A Real-Time Error Detection (RTD) Architecture and its use for Reliability and Post-Silicon Validation for F/F based Memory Arrays

Yiannakis Sazeides, Arkady Bramnik, Ron Gabor and Ramon Canal

Abstract—This work proposes in-situ Real-Time Error Detection (RTD): embedding hardware in a memory array for detecting a fault in the array when it occurs, rather than when it is read. RTD breaks the serialization between data access and error detection and, thus, it can speed-up the access-time of arrays that use in-line error-detection and correction. The approach can also reduce the time needed to root-cause array related bugs during post-silicon validation and product testing. The paper presents how to build RTD into a memory array with flip-flops to track in real-time the column-parity and introduces a two-dimensional Error-Correction scheme based on RTD. As compared to SECDED, the evaluated scheme has comparable error detection and correction strength and, depending on the array dimensions, the access time is reduced by 8% to 24% at an area and power overhead between 12% to 53% and 21% to 42% respectively.

Index Terms— Reliability, Testing, and Fault-Tolerance, Memory design, Error-checking

1 INTRODUCTION

Error detection and error detection and correction codes [1] are widely used to protect the data in memory arrays of electronic devices from errors. Typically, a coding technique adds one or more parity bits to each word in an array to encode redundant information about the stored data. When a codeword (data plus parity) is read from the array, the value of the parity is calculated from the data and an error is detected if the calculated parity does not match the parity read from the array and if the code strength permits it, the error is corrected.

Coding schemes are used to protect the data in arrays from various errors in-the-field, e.g., soft-errors (SER) [2]. Naturally, the error protection of an array is also useful during manufacturing tests [3] to expedite the detection of defects in the array, as well as during post-silicon validation for the identification of bugs that manifest as corruption in codewords read from the array.

In this paper, we propose in-situ real-time error detection (RTD), an error protection approach that can detect a fault in a memory array when it happens, rather than when the faulty value is read. At a high level, what RTD does is to calculate in real-time what the parity of all codewords in an array are, and check them all the time against the parity of the codewords produced when the codewords were written in the array. Essentially, RTD can detect a fault instantaneously after it occurs, whereas other coding-based protection techniques, collectively referred to as non-Real-Time Error Detection (nRTD), detect the fault only after the stored data is read.

RTD has practical uses in reliability and post-silicon validation. For reliability [4], RTD can be used to speed-up array accesses for arrays required to provide in-line error-detection and correction. For post-silicon validation [5], RTD can be very effective in reducing the time needed to root-cause bugs that manifest as array-content corruptions for both test and production chips.

The paper explains the RTD's functionality and shows how to integrate RTD in an array built with flip-flops (F/F) to track in real-time its column-parity. We also present a two-dimensional (2D) ECC scheme based on RTD. A comparison of the 2D ECC RTD design against traditional (nRTD) SECDED reveals that adding RTD to an array provides a significant access time reduction albeit with an area and power overhead.

In the remaining of the paper, we discuss the RTD Architecture (Section II), a RTD 2D ECC scheme (Section III), an implementation of RTD for a F/F array (Section IV), an evaluation of RTD overheads (Section V) and its code strength (Section VI), RTD use for post-silicon validation (Section VII), related work (Section VIII), and conclusions (Section IX).

- *This work extends the "2D Error Correction for F/F based Arrays using In-Situ Real-Time Error Detection (RTD)" in IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, October 2020 [24].*

- *Y. Sazeides is with the Department of Computer Science, University of Cyprus, 1 Panepistimiou Avenue, 2109 Aglantzia, Nicosia, Cyprus. E-mail: yanos@cs.ucy.ac.cy*
- *A. Bramnik is with the Intel Israel Development Center, M.T.M. Scientific Industries Center, Haifa 31015, Israel. E-mail: arkady.bramnik@intel.com*
- *R. Gabor is with the, Israel. E-mail: ron.gabor@gmail.com*
- *R. Canal is with the Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, c/ Jordi Girona 1-3, 08034 Barcelona, Spain, E-mail: rcanal@upc.edu*

2 RTD Architecture

In this section, we illustrate the high-level functionality of a memory array protected with RTD. In Section XYZ we will discuss an implementation of RTD that shows how to actually embed RTD inside an array.

Fig. 1.a shows a baseline array without error protection that contains R rows, C columns, a read port to output (OUT) the value at the read-address, and one write port to store the input (IN) at the write-address. Fig 1.b presents how to extend the baseline design with traditional nRTD protection by adding parity bits per row. The parity is generated (using a parity Generator denoted by G in Fig. 1.b) on a write cycle according to the code used and the value that is getting stored. The generated codeword (data+parity) is then written in the array. On a read cycle, a codeword is read from a row and an error is detected with the help of a codeword checker (C).

Fig. 1.c introduces the extra array interfaces and functionality needed by RTD to detect in real-time whether the array contains a fault. Specifically, RTD requires having in-situ (i.e. built in the array) a port to track the real-time-column parity (RTCP) of all cell values per array column. This port does not need an address decoder, to select a specific row, since it produces the xor of the values in all cells per column. Additionally, RTD requires a SCP (stored-column-parity) register with C bits (as many as the array columns). This register maintains the parity for each column and it is updated on every array write cycle with the biwise-xor of the current value in SCP, the previous data (PD) in the row that is written and the new value to be stored (IN). An error signal vector (EV), C bits wide, is produced using the bitwise-xor of the SCP and RTCP. Anytime there is a mismatch at the same bit position between SCP and RTCP, the corresponding bit in the EV is asserted to flag the presence of a fault in the corresponding column (or in the corresponding SCP position).

The key property of RTD is that it can detect a fault as soon as it occurs (without first reading the entry that contains the fault). Put it another way, the RTD in-situ hardware helps break the dependence between data access and error detection. This RTD characteristic opens a new venue for enhancing reliability and post-silicon validation techniques. These include among other speeding-up the access for arrays that require in-line error detection and correction and reducing the time to root-cause bugs. We discuss these and other optimizations enabled by RTD in the remaining paper.

2.1 RTD Attributes

The RTD architecture presented in Fig. 1.c is for an array without any other protection besides RTCP and is capable of only of error detection. As we show next (Section III), RTD can be combined with other protection, for example traditional row parity, and enable fast error correction.

RTD protection requires to maintain correctly the SCP. For arrays that are updated at boot time (e.g., patch arrays [6]) and are read-only thereafter, the SCP requires no additional maintenance after it is written at boot time with the bitwise xor of the column values that the array is initialized with. For RTD arrays that are read-write, i.e., their content can be modified during run-time, maintaining correctly the SCP requires to read the previous data (PD) from the address to be overwritten on a write cycle. If the array is rarely written, one can use the regular array read port to read the PD before each write. However, if a read before a write is detrimental to performance, one can introduce an additional read port dedicated to read the PD in the row to be overwritten. The design of various RTD variations as well as their speed vs area trade-off compared to nRTD are explored in Sections IV and V.

The error detection strength of RTD in Fig. 1.c is an odd number of faults in each column, i.e. RTD can detect which columns contain an odd number of faults. If it is desirable to detect a burst of vertical errors in a column, vertical logical-interleaving can be used [8][9]. For instance, to detect any burst of two consecutive vertical errors we need to use RTCP with 2-way vertical interleaving that tracks separately the RTCP for even and odd rows. Additionally, two separate SCP registers, each with C bits, need to maintain separately the parity of even rows and odd rows. On a write and read cycles, additional logic (not shown in Fig. 1.c due to space constraints) will control which RTCP to use and SCP to update depending on whether an even or odd row is accessed. Section VI investigates the SER reliability benefits of using RTCP with vertical logical-interleaving.

Even though the focus of this work is in using RTD to provide the column real-time parity, the approach can be used to track the real-time parity of rows, of both rows and columns or even some other cell combination. In Section XYZ we present a design that provides error detection and correction using RTD for both rows and columns.

One other requirement for arrays protected by RTD is to set the SCP after initialization according to the array initial content. For read only arrays this is straightforward as the initial contant is written in the array at boot and, therefore, it is known and can be used to determine the value to intialize the SCP with. For general read-write arrays, with unkown content after initialization, one can employ a special flow to initialize the array and SCP to a deterministc state or simply write after initialization the RTCP output into the SCP.

3 RTD Use Case 1: 2D ECC

In this section, we present a 2D in-line ECC scheme based on RTD. In-line ECC checks/corrects the data before forwarded for use, i.e. the ECC lies in the read critical-path. Unless stated otherwise, to simplify the discussion, we assume that at any given time any faults are present only in one array row. We consider in detail the implications of faults in multiple rows in Section VI.

The scheme shown in Fig. 2 combines RTD and nRTD in a 2D fashion to provide error-correction. The RTD is used to track the array's column parity (as presented in Section II) and produce in real-time the EV that indicates the array columns that contain a fault. A conventional (nRTD) parity is used to detect faults in the data of a row after it is read. The row parity is generated (by a generator denoted by G in Fig. 2) and stored in a row together with the data in a write cycle. On a read cycle, a checker (C1) is used to check if the parity of the accessed data and the corresponding row parity match.

If during a read cycle the row parity status is no-error (NE) then the read data is forwarded to the output (OUT) as is (bitwise-xored with a zero correction vector (CV for data@raddress)). Otherwise, there is a row parity mismatch and, since this is the only row with faults (assumption that a single row can have faults), the CV is set equal to the EV and used to flip and repair the data bits in the column positions with errors (i.e., a correctable error (CE)).

For read-write arrays to maintain the SCP correctly, we need to handle carefully the case when an entry with a fault is overwritten. Specifically, in the bit positions that the entry has errors we need to flip them before using the data to compute the new SCP. This is accomplished with an extra checker (C2) that during a write cycle it checks if there is an error in the PD (the data to be overwritten) and produces a CV (CV for data@waddress) to calculate the SCP. Recall (see Section 2) that SCP does not need maintainance for read-only arrays.

A single bit row-parity is unable to detect an even number of errors in a row. If an entry with even number of faults is read, the row-parity will not detect an error and it will forward the data with faulty bits. To prevent such silent-data-corruption (SDC), the decoder (D in Fig. 2) can monitor the EV and trigger a DUE (detected-unrecoverable-error) when an even (non-zero) number of bits are set in the EV.

Based on the above, the behavior of the decoder (D) of our scheme can be defined in terms of the row parity status (1 indicating an error, 0 no-error and X don't care) and the number of 1's in the EV (0 for zero, o for odd and e for even but non-zero) as follows:

TABLE 1
2D ECC+RTD DECODER

Row Parity Status	0	0	1	1	X
Number of 1s in EV	0	o	0	o	e
Decoder Output (D)	NE	NE	DUE	CE	DUE

The first four columns define the behavior during a read cycle. The 0-0 occurs when there is neither a row-parity mismatch nor an error detected by RTD in any column. The 0-o happens when the row contains no error but there is an odd number of columns with errors in another array row. For both of these cases there is no error in the data that is read. The 1-0 means that the row-parity indicates an error, in the data, but the RTD does not flag any error in any column. This can occur when an even number of faults occur across rows in a column and should raise a DUE. Such event cannot occur when faults are limited in at most one array row. The 1-o scenario happens when the row-parity indicates an error and the number of columns with faults is odd. In this case, the error is corrected according to EV. Finally, anytime we detect an even number of columns with error we trigger a DUE. This avoids the SDC when a row with even number of faults is read since the row-parity is unable to detect an even number of errors in the row.

A. RTD with Horizontal Interleaving

The DUEs caused by a burst of even errors in a row can be turned to a CE by employing horizontal logical-interleaving with degree equal to the burst length [9]. For instance, for a two-bit error burst horizontal logical interleaving will employ two parity bits per row one for the bits in even positions and the other for bits in odd positions. On a read access, two checkers will produce separate parity status for the even and odd bit positions. Two separate count of 1's in the EV are used, one for the even positions and another for the odd. The Decoder functionality for such scheme is defined in terms of the even and odd parity status and the number of 1's in the odd and even EV bit positions as follows (o indicates odd number of 1s, e even but not zero, and y 0 or odd and X don't care):

TABLE 2
2D ECC+RTD+2-WAY HORIZONTAL INTERLEAVING DECODER

Even Par. Stat.	0	X	0	1	1	1	X	X
Odd Par. St.	0	1	1	X	0	1	X	X
# of 1s in even EV bits	y	X	y	0	o	o	X	e
# of 1s in odd EV bits	y	0	o	X	y	o	e	X
Decoder (D)	NE	DUE	CE	DUE	CE	CE	DUE	DUE

The behavior is similar to the one without interleaving. DUE is triggered when either or both the number of even or odd columns with error is even and when a partition has an error but its corresponding EV count is 0. Otherwise, any error is correctable, even when there is an error in both the even and odd partitions. More detail implications of using logical-interleaving (both vertical and horizontal) are discussed in Section VI.

4 RTD USE CASE 2: REDUCE DUE

RTD can be used to avoid DUEs for the cases that error recovery is not viable when an array corruption is detected. This is the case for arrays that in-line error correction is not feasible (e.g., due to tight timing constraints) and error recovery undesirable (e.g., due to complex clean up process). One may opt in such situation to protect the array with RTD and upon error detection pro-actively either i) halt-exeuction and try to repair the array using demand-scrubbing [7] or ii) transfer execution to a different core (if the error is within the boundaries of a core and the transfer process does not entail the use of the corrupted state).

A demand-scrubbing upon error detection is possible when array uses RTD that tracks parity per column, and a nRTD EDC code per row, is suitable for a correction procedure inspired by the 2D ECC work [8] and CPPC [10] . More specifically, when RTD detects an error, the array is scrubbed by reading each of its rows and checking them for errors. If a row has no error, its content is XORed with a *register* as wide as the row, which is initialized to zero before the scrubbing starts. When all rows are scrubbed, and only a single row is detected with error (note such row's content is not XORed in the *register*), the row is corrected using the value produced by XORing the *register* with the column parity tracked by RTD. If more than one row has an error, a DUE is raised. A DUE is also raised if the RTD indicates an error, but scrubbing per row does not (this can occur when two bits in the same row have flipped). RTD key difference from CPPC: is that the error is detected before it is read and error detection is out-of-band does not lie in the read critical path.

5 RTD Applicability and Implementation

RTD is applicable to arrays built with different types of cells such as SRAM, CAM, latches or F/F. In this paper, we show how to implement RTD for a F/F based array. Such arrays are popular in modern CPUs [11]. F/F arrays with size up to a few thousand bits are known to offer area and power advantages over equal-size SRAM-based arrays [12][13]. While F/F cells are larger than SRAM cells, SRAM arrays have large overheads due to peripheral circuitry, e.g., sense-amps and pre-chargers. Consequently, F/F based arrays are attractive candidates for inclusion in products, and novel techniques for error-protecting and debugging them, as in this work, are of practical value.

An RTD implementation for a latch-based array is similar to the one with F/Fs and we do not present it due to space limitations. RTD is applicable to SRAM and CAM arrays but it requires using modified cells with extra port(s) to facilitate RTD. Developing and analyzing such SRAM and CAM cell designs represents an interesting direction for future work.

5.1 2D ECC RTD Implementation for a F/F based Array

Our implementation of RTD is based on the F/F array design proposed in [13]. This design is bit-slice based. Each bit-slice contains a column of F/Fs and a column with a multiplexer tree -for faster read latency- that is used to read one of the cells out according to which bit-slice row is selected. The logic design of a bit-slice with 8 F/Fs with 1 read and 1 write port is presented in Fig. 3. An array will consist of many bit-slices that share the read and write address decoders for selecting which row to read and write.

Before presenting the RTD implementation, we first show in Fig. 4 (left) a design for a traditional (nRTD) in-line SECDED build using the bit-slice in Fig. 3. The design assumes 4-bits of data per row and, therefore, requires four parity bits to provide SECDED protection [1]. The figure also shows that the error detection and correction is realized through a checker and a decoder [14]. The checker produces a syndrome that is decoded to determine, in the case the error is correctable, the 1-hot encoding of the bit-position with the error. This error-vector is bitwise-xored with the data to correct the error.

The RTD implementation of the 2D ECC in Sec. **Error! Reference source not found.** is shown in Fig. 4 (right). It introduces in-situ, built in the bit-slice a column that determines the RTCP of the F/Fs in the bit-slice. This is identical to the mux column in Fig. 3 but using xor gates instead of muxes. The design in Fig. 4 also includes an extra mux column to read the PD (the data to be overwritten on a write). This column is not necessary for read only arrays and arrays where performance is not hurt when performing a read before a write. In Section XYZ we will evaluate both RTD designs with and without the extra mux column.

The total number of bit-slices in Fig. 4 are five, four for the data and one for the row-parity. On a read cycle, the data from the selected row are checked for error using the row-parity. In the case of an error, the data are xored with the EV produced by xoring the SCP and RTCP (as in Fig. 2). Note that Fig. 4, for readability, only shows the design used during a read cycle.

The example in Fig. 4 helps highlight the trade-offs presented by RTD. It requires fewer but wider bit-slices and instead of a SECDED checker and syndrome decoder, it only needs a parity-tree. To assess the benefits and overheads of RTD based 2D ECC, we will compare experimentally its delay, area and power against SECDED ECC as well as the error correction and detection strength for various fault patterns in Section XYZ.

5.2 2D ECC RTD Implementation using both Real-Time Column and Real-Time-Row Parity

In this Section we present an array design that uses RTD to track both the real-time parity per column and per row is shown in Fig. 5. This design requires a port to provide the real-time-parity for each row (RTRP) in addition to the RTCP. It also needs to maintain a register with the stored row parity (SRP) in addition to the SCP.

The output, when reading from such array the data in row ***i***, is the result of the bitwise XOR of the data in the row (***di,0-di,3***) and the array's column-correction-vector (***ccv0-ccv3***). The ***ccv*** is all zero, and the output is the same as the data in the row, when the row's real-time error signal (***ri.err***) is zero, i.e. RTD does not detect an error in row ***i***. When RTD indicates that there is an error in the row that is read (***ri.err***=1), the ***ccv*** is set equal to the array's column real-time-error register (***c0.err-c3.err***) that identifies in real-time which columns contain a corruption. When the array contains a single row with a single data bit corrupted, the proposed scheme will detect and correct the corruption when the row is read by inverting the corrupted bit.

On a write in a row whose corresponding row RTD error signal indicates it has an error, the array's column real-time error register will be reset and the expected column parity, for the column that contains the error, will be calculated using the inverse of the value stored in the corrupted cell (logic is not shown in Fig. 5).

A fault in a cell that contains a row's expected parity can be ignored as the row data will be XORed with a column-correction-vector that does not indicate any data error and, therefore, all row data will be XORed with a zero. Similarly, we can ignore an error in a cell holding the expected parity of a column, because it will not set a row error signal and, consequently, all row data will be XORed with a zero ***ccv*** again. Finally, an error in a cell in the arrays' column error register will be inconsequential, since again no row data error is signaled and no data bit gets inverted.

As mentioned earlier, logical or physical interleaving can be employed [8][9] to enable this design to detect and correct multiple errors.

This alternative 2D ECC RTD scheme has clearly higher overhead as compared to the design in Fig. 4 (extra port for RTRP and maintain SRP register) but it can be faster than the design in Fig. 4 as it can detect a row error without reading it first. Due to limited space we do no analyze further this end-to-end RTD design.

6 RTD for Post-Silicon Validation to SpeedUp Bug Localization

Bug localization during post-silicon validation can be quite taxing, as it may require months to complete [15], delaying the launch of a product and resulting in grave economic consequences. What makes bug localization so challenging is the potentially large time window between a bug activation and its manifestation to an observable error, a vast expanse that needs to be covered to root-cause the bug. For instance, consider the example in Fig. 6 where a very rarely occurring bug corrupts an array entry (entry 4). Without any form of protection (No-Protection), the bug manifestation will be detected after the specific entry is read and the faulty value causes some abnormal behavior (e.g., divide by zero, or an illegal address exception), or it leads to a wrong program output that is detected by comparing against a golden reference. If the array employs a protection scheme that is not real time (nRTD), the error can be detected when the faulty entry is read. Although nRTD can reduce the error-detection latency, as compared to No-Protection, the time gap between the error cause and the error detection by nRTD can be arbitrarily long, e.g., more than a billion cycles. Consequently, even with nRTD, the root-causing procedure remains exceedingly hard and time consuming. The use of a RTD protection approach virtually eliminates the detection latency.

RTD usefulness for post-silicon validation hinges on its ability to capture difficult-to-detect bugs. The logical functionality of an array is not complex and it is feasible to test thoroughly during pre-silicon validation. Electrical bugs, however, that depend on subtle interplay between a design and its electrical state [5] can manifest after manufacturing and RTD can be quite effective in immediately detecting such bugs that cause memory-array corruption. For instance, such bug can be the result of a combination, at the same time, of a voltage drop and writing data over a critical speed-path, which causes the incorrect update of part of the data and/or parity bits in the memory array.

RTD can be used for the post-silicon validation of arrays that do not require protection in-the-field, because they are either expected to have small in-the-field error contribution, or are not architecturally vulnerable [16]. In either case, bits in control registers [17], referred to as chicken bits [18], can be used to enable the RTD protection in an array during manufacturing test and post-silicon validation, and disable them for field operation to avoid RTD's power overhead.

7 RTD Delay, Area and power Evaluation

7.1 Methodology

The 2D ECC RTD implementation introduced in Section 5.2 as well as the SECDED ECC are evaluated in terms of their impact on the salient metrics of delay (timing), area and power.

The mux-columns in the bit-slices are implemented using 2-input NAND-NOR trees as in [13]. The evaluated 2D ECC RTD design uses two-way horizontal logical-interleaving and its RTCP-columns are built using 2-input XOR trees. The SECDED designs, depending on the number of data bits, use different Odd-Weight columns generate and check matrices as in [14]. The checker produces the syndrome using 2-input XOR parity trees and each syndrome output bit is decoded using 2-input NOR+NAND trees. Numerous designs are evaluated with di*fferent number of* rows (4,8,16,32,64,128) and columns (2,4,8,16). To search the design space fast we use the analytical methodology in [19] and estimate area, delay and power figures expressed as equivalent gates in this work as follows:

TABLE 3

Area, Power And Delay Figures In Terms Of Equivalent Gates

Gate	Area,Power	Delay
Not	0.1	1
2-input Nand/Nor	0.2	1.5
2-input Xor	0.6	2.5
F/F cell	1	N/A

First-order analytical models are defined for the area, delay and power for each array design we evaluate (we present few indicative models in the APPENDIX).

We have validated the models by comparing normalized trends against three RTL implementations of a 64x64 F/F-based array; with no protection, only row-parity protection and an RTD design as in Fig. 2. The arrays are implemented in System Verilog. The designs are validated at the RTL level for functional correctness, synthesized to a commercial low-power 45nm standard-cell library under worst-case conditions (0.8V, 125C), and placed-and-routed with the Cadence digital implementation flow for minimum delay. The maximum error observed in timing is 4.5% and in area 5.3%.

7.2 Evaluation

Evaluation with and without PD port

Evaluation with and without vertical interleaving(?)

Fig. 4 reports the area and delay analysis for different array sizes being the number of rows in the x-axis and the number of columns in the line colors. The numbers are normalized to the SECDED performance (i.e., $\text{Area}_{2DECC+RTD}/\text{Area}_{SECDED}$; and $\text{Delay}_{2DECC+RTD}/\text{Delay}_{SECDED}$). Clearly, 2DECC+RTD outperforms in terms of delay SECDED in any configuration with 8% to 24% improvement. This is the direct benefit of using RTD. On the other hand, the area overhead of 2D-ECC+RTD is considerable (12% to 53%), especially for a small number of rows. In addition, the power overhead is substantial 21% to 42% (not shown in the graph for clarity). We note that our findings are specific to the designs evaluated and the methodology used. The cost and benefits from RTD may depend on many parameters including port topology and technology.

Overall, RTD is not free, and a designer will need to weigh the return-on-investment from RTD's potential to shorten access time and facilitate post-silicon validation (Section **Error! Reference source not found.**), against the costs RTD entails (e.g., die-area and power overhead). Such trade-off is difficult to quantify, as it requires intimate familiarity with design cycles and manufacturing costs, and it is beyond the scope of this work. Our main goal is to introduce the RTD approach as a design option.

8 RTD Impact on Soft-Error rate (SER)

SBU (Single Bit Upsets) are the most dominant SER fault type but multi-bit upsets (MBUs) also happen occasionally -and they are increasing in smaller technology nodes [20]. In [11], authors analyze MBUs in a 14nm FinFET-based F/F array subject to neutron radiation and they observed eleven different fault patterns of MBUs. MBUs of up to 4 bits bursts are observed in F/F arrays. In Fig. 8, we analyze the strength of several variations of 2D ECC + RTD (with different degrees of horizontal (H) and vertical (V) logical-interleaving) to detect and correct these patterns and we will compare this against the strength of a SECDED code. The outcome for each case is according to the corresponding Decoder behavior (see Section **Error! Reference source not found.**). For instance, consider fault pattern p3, a two-bit horizontal burst. SECDED will detect it but will be unable to correct it (i.e., a DUE). The 2D ECC RTD decoder without interleaving (Table 1) will detect an even number of 1s in the EV and it will also trigger DUE (column X-e-DUE). The decoder for the RTD scheme with 2-way horizontal interleaving (Table 2) will flag this as CE (column 1-1-o-o-C) because the two separate row parity trees per row detect the fault pattern and the EV corrects it.

The F/F array evaluated in [11] does not include: i) the logic column for a normal read access of the data; ii) the column for to read the value to overwrite; and iii) the column that calculates the real time column parity. These columns may present an "isolation" for a single-event to cause multiple faults in the horizontal/diagonal direction. Consequently, in Fig. 8, we present results for the patterns observed in [11] as well as for the same patterns without horizontal/diagonal MCU, only vertical MCUs (the four right-most columns- that capture better the strength of the proposal in our paper).

From the figure, we can observe that all 2D ECC+RTD configurations incur no SDCs (except the leftmost RTD column without any interleaving that is applied to all fault patterns without isolation) whereas SECDED does for some patterns. Yet, correction capabilities of 2D ECC+RTD are limited when no interleaving is used (leftmost RTD column without any interleaving) unless we consider the isolation introduced by this proposal (rightmost RTD column without interleaving). As the interleaving degree increases, RTD is able to correct more fault-patterns. When we consider the isolation presented by our scheme, a vertical 4-way interleaving is able to correct all fault-patterns.

The choice of interleaving depends on the expected fault patterns and their frequency. It rests with a designer to decide on the tradeoff between access time speed-up, hardware overhead and error detection and correction strength.

9 Related Work

Previous work has proposed using 2D ECC for caches [8][10]. The main difference of our work is that we do not require a very expensive (in terms of cycle count) correction procedure to determine the columns with errors. We circumvent this by using in-situ in hardware another port that tracks the RTCP. Two-dimensional codes were also the focus of [21] where a code for extreme error detection and correction is proposed (for very big error density).

Several schemes have been proposed to speed up SEC and SECDED codes. In [22] they propose an ECC scheme with small (or even without) delay penalty as compared to an unprotected array by relying on delayed clock for the array that holds the parity bits. However, this scheme requires separates arrays for data and check bits (which entails extra power, area penalty) and requires a complex system-hold capability upon error detection. Another work [23] proposed SEC and SECDED codes with lower delay than traditional algorithms. The main idea is to use extra parity bits to facilitate faster error detection and correction. In this work we do not require separate use for data and parity bits and explore the idea of using more combinational logic to simplify error protection.

10 Conclusions

We propose RTD, a hardware technique for detecting faults in arrays immediately after they happen, instead of after they are read. The paper presents the high-level architecture of RTD that relies on in-situ array hardware for tracking in real-time the column-parity. Then it shows how to use RTD to design a 2D ECC scheme. The work details how to implement the RTD scheme for an array with F/Fs. An evaluation, that compares the 2D ECC RTD against a SECDED design, reveals that RTD presents a trade-off between reducing access time vs extra area and power. The code strength of the two schemes is found to be comparable. RTD can also help reduce the time to root-cause bugs during post-silicon validation.

Future work…

Acknowledgment

Part of this work was performed during a sabbatical of the first author at the Intel Israel Development Center in Haifa.

REFERENCES

[1] R. W. Hamming, "Error detecting and error correcting codes," *in The Bell System Technical Journal*, vol. 29, no. 2, pp. 147-160, April 1950.

[2] R.C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, Volume 5, Issue 3, 2005, pp. 305 – 316.

[3] P. Ehligand and S. Pezzino, "Error Detection in SRAM," *Application Report SPRACC0*, Texas Instruments, Nov. 2017.

[4] Daniel P. Siewiorek and Robert S. Swarz., "Reliable Computer Systems (3rd Ed.): Design and Evaluation," A. K. Peters, Ltd, 1988.

[5] S. Mitra, S. A. Seshia and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," *Design Automation Conference*, 2010, pp. 12-17.

[6] L. Gwenap, "P6 Microcode can be Patched," Microprocessor Report, 1997.

[7] S. S. Mukherjee, J. Emer, T. Fossum and S. K. Reinhardt, "Cache scrubbing in microprocessors: myth or necessity?," *10th IEEE Pacific Rim International Symposium on Dependable Computing*, 2004.[7]

[8] J. Kim, N. Hardavellas, K. Mai, B. Falsafi and J. Hoe, "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding," 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), Chicago, IL, 2007, pp. 197-209

[9] J. Maiz et al., "Characterization of Multi-bit Soft Error events in advanced SRAMs", *EDM* 2003

[10] M. Manoochehri, M. Annavaram, and M. Dubois, "CPPC: correctable parity protected cache," I*n Proceedings of the 38th annual international symposium on Computer architecture (ISCA)*, 2011,pp. 223-234.

[11] S. Kumar et al., "Analysis of Neutron-Induced Multi-Bit-Upset (MBU) Clusters in a 14nm Flip-Flop Array," IEEEE TNS, vol. 66, no. 6, 2019

[12] P. Meinerzhagen, S. M. Y. Sherazi, A. Burg and J. N. Rodrigues, "Benchmarking of Standard-Cell Based Memories in the Sub-VT Domain in 65-nm CMOS Technology," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 2, 2011, pp. 173-182.

[13] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, Area, and Performance Optimization of Standard Cell Memory Arrays Memory Arrays Through Controlled Placement," *ACM TODAES.* 21, 4, 59, May 2016.

[14] M. Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," in IBM Journal of Research and Development, vol. 14, no. 4, pp. 395-401, July 1970.

[15] D. Lin, E. Singh, C. Barrett and S. Mitra, "A structured approach to post-silicon validation and debug using symbolic quick error detection," *IEEE International Test Conference (ITC)*, 2015, pp. 1-10.

[16] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," *International Symposium on Microarchitecture (MICRO)*, 2003.

[17] J. Geuzebroek and B. Vermeulen, "Integration of Hardware Assertions in Systems-on-Chip," *IEEE International Test Conference (ITC)*, 2008.

[18] I. Wagner and V. Bertacco, "The Verification Universe" in "Post-Silicon and Runtime Verification for Modern Processors", *Springer*, 2011

[19] D. Rossi, N. Timoncini, M. Spica and C. Metra, "Error correcting code analysis for cache memory high reliability and performance," 2011 Design, Automation & Test in Europe, Grenoble, 2011, pp. 1-6.

[20] G. Hubert, L. Artola and D. Regis, "Impact of scaling on the soft error sensitivity of bulk, FDSOI and FinFET technologies due to atmospheric radiation," Integration the VLSI journal, vol. 50, pp. 39-47, 2015.

[21] C. Argyrides, D. K. Pradhan, and T. Kocak, "Matrix Codes for Reliable and Cost Efficient Memory Chips," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 3, pp. 420-428, March 2011, doi: 10.1109/TVLSI.2009.2036362

[22] M. Nicolaidis, T. Bonnoit and N. Zergainoh, "Eliminating speed penalty in ECC protected memories," 2011 Design, Automation & Test in Europe, Grenoble, 2011, pp. 1-6, doi: 10.1109/DATE.2011.5763256

[23] P. Reviriego, S. Pontarelli, J. A. Maestro and M. Ottavi, "A Method to Construct Low Delay Single Error Correction Codes for Protecting Data Bits Only," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 3, pp. 479-483, March 2013, doi: 10.1109/TCAD.2012.2226585

[24] Y. Sazeides et al., "2D Error Correction for F/F based Arrays using In-Situ Real-Time Error Detection (RTD)," 2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1-4

APPENDIX: 2D ECC RTD MODELS FOR AREA AND DELAY

Note: Baseline (array without protection) model parameters are listed in **bold**

Area$_{RTD}$ *= Array Cells +2 * Read Ports+ RTCP Port +SCP Cells*
= (***#bitslices***$_{\mathbf{baseline}}$ *+ #bitslices*$_{rowparity}$*) **
(***#rows*** ** (2 ** ***NAND***$_{\mathbf{area}}$ ***+ cell***$_{\mathbf{area}}$*)*
*+ 2 ** ***#NANDsNORsperslice* NAND***$_{\mathbf{area}}$
*+ #XORsperslice*XOR*$_{area}$ *+ cell*$_{area}$*)*

Delay$_{RTD}$ *= the sum of the following delays (that include interconnect delay) define the critical path:*

R1. Invert read address (NOT gate)
R2. Read Address Decoder (NAND+NOR tree), depth = log$_{\mathbf{2}}$***(log***$_{\mathbf{2}}$***(#rows))***
R3. Global Read Row Gating, (NAND+NOT gates)
R4. Mask F/F Output, (NAND gate)
R5. Column MUX (NAND+NOR tree), depth=log$_{\mathbf{2}}$***(#rows)***
R6. 2-way horizontal interleaved row-parity (XOR tree), depth =log$_2$*(#bitslices*$_{baseline}$ */2+ 1)*
R7. Produce Correction Vector (NAND)
R8. Bitwise-XOR (XOR)

Yiannakis Sazeides is with the Department of Computer Science at the University of Cyprus. He has worked and contributed towards the development and design of high performance processors with Compaq and Intel. His research interests lie in the area of Computer Architecture with particular emphasis in fault-tolerance and reliability, data-center modelling, memory hierarchy and dynamic program behavior. He has received four best paper awards (MICRO 1997, DATE 2016, CAL 2017, DATE 2019) and has been granted five patents.

Arkady Bramnik is RAS and Functional Safety Hardware Architect at Intel Corporation in Haifa, Israel. His professional interests lie in Fault Tolerant Computer Architecture, permanent and transient error modeling, error detection and correction techniques. He is a co-author of 7 granted patents and 7 papers (including a DATE 2019 Best Paper Award).

Ron Gabor biography appears here.

Ramon Canal biography appears here.

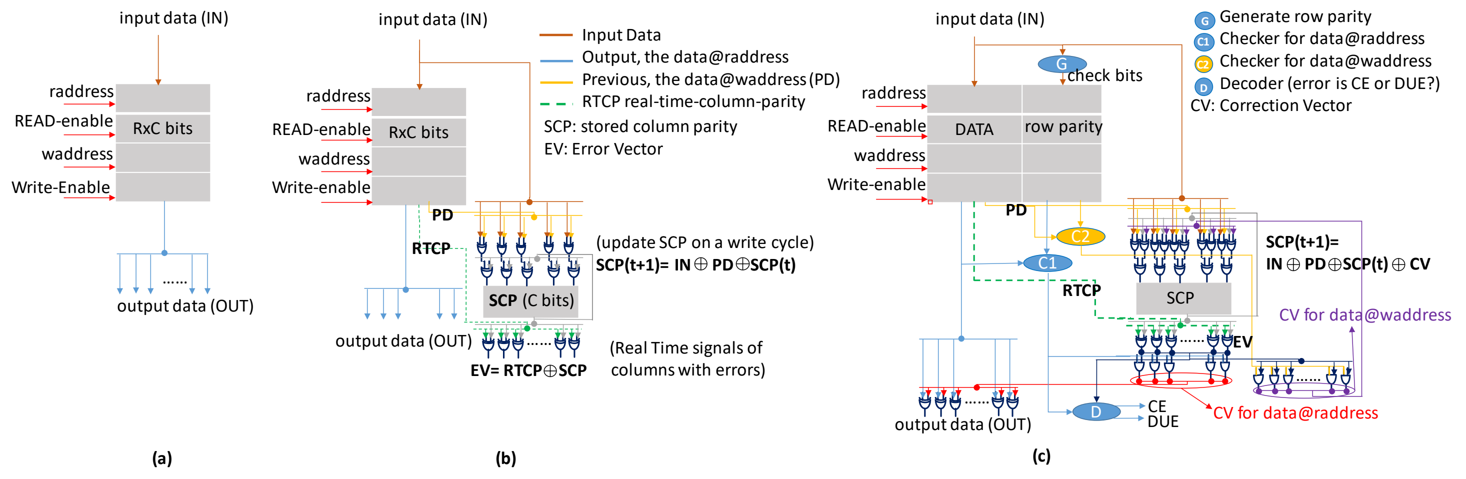

input data (IN)
raddress
READ-enable
waddress
Write-Enable
RxC bits
output data (OUT)
(a)
Input Data
Output, the data@raddress
Previous, the data@waddress (PD)
RTCP real-time-column-parity
SCP: stored column parity
EV: Error Vector
PD
RTCP
(update SCP on a write cycle)
SCP(t+1)= IN ⊕ PD ⊕ SCP(t)
SCP (C bits)
EV= RTCP⊕SCP
(Real Time signals of columns with errors)
(b)
Generate row parity
Checker for data@raddress
Checker for data@waddress
Decoder (error is CE or DUE?)
CV: Correction Vector
check bits
DATA
row parity
SCP(t+1)= IN ⊕ PD⊕SCP(t) ⊕ CV
CV for data@waddress
CV for data@raddress
CE
DUE
(c)

Fig. 1: a) Baseline Array without Protection, b) Array with RTD of Column Faults, c) Array with 2D ECC using RTD of Column Faults + nRTD of Row Faults

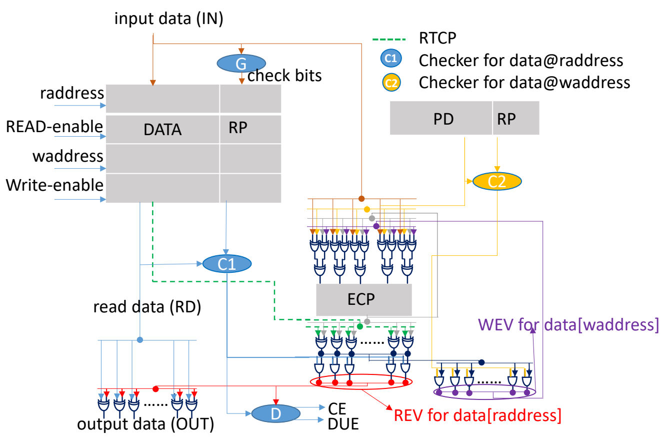


Fig. 2: 2D ECC RTD Architecture

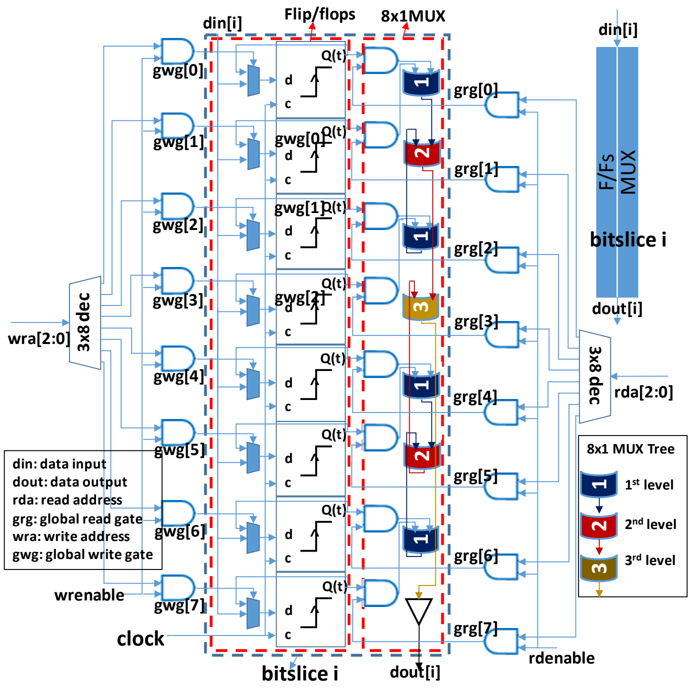
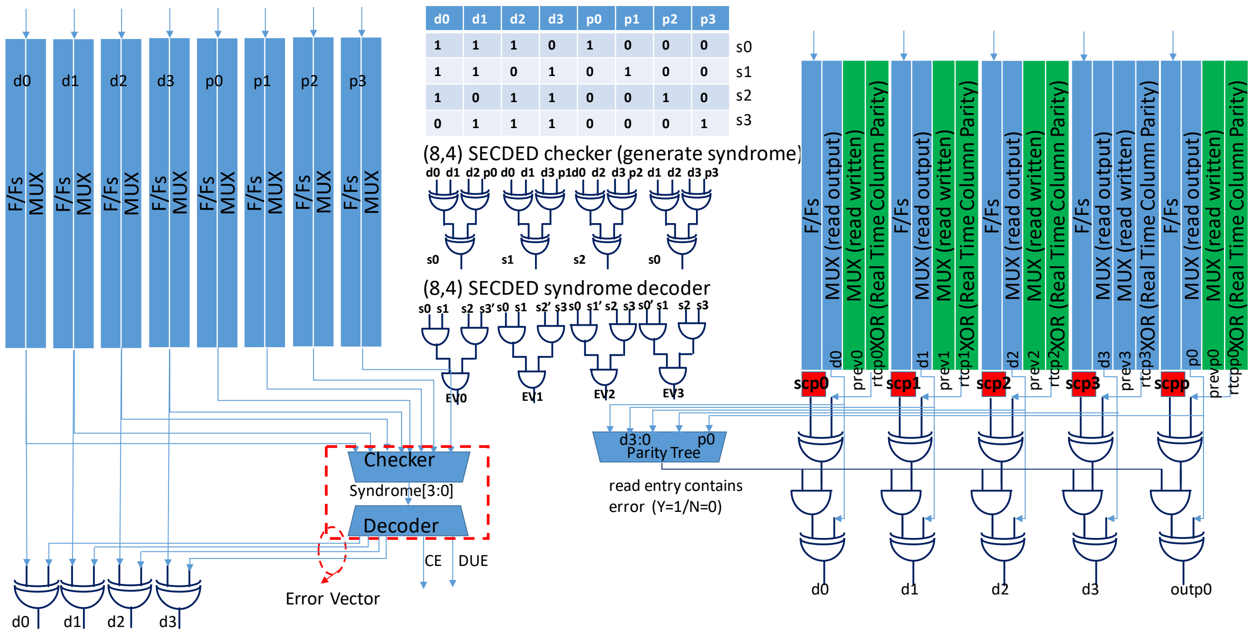
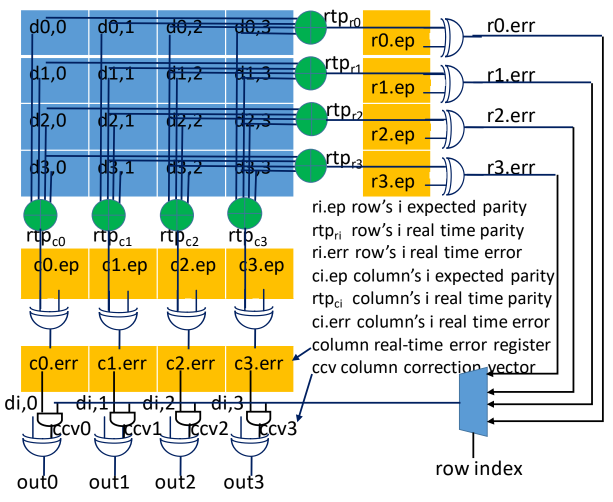


Fig. 3: Bit-slice with a column of F/Fs and a mux tree [11]



d0	d1	d2	d3	p0	p1	p2	p3	
1	1	1	0	1	0	0	0	s0
1	1	0	1	0	1	0	0	s1
1	0	1	1	0	0	1	0	s2
0	1	1	1	0	0	0	1	s3

Fig. 4: Traditional bit-sliced SECDEC organization (left) and 2D ECC bit-sliced column-based RTD organization (right) for an array with 4-data columns. Green columns are new bit-slice in-situ logic to support RTD.



d0,0 d0,1 d0,2 d0,3
d1,0 d1,1 d1,2 d1,3
d2,0 d2,1 d2,2 d2,3
d3,0 d3,1 d3,2 d3,3
rtp_{r0} rtp_{r1} rtp_{r2} rtp_{r3}
r0.ep r1.ep r2.ep r3.ep
r0.err r1.err r2.err r3.err
rtp_{c0} rtp_{c1} rtp_{c2} rtp_{c3}
c0.ep c1.ep c2.ep c3.ep
c0.err c1.err c2.err c3.err
ri.ep row's i expected parity
rtp_{ri} row's i real time parity
ri.err row's i real time error
ci.ep column's i expected parity
rtp_{ci} column's i real time parity
ci.err column's i real time error
column real-time error register
ccv column correction vector
di,0 di,1 di,2 di,3
ccv0 ccv1 ccv2 ccv3
out0 out1 out2 out3
row index

Fig. 5: Error Correction through Combined Row and Column RTD (figure does not show the logic that computes the expected parity and the logic that inverses the corrupted cell when calculating a column's expected parity)

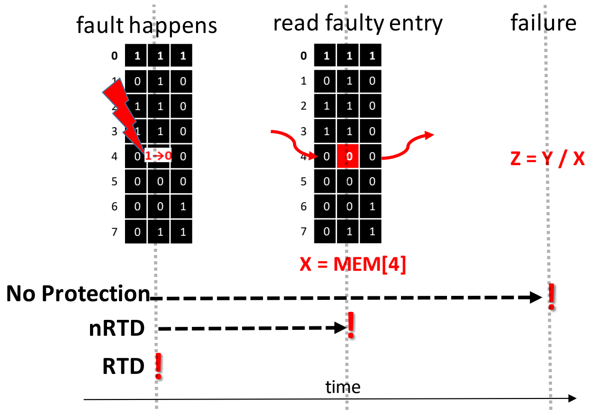


Fig. 6: RTD vs nRTD: Bug Detection Latency

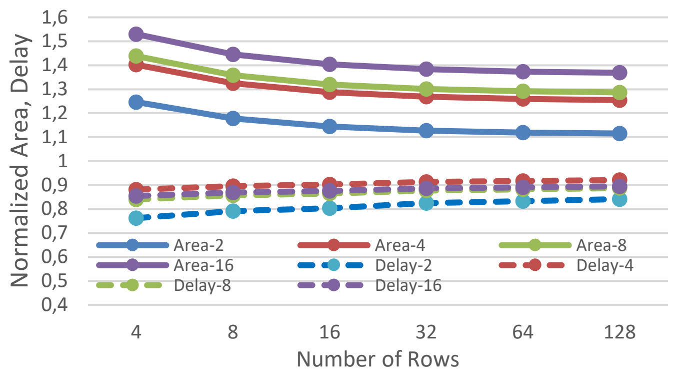


Fig. 7: SECDED vs. 2D-ECC RTD area and delay

PATTERN	SECDED	RTD	RTD + 2-way H inter.	RTD+ 2-way V inter.	RTD+ 4-way V inter.	RTD+ 2-way H inter.+ 2-way V inter.	PATTERN no horizontal/ diagonal MBU	RTD	RTD+ 2-way V inter.	RTD+ 4-way V inter.
p0	CE	CE	CE	CE	CE	CE		CE	CE	CE
p1,2	CE	DUE	DUE	CE	CE	CE		DUE	CE	CE
p3	DUE	DUE	CE	DUE	DUE	CE		CE	CE	CE
p4,5	CE	DUE	CE	CE	CE	CE		CE	CE	CE
p6	SDC	DUE	DUE	DUE	DUE	DUE		CE	CE	CE
p7	CE	DUE	DUE	DUE	CE	DUE		DUE	DUE	CE
p8	DUE	SDC	DUE	DUE	DUE	CE		DUE	CE	CE
p9	CE	DUE	DUE	DUE	CE	DUE		DUE	DUE	CE
p10	SDC	DUE	DUE	DUE	DUE	DUE		CE	CE	CE
p11	DUE	SDC	DUE	DUE	DUE	CE		DUE	CE	CE

Fig. 8: Strength of SECDED and different 2D ECC+RTD configurations for different MBU patterns (assuming one error at a time and no error accumulation).

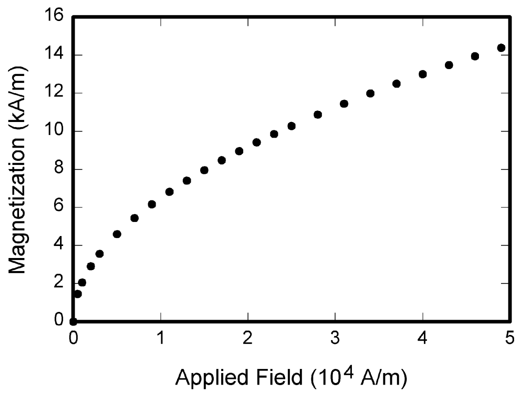


Fig. 1. Magnetization as a function of applied field. Note that “Fig.” is abbreviated. There is a period after the figure number, followed by one space. It is good practice to briefly explain the significance of the figure in the caption.