTF

## 7.2 DIMENSION COMPARISON

In DWM having extra domains on the sides of the data domains is a necessity to preserve the data integrity. However, additional heads on these domains are sources of area consumption, reducing the density of the racetrack. By adding fewer STT-MRAM cells in spite of these heads we limit the area overhead needed to protect the racetrack under DEC.

In p-ECC-O [1] four extra heads (two to read/write and two read-only) are added (two on each side). Thus, the number of additional heads is four per racetrack. Despite using only two extra read heads per racetrack, because of the additional domains that p-ECC needs, this algorithm consumes more area than p-ECC-O if the racetrack length exceeds 16. In this article, figure 8 is obtained using the data from [12] as a reference for the cell area and sensing circuit. In figure 8, we compare the impact on area per bit using four

different techniques: HW0, DEC64, p-ECC and p-ECC-O. HW0 is included in this figure to demonstrate the utility of our scheme in comparison with just storing the derived data. As we can see, for both n=16 and n=32 our scheme decreases the area overhead (by 2.6 and 3.7 respectively versus p-ECC).

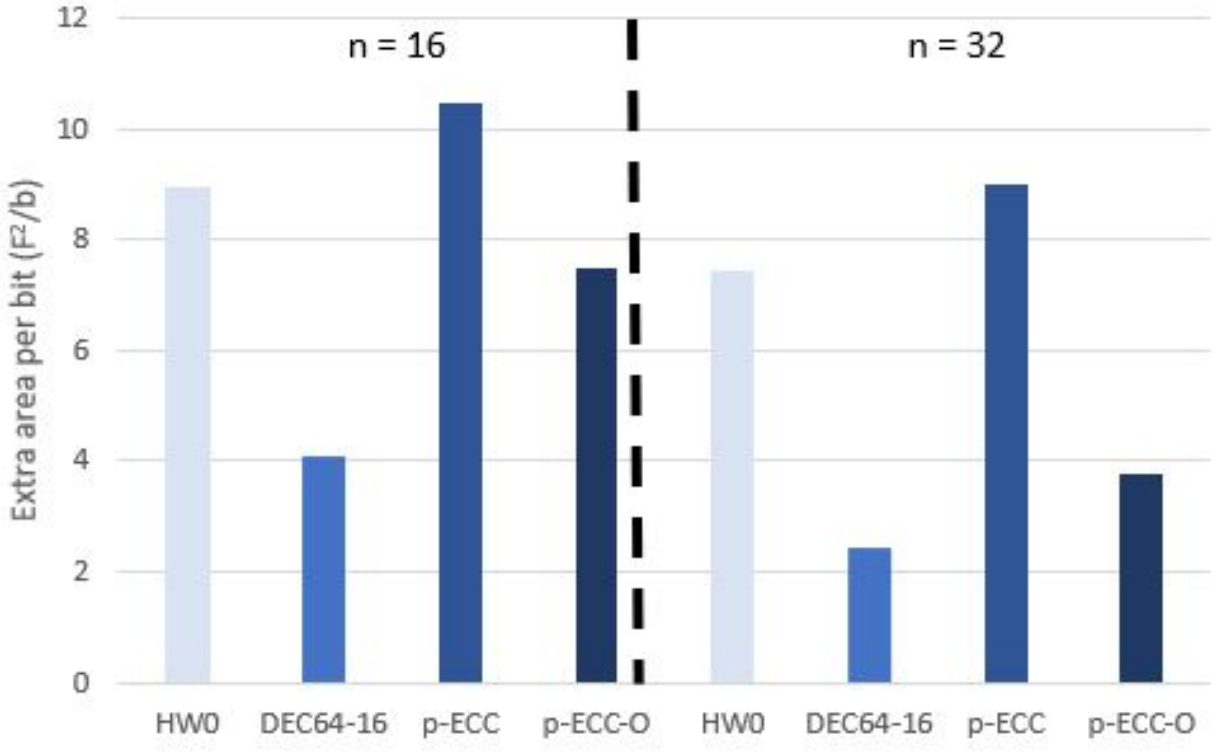


Figure 8: Area overhead

### 7.3 POWER CONSUMPTION

According to our simulation (section V-C), a transverse read consumes 1.67x more energy than a normal read. As seen in section III, to compute DEC, two transverse reads and a normal read are required after every shift. Furthermore, to check if an error occurred, all the STT-MRAM cells need to be read; DEC64 for n=32 requires 832 cells and DEC64 for n=16 requires 704 cells. In comparison, p-ECC-O shifts only by one domain at a time to

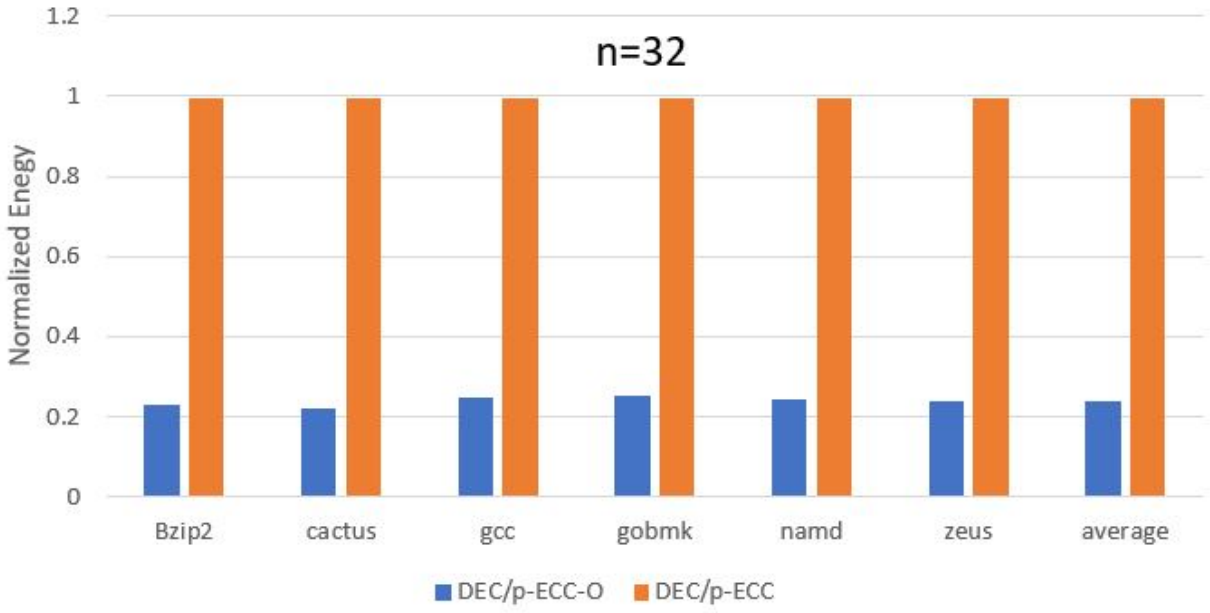


Figure 9: Power consumption: DEC64-32/p-ECC-O, DEC64-32/p-ECC

guarantee the integrity of the system. However, this characteristic has a huge impact on the power consumption and performance because heads are repeatably used. On figure 9 we compare our scheme with both of their schemes. On average the power consumed by DEC64 for n=32 is equivalent of the energy consumed by p-ECC and 20% of p-ECC-O.

## 7.4 PERFORMANCE EVALUATION

As discussed previously, the transverse reads are expected to negatively impact the performance. Compared to other techniques were the reads can be done in parallel, transverse reads need to be done in series with the normal read. In order to decrease this impact, the system has to first read the normal way, then carry out the transverse reads. This enables the read to complete quickly if there is no fault, reducing average access time. If the block is not accessed before the extra reads and the comparison with DEC complete, then our scheme

is perceived has a normal read, which doesn't impact the performance. Thanks to that this algorithm performs as seen on figure 10. On average DEC64-32 incurs a performance overhead of 1.5%.

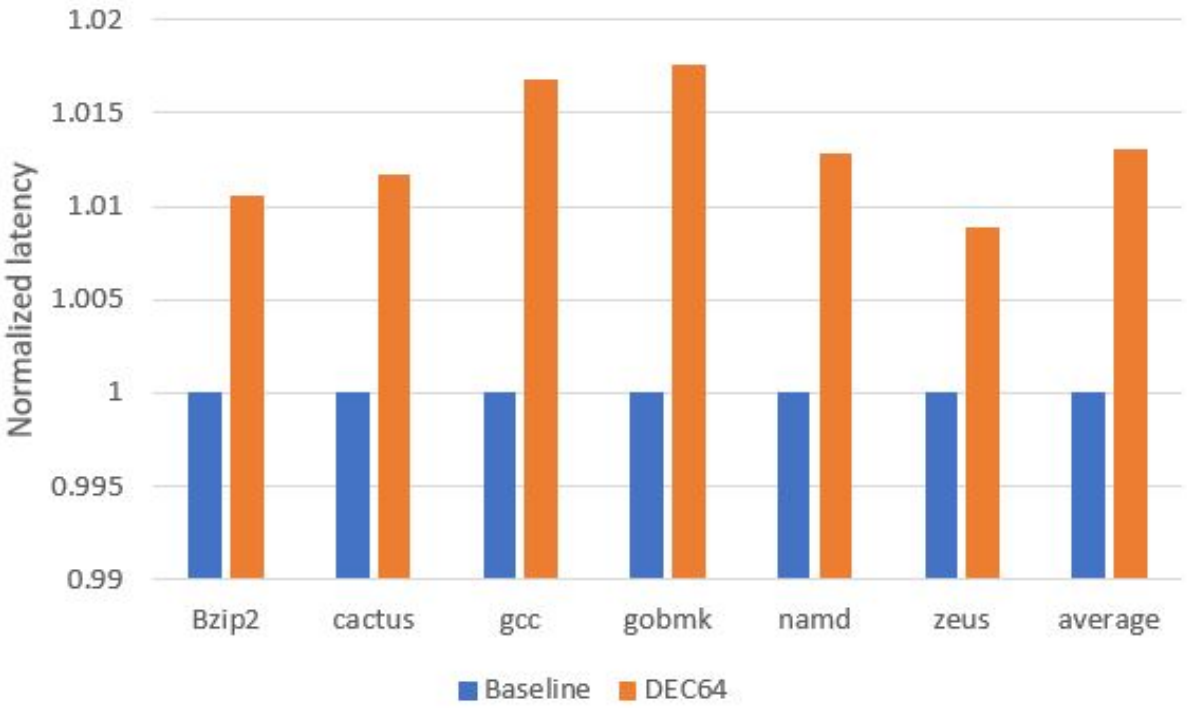


Figure 10: Latency overhead

# 8.0 CONCLUSIONS

In domain wall memory prototypes, the extremely high probability of positional shift errors can discourage its potential use despite its area and power advantages compared to other traditional and emerging memories. To overcome this reliability limitation and allow the intrinsic benefits of domain-wall memory to shine, we presented DEC, an efficient correction code which is original in its methodology, considering its capability to correct errors in the primary data by only protecting a smaller value calculated from the original data. In order to calculate this derived information, our novel reading technique, transverse reading, is used to calculate the number of ones in a nanowire. Using these concepts we reach an DUE MTTF above the target. In addition, our scheme preserves the area advantage of DWM by decreasing the area overhead by 2.6 and 3.7 times against state of the art technique.

## 8.1 FUTURE WORK

While DEC can correct shifting errors, DWM is also subject to pinning errors. Pinning errors cause the racetrack to lose or duplicate bits. This can modify data far from the read head. As a consequence, pinning errors are extremely difficult to detect and correct. Another direction for future work is to explore the possibilities that DEC offers in instances other than DWM.

## BIBLIOGRAPHY


[1] C. Zhang, G. Sun, X. Zhang, W. Zhang, W. Zhao, T. Wang, Y. Liang, Y. Liu, Y. Wang, and J. Shu, "Hi-fi playback: Tolerating position errors in shift operations of racetrack memory," *ACM SIGARCH Computer Architecture News*, Vol. 43, pp. 694–706, ACM, 2015.

[2] H. Xu, Y. Alkabani, R. Melhem, and A. K. Jones, "FusedCache: A naturally inclusive, racetrack memory, dual-level private cache," *IEEE Transactions on Multi-Scale Computing Systems*, Vol. 2, No. 2, No. 2, pp. 69–82, 2016.

[3] Y. Huai, "Spin-Transfer Torque MRAM (STT-MRAM): Challenges and Prospects,"

[4] G. S. W. Z. F. M. H. L. W. Zhao, C. Zhang, "Quantitative modeling of racetrack memory, a tradeoff among area, performance, and power," *Design Automation Conference*, 2015.

[5] S. S. P. Parkin, M. Hayashi, and L. Thomas, "Magnetic Domain-Wall Racetrack Memory," *Science*, Vol. 320, No. 5874, pp. 190–194, Apr. 2008.

[6] A. V. V. K. V. Yeow Meng Chee, Han Mao Kiah and E. Yaakobi, "Codes Correcting Limited-Shift Errors in Racetrack Memories," *IEEE International Symposium on Information Theory*, 2018.

[7] D. Kline, H. Xu, R. Melhem, and A. K. Jones, "Racetrack Queues for Extremely Low-Energy FIFOs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, No. 99, No. 99, pp. 1–14, 2018.

[8] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "TapeCache: a High Density, Energy Efficient Cache based on Domain Wall Memory," *Proc. of ISLPED)*, pp. 185–190, 2012.

[9] Q. Hu, G. Sun, J. Shu, and C. Zhang, "Exploring main memory design based on racetrack memory technology," *Proceedings of the 26th edition on Great Lakes Symposium on VLSI*, pp. 397–402, ACM, 2016.

[10] G. Cabrera and L. Falicov, "Theory of the residual resistivity of Bloch walls I. Paramagnetic effects," *physica status solidi (b)*, Vol. 61, No. 2, No. 2, pp. 539–549, 1974.

[11] L. Berger, "Prediction of a domain-drag effect in uniaxial, non-compensated, ferromagnetic metals," *Journal of Physics and Chemistry of Solids*, Vol. 35, No. 8, No. 8, pp. 947–956, 1974.

[12] W. Zhao, Y. Zhang, H. Trinh, J. Klein, C. Chappert, R. Mantovan, A. Lamperti, R. Cowburn, T. Trypiniotis, M. Klaui, *et al.*, "Magnetic domain-wall racetrack memory for high density and fast data storage," *Solid-State and Integrated Circuit Technology (ICSICT), 2012 IEEE 11th International Conference on*, pp. 1–4, IEEE, 2012.