Low Cost Multi-Error Correction for 3D Polyhedral Memories

Mihai Lefter, Thomas Marconi, George Razvan Voicu, Sorin Dan Cotofana
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Delft, The Netherlands
{m.lefter, t.marconi, g.r.voicu, s.d.cotofana}@tudelft.nl

***Abstract*—In this paper we propose a novel error correction scheme/architecture specially tailored for polyhedral memories which: (i) allows for the formation of long codewords without interfering with the memory architecture/addressing mode/data granularity and (ii) make use of codecs located on a dedicated tier of the 3D memory stack. For a transparent error correction process we propose an online memory scrubbing policy that performs the error detection and correction decoupled from the normal memory operation. To evaluate our proposal we consider as a case study a 4-die 4-MB polyhedral memory and simulate various data width codes implementations. The simulations indicate that our proposal outperforms state of the art single error correction schemes in terms of error correction capability, being able to diminish the Word Error Rates (WER) by many orders of magnitude, e.g., WER from 10^{-10} to 10^{-21} are achieved for bit error probabilities between 10^{-4} and 10^{-6}, while requiring less redundancy overhead. The scrubbing mechanism hides the codec latency and provides up to 10% and 25% write and read latency reductions, respectively. In addition, by relocating the encoders/decoders from the memory dies to a dedicated one a 13% footprint reduction is obtained and parallel energy effective scrubbing can be enabled, which results in further WER reductions.**

I. Introduction

Increased integration factor and technology shrinking make Integrated Circuits (ICs) more prone to different defect types during the manufacturing process [1] and to in field degradations [2]. With memory cell size reduction, Multi-Bit Upsets (MBUs) become much more frequent since a particle hitting a memory array creates disturbances in multiple physically adjacent cells [3], [4], [5], [6]. In [7] a maximum MBU bit multiplicity of over 100 bits is predicted for 32 and 22nm SRAM generations, thus making traditional Single-Error Correction (SEC) ECCs with column interleaving [8] not anylonger sufficient [9] to mitigate the large amount of MBUs. In view of this, alternative approaches, like ECCs with Multi-Error Correction (MEC) capabilities become of high interest.

Three dimensional stacked ICs (3D-SICs) based on Through-Silicon-Via (TSV) interconnects [10] rely on an emerging technology which further boost the trends of increasing transistor density [11] and performance, since it enables smaller footprints, high bandwidth low latency interconnection and heterogeneous integration, while facilitating dependable computing [12]. While most of the 3D memory designs just follow a certain folding strategy, the 3D polyhedral memory architecture proposed in [13] brings a different fresh view into the field. It consists of multiple identical memory banks stacked on top of each other, while TSVs bundles distributed across the entire memory footprint traverse all the stacked dies to enable an enriched memory access set not achievable in planar counterparts.

In this paper we introduce a novel error correction scheme/architecture tailored for polyhedral memories. The main idea behind our approach consists in performing MEC ECC encoding/decoding on larger data widths (e.g., 512 to 4096 bits) such that a better error correction capability is obtained with the same or even lower redundancy than the one required by state of the art 64 data bit SEC schemes. For this we exploit the polyhedral memories organization, which inherently fast and customizable wide-I/O vertical access mechanisms allow for the long codeword creation with data and check bits from multiple mats. Furthermore, we propose to make use of codecs located on a dedicated tier of the 3D memory stack, since polyhedral memories allow for a smooth transfer of those long codewords to/from the dedicated error correction tier. In order to make the error correction process transparent to the memory users, e.g., processing cores, we propose an online memory scrubbing policy that performs the error detection and correction decoupled from the normal memory operation.

We evaluated our proposal by considering as a case study a 4-die 4-MB polyhedral memory protected by the proposed MEC mechanism with various data width codes implementations. The simulation experiments indicate that our proposal outperform state of the art schemes in terms of error correction capability, by significantly diminishing the Word Error Rates (WER), e.g., WER from 10^{-10} to 10^{-21} are achieved for bit error probabilities between 10^{-4} and 10^{-6}. Additionally, the scrubbing mechanism hides the codec latency and provides up to 10% and 25% write and read latency reductions, respectively. Furthermore, by relocating the encoders/decoders from the memory dies to a dedicated one a 13% footprint reduction is obtained and parallel energy effective scrubbing can be enabled, which results in even larger WER reductions.

The outline of the paper is the following. In Section II we describe our MEC error correction proposed approach. In Section III we evaluate the implications of our proposals and perform a comparison with state of the art memory error correction approaches. Section IV concludes the paper.

978-1-5090-6037-5/17/$31.00

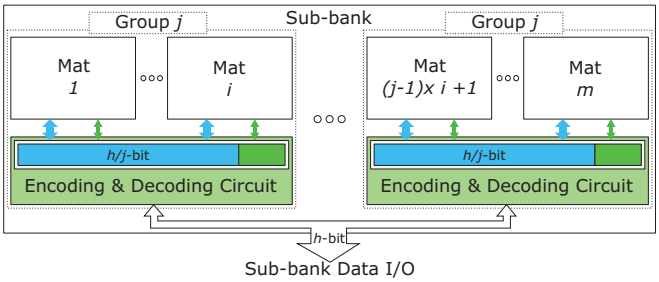

Fig. 1. *On Die Internal ECC* for Polyhedral Memories

II. ECC for Polyhedral Memories

In this section we detail our proposal for a Multi-Error Correction (MEC) ECC mechanism specially tailored for polyhedral memories [13]. The main goal is to allow for the formation of long data width codewords, case in which BCH codes are more effective, while not changing the normal memory addressing granularity, which in most of the current architectures is 64 bit. Based on the encoders/decoders (codecs) placement we detail next two scenarios: (i) *On Die Internal ECC* and (ii) *Dedicated Die ECC*. We note that the two scenarios can be employed independently or synergistic.

A. On Die Internal ECC

In this scenario the codecs could be physically located on each die, i.e., coplanar with the memory dies, as depicted in Figure 1. The h-bit data I/O of each sub-bank is split in j groups of h/j bits on which on the fly encoding/decoding is applied on memory write/read accesses. The total number of required codecs is equal with the number of groups (j) multiplied with the number of sub-banks in a die.

B. Dedicated Die ECC

As detailed in [13], polyhedral memories allow for vertical data access at different granularities: various sub-banks could be accessed in parallel on different dies, with the largest amount of vertically accessed data being obtained when all the TSVs are utilized. To better exploit the polyhedral memories rich access set we propose the *Dedicated Die ECC* scenario, depicted in Figure 2. Specifically, we propose to: (i) form extended codewords by combining data and check bits from multiple mats, but which could now be part of different sub-banks, as opposed to the previous scenario which was restricted to combining data only from mats in the same sub-bank, (ii) augment the 3D memory stack with an extra die dedicated to the execution of the encoding and decoding actions, (iii) employ the TSVs to transfer the extended codewords on/from the dedicated error correction die, and (iv) rely on an online programmable scrubbing based memory maintenance policy, which performs the error detection and correction activities with as little interference as possible with data read/write requests coming from the SoC computation cores. Based on the required error protection level, the online memory maintenance policy operates in one of the following modes: **scrubbing only**, **on the fly write with scrubbing**, and **on the fly write and read**.

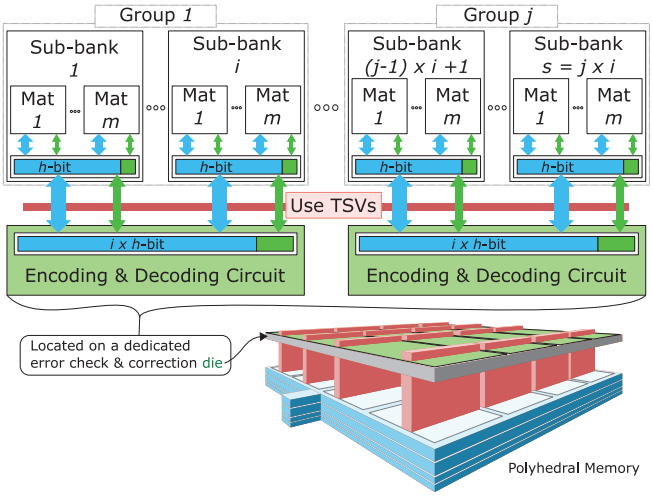

Fig. 2. MEC-based ECC for Polyhedral Memories

Memory maintenance operates mainly in the **scrubbing only** mode, in which, at certain time periods (scrubbing intervals), the entire memory is scanned (in a time period which we call scrubbing time) and eventual errors are corrected. The scrubbing interval and the scrubbing time depend on: (i) the employed MEC code performance (ECC encoding/decoding latency), (ii) the memory latency, (iii) the number of MEC codecs available on the error correction die, (iv) eventual memory conflicts, (v) the overall memory capacity, and, (vi) the bit error occurrence rate, which depends on aging and environmental aggressions.

The scrubbing maintenance operates in a similar way the DRAM refresh does and in case of conflicts its read/write accesses have lower priority than normal read/write memory accesses issued by the computation cores. We note that, different from the DRAM refresh, memory scrubbing requires additional steps related to the codeword formation and transfer to/from the codec die. Moreover coding and/or decoding it is much more time consuming than a write-back thus by implication maintenance related memory accesses are less frequent than the computational related ones. Consequently, in order to optimize both scrubbing intervals and scrubbing times, the scrubbing controller can take advantage of: (i) the polyhedral memory reach access mode set and (ii) the low maintenance related accesses occurrence, to dynamically adapt its access schedule such that memory conflicts are minimised if not completely avoided.

In this way memory error correction activities are executed in background and no decoding/coding actions are part of normal memory read/write accesses. This has the main advantage that the memory access time is practically unaffected by its augmentation with MEC ECC capabilities. However, if the memory operates in highly aggressive environmental conditions, or aging effects become predominant, resulting in high error rate, it is possible that the scrubbing process cannot keep the pace in cleaning the induced bit-flips. In this case faulty data might be read by the application(s) running on the platform embedding the ECC protected memory which

may result in unpredictable application behaviour. Note that not all the bit-flips are malicious as they might be rewritten before being read again. Such situations can be detected by the Operating System (OS), which can decide to change the ECC modus operandi to an on de fly coding/encoding mode.

Note that even if from the memory user point of view a data write seems coding free, it launches an encoding process too. This is because a write at address A invalidates all the check bits of codeword C (to which the data item at address A belongs). Consequently, the codeword C data bits need to be transferred via TSVs on the dedicated error correction die in order to be re-encoded, after which the newly calculated check bits are written back into the memory (data bits could be also rewritten for more protection). Invalidated codewords are handled by the scrubbing process and depending on the bit error rate we could place the invalidated codeword in the front of the scrubbing queue or to take no further action as there is a large chance that another write may occur within the same codeword in the close future. Note that the write invalidation policy may have an impact on the error correction capability. Eventual bit errors which may appear between the invalidation and the actual check bits update become indiscoverable, as when they are read in relation with the codeword re-encoding they will be considered valid.

If the aging and environmental conditions ask for a better protection, the **on the fly write with scrubbing** mode could be employed, in which the eventual write invalidation errors described above are avoided. When a word write is requested at address A the following actions have to take place:

1) Load the codeword C (to which address A belongs) on the error correction die via the afferent TSV bundle.
2) Decode C and correct possible errors if any.
3) Replace the value at A with the new to be written value.
4) Re-encode the codeword C.
5) Store C back to the memory via the same TSV bundle.

While the **on the fly write with scrubbing** operation mode eliminates the write invalidation problem, errors could still propagate in the following manner: eventual bit errors which may appear between two scrubbing scans could still propagate when read operations on those locations are performed. To avoid such situations the **on the fly write and read mode** can be employed. In this operation mode when a read request at address A (which is part of codeword C) is issued, the following actions have to take place:

1) Load via the afferent TSV bundle the entire codeword C on the codec die.
2) Decode C and correct possible errors if any.
3) Send the (corrected) data value stored at A to the computation core that issued the request.
4) In case errors were found at Step 2:
   - Re-encode the codeword C.
   - Store C back via the same TSV bundle.

We note that from the SoC (memory user) point of view, which is the one that really matters, the memory maintenance is 100% successful when it can keep pace with the error formation rate such that data read generated by the running application(s) never return corrupted data. This does not mean that all memory locations should be error free, which is quite normal as memory locations not currently read by the application do not influence its behavior thus they may contain erroneous bits. The error protection operation modes can be dynamically switched by the application by monitoring the scrubbing effectiveness. In this way the protection level is adjusted to the environmental aggression level and SoC aging status. We note however that the execution of codec activities during read/write operations as done when "on the fly" modes are activated has detrimental impact on SoC performance in terms of throughput, latency, and energy consumption.

The 3D memory is constructed by stacking identical memory dies which dimensions determine the IC footprint. In view of this the error correction die can be as large as the memory dies, thus for state of the art memory capacities it might be able to accommodate more than one MEC codec, case in which the memory maintenance process can be parallelized resulting in a higher error resilience. Finally, it is worth noticing that if memory accesses have a larger granularity, the MEC (re-)coding overhead is diminishing, and the read/write complexity can be reduced by, e.g., write buffers, caching.

To explore the potential of our proposal, we evaluate in the next section the error correction capability of our approach and compare it with the one of state of the art approaches.

III. Performance Evaluation

In this section we evaluate our MEC ECC proposal for polyhedral memories. First, we assess the reliability gains induced by the utilization of extended codewords instead of the traditional SEC ECCs. Second, we evaluate the implications of our proposal in terms of memory footprint, access time, and energy consumption. In our simulations we consider as a case study a 4 die 4-MB polyhedral memory with a 512-bit wide horizontal interface and a 4096-bit wide vertical interface. On each die there are 8 sub-banks (with an 512-bit I/O width), each being composed of 8 mats (with an 64-bit I/O width).

A. Error Correction Capability

To evaluate our proposal performance in terms of reliability for different redundancy ratios, we simulated the following protections mechanisms:

1) State of the art 64-bit SEC ECC (**SEC_CONV**).
2) Binary BCH codes ([14], [15]) operating on up to 4096-bit, denoted as **D/C/E**, with D, C, E as the number of data bits, check bits, and correctable errors, respectively.

To simulate memory fault occurrences, we made use of a Binary Symmetric Channel (BSC) model with crossover probabilities (α), i.e., the probability that a memory bit is being flipped, from 10^{-6} up to 10^{-2}. The performance of all the considered designs in terms of Word Error Rate (WER) assessed by means of Monte Carlo simulations is graphically depicted in Figure 3. The traditional SEC_CONV ECC protected memory, plotted with dark blue solid lines and circle markers, is considered as a reference, thus in the figure we plotted: (i) polyhedral memory implementations with

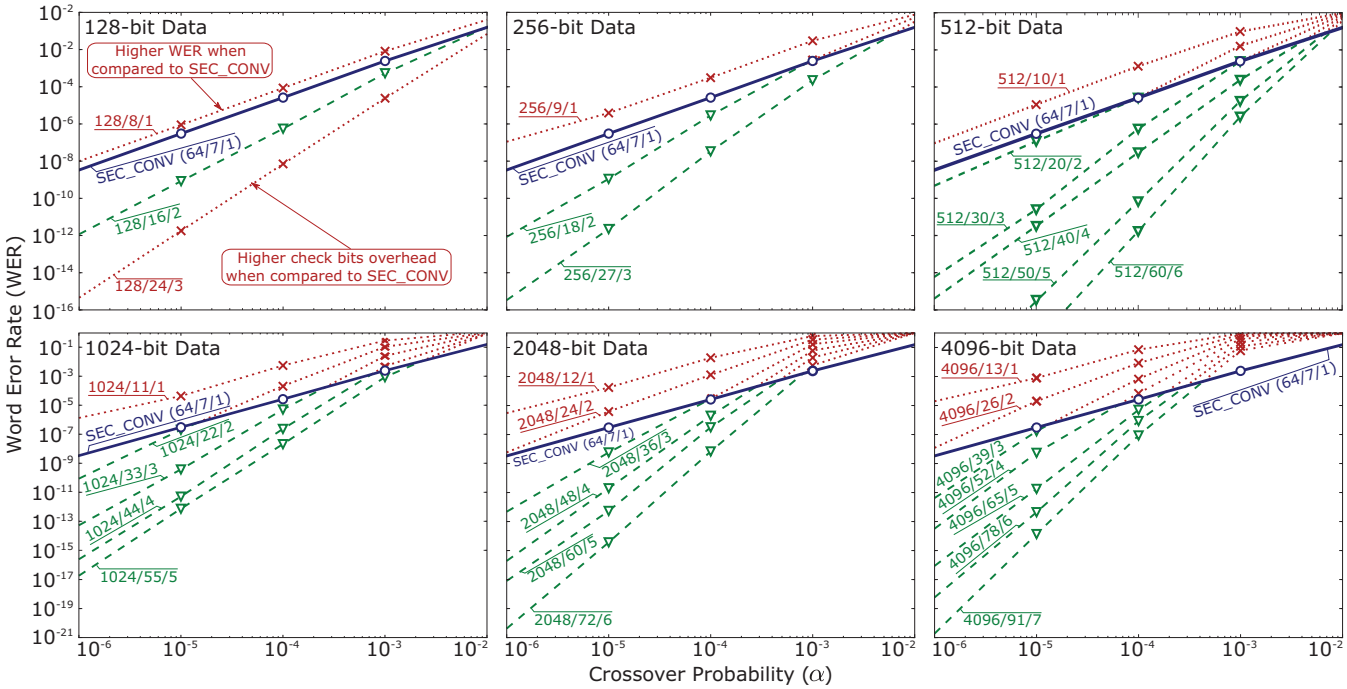

Fig. 3. Word Error Rate vs Bit Flip Probability

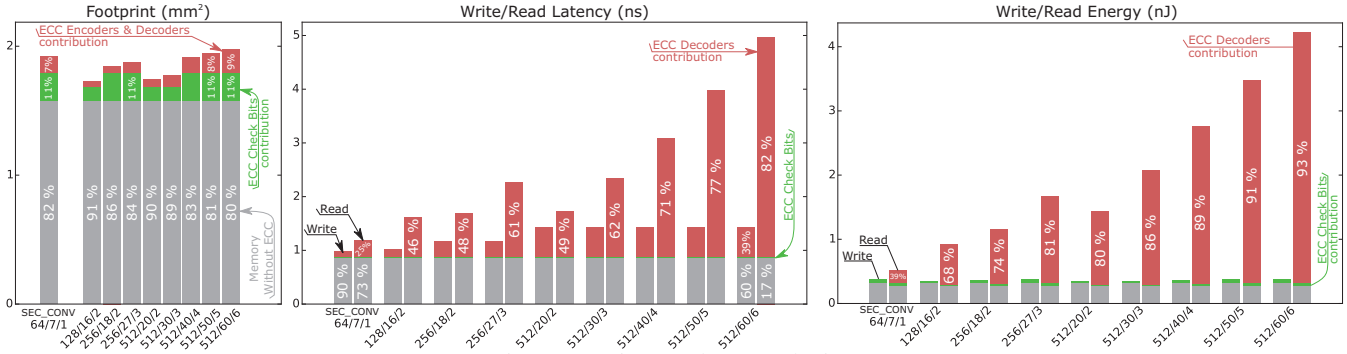

Fig. 4. On-Die Internal ECC Evaluation.

higher WER (e.g., 128/8/1) and/or with higher redundancy ratio (check bits overhead, e.g., 128/24/3) when compared with SEC_CONV with dotted red lines and x markers, while, (ii) polyhedral memories with smaller WER and lower redundancy with green dashed lines and V markers.

Figure 3 shows that there is no "panacea universalis", but most D/C/E polyhedral memory configurations outperform the state of the art in terms of WER at the expense of lower redundancy requirements. We observe the following situations:

1) D/C/E configurations with small data width and not enough redundancy, i.e., $128/8/1$ and $256/9/1$, which provide a higher than SEC_CONV WER for the entire considered α domain (not of practical interest).
2) D/C/E configurations with small-medium data width and low redundancy, i.e., 128/16/2, 256/27/3, 512/50/5, which substantially outperform SEC_CONV in terms of WER for the entire considered α domain while also diminishing the required redundancy by, e.g., 20% in the $256/27/3$ case.
3) D/C/E configurations with medium-large data width and extremely low redundancy, i.e., $1024/44/4$, $2048/60/5$, $4096/78/6$, which outperform by many orders of magnitude SEC_CONV in terms of WER for $\alpha < 10^{-4}$ while also diminishing the required redundancy by, e.g., 4.2x in the $2048/60/5$ case.

Figure 3 clearly demonstrates that our approach creates a large design space one can explore in quest for the most appropriate D/C/E configuration corresponding to the expected memory operation conditions, i.e., the α range, and targeted maximum acceptable WER value. For example, if the operating conditions require $\alpha \leq 10^{-5}$ and WER $\leq 10^{-12}$ any of the following configurations can be utilized: $512/40/4$, $1024/44/4$, $2048/60/5$, and $4096/78/6$. The actual choice can be done based on criteria like the lowest redundancy ratio but other aspects related to the memory architecture and utilization may also play a role in the best candidate selection. We mention that the above results can be regarded as a lower bound and that we expect even lower WER for BCH protected memories if a more realistic fault injection model, dedicated for MEC, is employed.

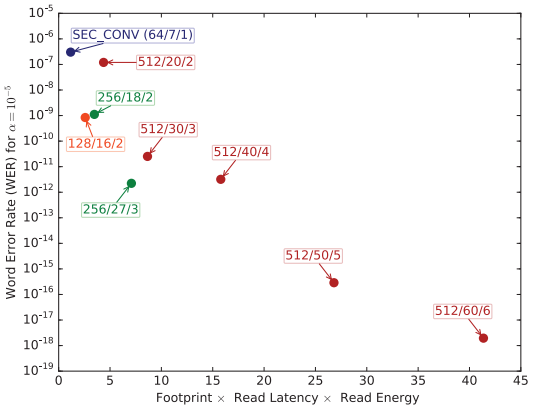

Fig. 5. On-Die Internal Trade Offs.

B. On Die Internal ECC Overhead

While in the previous section we demonstrated that our proposal can substantially outperform state of the art ECC approaches with similar or less redundancy, it is of interest to evaluate the actual implications of the considered MEC ECC schemes in terms of die footprint, memory access time, and energy consumption. To this end we considered a $22nm$ CMOS technology and estimated the footprint, latency, and energy of the utilized BCH encoders/decoders by using formulas from [16], [17], [18]. Additionally we obtained the ECC augmented memory dies footprint, latency, and energy values with a modified Cacti 6 [19] simulator utilizing the approach from [13]. We have evaluated only the MEC BCH protected memories which, in the previous reliability subsection, proved better in terms of error correction capability. The results of our evaluations are depicted in Figure 4.

One can conclude from the footprint graph (Figure 4 left) that: (i) the combined ECC area contribution (including check bits and codecs) is smaller than 20% for all of the considered ECC implementations, and (ii) the codecs overhead is always smaller than the check bits overhead. When comparing the implementations, the SEC_ CONV implementation requires one of the biggest footprint, being matched only by the largest 512-bit codecs considered ($512/50/5$ and $512/60/6$). We note that the $128/16/2$ implementation provides a 10% footprint reduction when compared with SEC_CONV while substantially outperforming it in terms of WER (see Figure 3).

In terms of latency (Figure 4 center), the ECC contributions are increasing when larger data width codecs are employed, reaching a maximum of 40% for write operations, while for read operations the ascending trend is much more pronounced, reaching a maximum of 83%. The ECC contribution to the memory access time is almost entirely determined by the codecs and all of the **D/C/E** MEC ECC implementations exhibit an increased latencies when compared to SEC_CONV.

In terms of energy consumption (Figure 4 right) a significant increase is observed for read operations for the ECC schemes spanning from 39% for SEC_CONV and going up to 93% for larger data width MEC codecs. For write operations the encoder contributions are insignificant (less than 3%) for all the implementations.

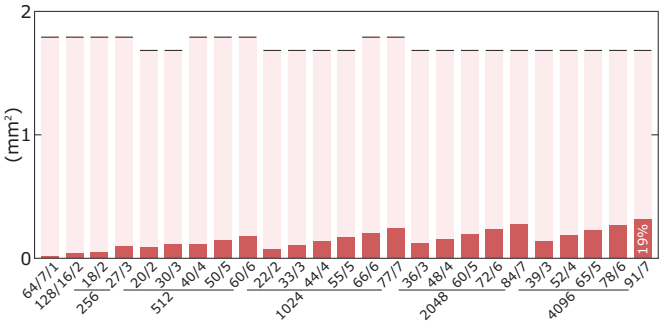

Fig. 6. Dedicated Die ECC - Footprint Comparison.

In order to get a better insight into the possible trade-offs, we have plotted on the bottom left of Figure 4 the WER (for a constant crossover probability $\alpha = 10^{-5}$) of all the considered approaches against the compound footprint × read latency × energy metric. One can observe in the figure that there is no absolute winer. The SEC_CONV implementation is the best in terms of the compound metric, while it is the worse in terms of error correction capability: it provides a WER which is with less then 2 orders of magnitude smaller than the considered α. On the other hand, the $512/60/6$ implementation is the best in terms of error correction capability (it provides a WER which is with 13 orders of magnitude smaller than the considered α) but it is the worse in terms of the footprint × read latency × energy metric. More balanced options are, e.g., $128/16/2$ or $256/27/3$ implementations, and given that at design time one knows the expected α range and the maximum acceptable WER value one can make use of such a plot in order to identify the most appropriate MEC ECC organization.

C. Dedicated Die External ECC Analysis

Since the *Dedicated Die External ECC* approach has a special error correction die, in Figure 6 we plotted the polyhedral memory footprint (left side of the figure) and the codec area on the error correction die (right side) for all the evaluated **D/C/E** schemes. Notice that the ECC die has a substantial amount of available area as even the largest codec is not utilizing more than 20% of its real-estate. This means that the remaining area could be either employed by extra codecs, to improve the error correction capability, or by other SoC resources. We also note that all of the *Dedicated Die External ECC* approaches provide a footprint reduction of 10% or 13% (depending on the required check bits) when compared with the SEC_CONV *On Die Internal ECC*, mostly because the codecs have been relocated on a dedicated die.

As mentioned in Section II-B, there are various operation modes for the *Dedicated Die External ECC* approach. Here, we limit our discussion to the scrubbing operation mode - the preferred modus operandi as in this use case no ECC related latency penalty per memory access is induced, irrespectively of the employed MEC ECC. This translates into write and read latency reductions of 10% and 25%, respectively, when compared to the SEC_CONV *On Die Internal ECC* scheme.

An important metric for the *Dedicated Die External ECC* approach is the scrubbing time, which is essential to be as short as possible, in order for the error correction controller

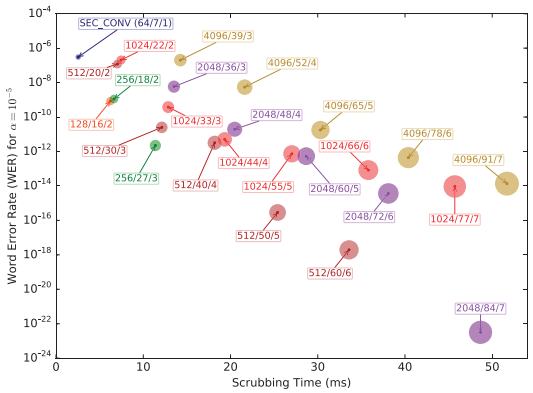

Fig. 7. WER vs. Ideal Scrubbing Time.

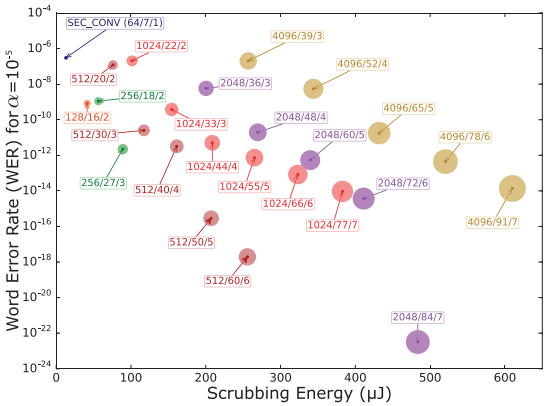

Fig. 8. WER vs. Ideal Scrubbing Energy.

to cope with high error rates. In Figure 7 we assume a bit error probability $\alpha = 10^{-5}$ and present the relation between the scrubbing time and the obtained codecs WER. We evaluated a scrubbing time lower bound as we assumed that no memory access conflicts occurred between the normal SoC memory accesses and scrubbing related TSV traffic. In practice such conflicts may occur but most can be solved by an adaptive scheduling policy. Figure 7 shows that the SEC_CONV implementation has the fastest scrubbing time but it is very weak in terms of error correction capability. In contrast, the $2048/84/7/6$ implementation is the best in terms of error correction capability but it is the worst in terms of scrubbing time. Many other possibilities exist between the two extremes, one of the most balanced option being $512/50/5$.

Regarding energy, again there are no direct penalties incurred per memory accesses but the *Dedicated Die External ECC* scrubbing mechanism consumes energy whenever it performs a memory scan. One can observe in Figure 8 that the SEC_CONV implementation performs best in terms of energy (it consumes the least amount of energy), while it is the worse in terms of error correction capability. At the same time the $2048/84/7/6$ implementation is the best in terms of error correction capability but it is the worse in terms consumed energy. Again, one of the most balanced options is $512/50/5$.

We note however that if more encoders/decoders are employed on the ECC die (as there is available room for that), the scrubbing time can be reduced. For example, if seven $2048/84/7$ codecs are employed (such that the entire ECC die is utilized), the ideal scrubbing time is reduced to almost the same value as the one corresponding to SEC_CONV, and $2048/84/7$ becomes the most balanced option from the WER vs. ideal scrubbing time perspective.

IV. Conclusions

In this paper we introduced a novel error correction mechanism for 3D wide-I/O polyhedral memories. The main idea behind our approach was to create the premises for applying ECC on larger data widths such that MEC can be performed with the same or even lower redundancy than the one required by state of the art 64 data bit SEC schemes. In addition, we proposed an online memory scrubbing policy that can perform the error detection and correction decoupled from the normal memory operation. We evaluated our proposal by considering a 4-die 4-MB polyhedral memory as a case study and simulated various data width codes implementations. The simulations indicated that our proposal outperforms state of the art schemes in terms of error correction capability, being able to diminish the Word Error Rates (WER) by many orders of magnitude, while requiring less redundancy overhead. Moreover, by relocating the codecs on a specialized die in the 3D memory stack we managed to hide the codec latency and provided 10% and 25% write and read latency reductions, respectively. At the same time a 13% footprint reduction was obtained.

References

[1] S. Nassif, "The light at the end of the cmos tunnel," in *ASAP*, July 2010.
[2] S. Rusu *et al.*, "T3: Trends and challenges in vlsi technology scaling towards 100nm," in *ASP-DAC*, 2002.
[3] K. Mohr and L. Clark, "Delay and area efficient first-level cache soft error detection and correction," in *ICCD*, 2006.
[4] N. Seifert *et al.*, "Radiation-induced soft error rates of advanced cmos bulk devices," in *IRPS*, 2006.
[5] G. Georgakos *et al.*, "Investigation of increased multi-bit failure rate due to neutron induced seu in advanced embedded srams," in *VLSI Circuits, IEEE Symposium on*, 2007.
[6] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *IRPS*, 2011.
[7] E. Ibe *et al.*, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *IEEE TED*, 2010.
[8] T. Suzuki *et al.*, "0.3 to 1.5v embedded sram with device-fluctuation-tolerant access-control and cosmic-ray-immune hidden-ecc scheme," in *ISSCC*, 2005.
[9] R. Naseer and J. Draper, "Parallel double error correcting code design to mitigate multi-bit upsets in srams," in *ESSCIRC*, 2008.
[10] P. Garrou *et al.*, *3D Integration: Technology and Applications*. Wiley-VCH, 2008.
[11] B. Prince, *Vertical 3D Memory Technologies* . Wiley, 2014.
[12] S. Safiruddin *et al.*, "Is 3d integration the way to future dependable computing platforms?" in *OPTIM*, May 2012.
[13] M. Lefter *et al.*, "A shared polyhedral cache for 3d wide-i/o multi-core computing platforms," in *ISCAS*, 2015.
[14] A. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres (Paris)*, 1959.
[15] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, 1960.
[16] X. Zhang and K. Parhi, "High-speed architectures for parallel long bch encoders," *IEEE TVLSI*, 2005.
[17] D. Strukov, "The area and latency tradeoffs of binary bit-parallel bch decoders for prospective nanoelectronic memories," in *ACSSC*, 2006.
[18] Y. Sankarasubramaniam *et al.*, "Energy efficiency based packet size optimization in wireless sensor networks," in *IWSNPA*, 2003.
[19] N. Muralimanohar *et al.*, "CACTI 6.0: A tool to model large caches," HP Laboratories, Tech. Rep., 2009.