2D Error Correction for F/F based Arrays using In-Situ Real-Time Error Detection (RTD)

Yiannakis Sazeides
University of Cyprus, Cyprus

Arkady Bramnik
Intel, Israel

Ron Gabor
Toga Networks, Israel

Chrysostomos Nicopoulos
University of Cyprus, Cyprus

Ramon Canal
Universitat Politècnica de Catalunya, Spain

Dimitris Konstantinou
Democritus University of Thrace, Greece

Giorgos Dimitrakopoulos
Democritus University of Thrace, Greece

***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 an array with flip-flops to track in real-time the column-parity and introduces a two-dimensional RTD based error-correction scheme. 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 8-24% at an area and power overhead between 12-53% and 21-42% respectively.**

***Keywords*—reliability, memory arrays, error detection and correction, real-time error detection, bugs, post-silicon validation**

I. Introduction

Error detection and correction codes [1] are widely used to protect 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 word. When a codeword (data plus parity) is read from the array, the parity is calculated from the data and an error is detected if it does not match the parity 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 an 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 nRTD, detect the fault only after the stored data is read.

RTD has practical uses in reliability and post-silicon validation. For reliability (e.g., protect against soft-errors [2]) RTD can speed-up the access time of arrays required to provide in-line error-detection and correction. For post-silicon validation [3], 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 provides a significant access time reduction albeit with an area and power overhead.

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

II. RTD Architecture

We illustrate the extra functional requirements of an array with RTD using an example. The implementation of RTD, how to embed it inside an array, is the subject of Sec. IV. 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 shows the additional array interfaces and functionality needed to detect in real-time whether each array column contains a fault. Specifically, this requires having ***in-situ*** (i.e. built in the array) the following two extra read ports:

i) a port to track RTCP (the real-time-column parity). This port does not need an address decoder, to select a specific row, since it produces the xor of all bit values in each column, and

ii) a port that reads the previous data (PD) at the write address about to be overwritten during a write cycle. This port shares the same address decoder with the write port.

Additionally, RTD requires a SCP (stored-column-parity) register with C bits (as many as array columns). This register maintains the column parity and it is updated on every array write cycle with the bitwise-xor of the current value in SCP, the previous data (PD) and the value to be written in the array (IN). Finally, 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 SCP position).

The two key properties of RTD are: i) can detect a fault as soon as it occurs (without first reading the entry that contains the fault), and ii) maintaining the SCP does not require a read access to the array (to obtain the PD) before an array write. Put it another way, RTD in-situ hardware helps break the dependence between data access and error detection without the performance overhead of a read before write. These RTD characteristics open an opportunity to speed-up the access for arrays that require in-line error detection and correction, the main subject of the remaining paper.

978-1-7281-9457-8/20/$31.00 ©2020 IEEE

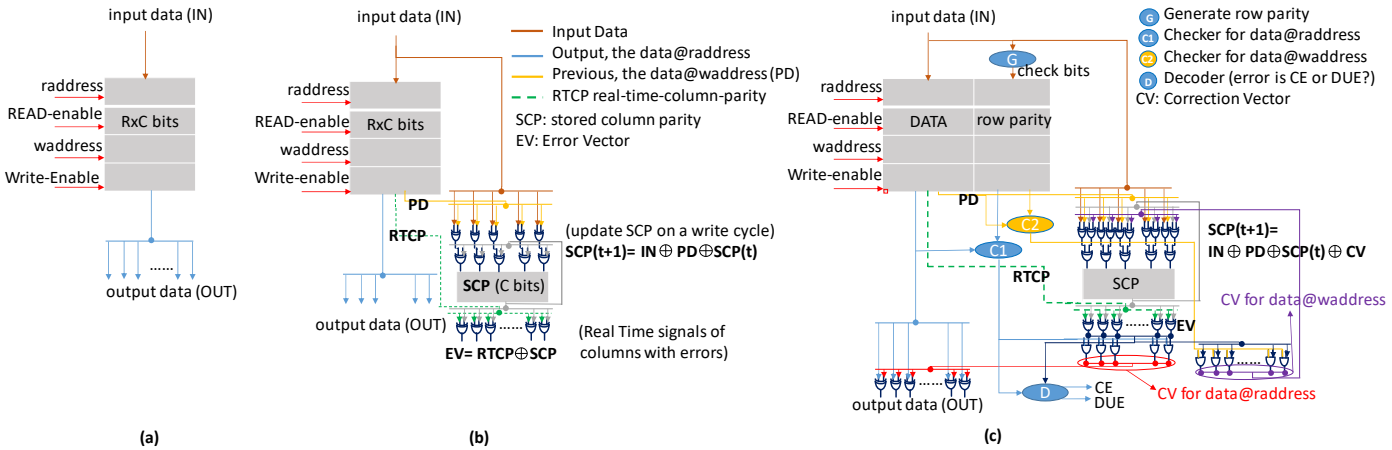

Fig. 1: a) Baseline Array without Protection, b) Array with RTD of Column Faults, c) Array with 2D ECC using RTD of Column Faults + nRTD of Row Faults

A. RTD Attributes

RTD requires one extra port to provide the RTCP and for each write port, an extra read port to read the previous data from the address to be overwritten on a write cycle. No extra read port is needed for shared R/W ports. For arrays that are updated at boot time (e.g., patch arrays [4]) and are read-only thereafter, there is no need for extra read port.

The error detection strength of RTD in Fig. 1.b is an odd number of faults in each column, i.e. it 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 [5][6]. For instance, to detect any burst of two consecutive vertical errors we need to use RTCP with 2-bit (set) vertical interleaving that tracks separately the RTCP for even and odd rows. Additionally, two SCP registers, each with C bits, need to maintain separately the parity of even and odd rows. On write and read cycles, extra logic (not shown in Fig. 1 due to space constraints) will control which RTCP to use and SCP to update depending on whether an even or odd row is accessed.

III. A RTD USE CASE: 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., ECC lies in the read critical-path. Unless stated otherwise, to simplify the discussion, we assume that at any given time faults are present only in one array row.

The scheme shown in Fig. 1.c combines RTD and nRTD in a 2D fashion to provide error correction. The RTD is used to track the array's column parity (as in Sec. 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 each row. The row parity is generated (denoted by G in Fig. 1 (c)) and stored on 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 parity mismatch and, since this is the only row with faults (assumption that a single row contains 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)).

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.

A single bit row-parity is unable to detect an even number of errors during a read. To prevent such silent-data-corruption (SDC), the decoder (D in Fig. 1.c) monitors the EV and triggers a DUE (detected-unrecoverable-error) when an even number of bits are set in the EV.

Based on the above, the behavior of the decoder (D) 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 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 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 contained only in one array row. The 1-o 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 an even number of faults is read since a parity bit 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 become a CE by employing horizontal logical-interleaving with degree equal to the burst length [6]. For instance, for a two-bit 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 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-bit 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)	N	D	C	D	C	C	D	D

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. This scheme provides comparable protection strength as SECDED for a two-bit horizontal errors bursts.

IV. RTD Applicability and Implementation

In this paper, we show how to implement RTD for a F/F based array. Such arrays are popular in modern CPUs [7]. F/F arrays with size up to a few thousand bits offer area and power advantages over equal-size SRAM-based arrays [8]. Consequently, F/F based arrays are attractive for inclusion in products, and novel techniques for error-protecting and debugging them, as in this work, are of practical value.

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.

A. RTD Implementation for a F/F Array

Our implementation of RTD is based on the bit-slice based F/F array design proposed in [8]. Each bit-slice contains a column of F/Fs and a column with a multiplexer tree that is used to read one of the cells 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. 2. An array consists of many bit-slices that share the same read and write address decoders for selecting which row to read and write.

Before presenting the RTD implementation, we show in Fig. 3 (left) a traditional (nRTD) in-line SECDED build using the bit-slice in Fig. 2. 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 [9]. 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. III is shown in Fig. 3 (right). It introduces in-situ, built in the bit-slice, an extra mux column to read the PD (the data to be overwritten on a write) and a column that determines the RTCP of the F/Fs in the bit-slice. The total number of bit-slices 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

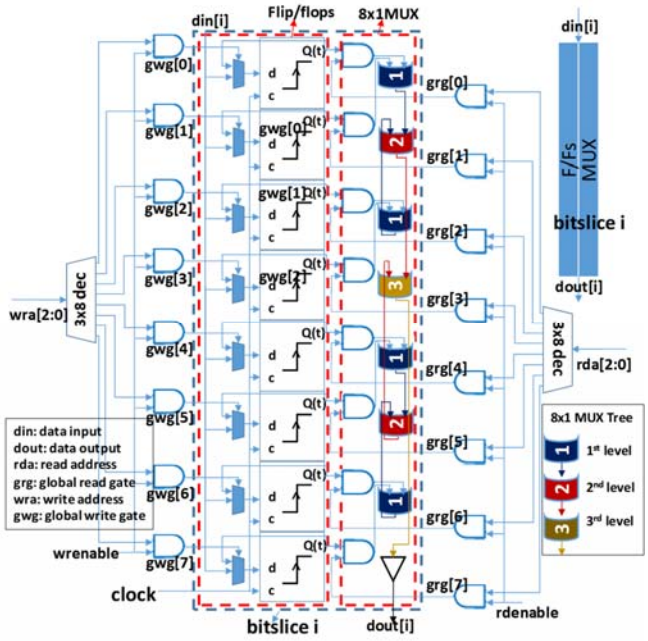

Fig. 2: Bit-slice with a column of F/Fs and a mux tree

the case of an error, the data are xored with the EV produced by xoring the SCP and RTCP (as in Fig. 1.c). Note that Fig. 3, for readability, only shows the design used during a read cycle.

Fig. 3 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.

V. RTD Delay, Area and Power Evaluation

A. Methodology

The proposed 2D ECC RTD implementation 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 [8]. The evaluated 2D ECC RTD design uses two-bit 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 [9]. 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 d*ifferent 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 [10] 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 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. 1.b. The

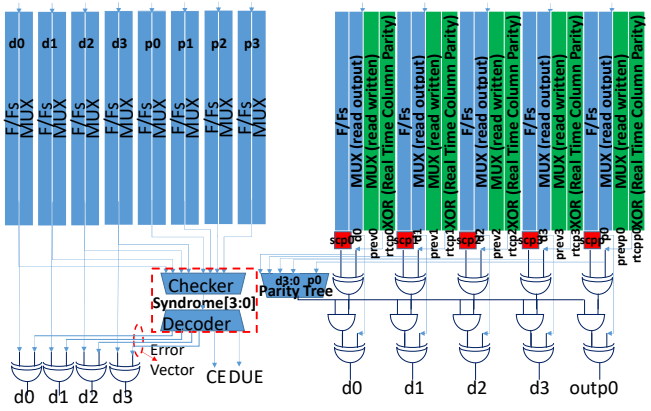

Fig. 3: Bit-sliced SECDEC (left) Bit-sliced 2D ECC RTD (right) for an array with 4-data columns. Green columns are new in-situ logic to support RTD.

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%.

B. *Evaluation*

Fig. 4 reports the normalized (to SECDED) 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. 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 Fig. 4 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 (Sec. VI), against the area and power costs RTD entails. Such trade-off is difficult to quantify, as it requires intimate familiarity with design cycles and manufacturing, and it is beyond the scope of this work. Our main goal is to introduce the RTD approach as a design option.

VI. RTD to Speedup Bug Localization

Bug localization during post-silicon validation can be quite taxing, as it may require months to complete [11], delaying the launch of a product. 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 a situation where a very rarely occurring bug corrupts an array entry. 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., an illegal address exception), or it leads to a wrong 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

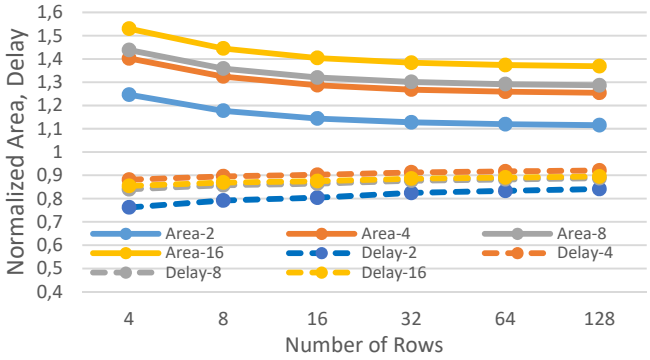

Fig. 4: SECDED vs. 2D-ECC RTD area and delay

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 RTD approach virtually eliminates the detection latency.

VII. Related Work

Previous work has proposed using 2D ECC for caches [5]. The main difference of our work is that to maintain the column parity we do not require a read-before-write that competes for a read port needed for regular reads as we add - in-situ in hardware - an extra read port for each write-port. Moreover, previous work requires a very expensive (in terms of cycle count) correction procedure to determine the columns with errors. We circumvent this by employing another port that tracks the RTCP.

VIII. Conclusions

RTD is a hardware technique for detecting faults in arrays immediately after they happen, instead of after they are read. RTD relies on in-situ array hardware for tracking in real-time the column-parity. We show how to use RTD to design a 2D ECC scheme. An evaluation comparing the 2D ECC RTD against a SECDED design for arrays with F/Fs, reveals that RTD can reduce access time at area and power cost. The code strength of the two schemes is comparable. RTD can also help reduce the time to root-cause bugs during post-silicon validation.

References

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

[2] R.C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE TDMR*, vol. 5, no. 3, 2005.

[3] S. Mitra et al., "Post-silicon validation opportunities, challenges and recent advances," *DAC*, 2010.

[4] L. Gwenap, "P6 Microcode can be Patched," *Microprocessor Rep.*, 1997

[5] J. Kim et al., "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding," *MICRO*, 2007.

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

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

[8] A. Teman et al., "Power, Area, and Performance Optimization of Standard Cell Memory Arrays Memory Arrays Through Controlled Placement," *ACM TODAES,* vol. 21, no. 4, May 2016.

[9] M. Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," in *IBM JRD*, vol. 14, no. 4, July 1970.

[10] D. Rossi et al., "Error correcting code analysis for cache memory high reliability and performance," *DATE*, 2011.

[11] D. Lin et al., "A structured approach to post-silicon validation and debug using symbolic quick error detection," *ITC*, 2015.