ISSN: 2278-4632
Vol-11 Issue-01 2021

MULTIPLE ERROR CORRECTION USING LATTICE ERROR CORRECTOR

[1] Ch Durga Bhavani, Pg. scholar Department of ECE, Vikas Group of Institutions, Nunna, Vijayawada, Mail id: kotieashwar@gmail.com
[2] Y. Uma Maheswari Assistant Professor Department of ECE, Vikas Group of Institutions, Nunna, Vijayawada, Mail id: umaecestaff@gmail.com
[3] S.Kishore babu Associate Professor Department of ECE, Vikas Group of Institutions, Nunna, Vijayawada, Mail id:

ABSTRACT

The reliability of memory subsystem is fast becoming a concern in computer architecture and system design. From on-chip embedded memories in Internet-of-Things (IoT) devices and on-chip caches to off-chip main memories, they have become the limiting factor in reliability of computing systems. This is because they are primarily designed to maximize bit storage density; this makes memories particularly sensitive to manufacturing process variation, environmental operating conditions, and aging-induced wear out. Addressing these concerns is particularly challenging in on-chip caches or embedded memories like scratchpads in IoT devices as additional area, power and latency overheads of reliability techniques in these memories need to be minimized as much as possible. Hence, this dissertation proposes MS-OLS Fault Tolerance in SRAM based scratchpad memories and last level caches. In the first part of the dissertation we propose Difference Set: an approach to deal with known hard faults in software managed scratchpad memories. Difference Set avoids hard faults found during testing by generating a custom-tailored application binary image for each individual chip. During software deployment-time, Difference Set optimally packs small sections of program code and data into fault-free segments of the memory address space and generates a custom linker script for a lazy-linking procedure. The second part proposes two software defined MS-OLS error detection and correction techniques: Software Defined Error Localization Code (SED-DEC) and MS-OLS-ML to recover from soft errors during run time. SED-DEC is mostly for embedded memories and uses novel and inexpensive MS-OLS Error-Localizing Codes (DS-SECs). These require fewer parity bits than single-error-correcting Difference Set codes. Yet our DS-SECs are more powerful than basic single-error-detecting parity: they localize single-bit errors to a specific chunk of a codeword. SED-DEC then heuristically recovers from these localized errors using a small embedded C library that exploits observable side information (SI) about the application's memory contents. MS-OLS-ML is a novel unequal message protection scheme that preferentially provides stronger error protection to certain "special messages". This protection scheme provides Single Error

ISSN: 2278-4632
Vol-11 Issue-01 2021

Detection (SED) for all messages and Single Error Correction (SEC) for a subset of special messages. MS-OLS-ML can be used in both last level caches and MS-OLS embedded memories.

1 .1 INTRODUCTION

Memories are one of the key bottlenecks in the performance, reliability and energy efficiency of most computing systems. As computing systems have scaled over the decades, the need for memory systems where large amount of data can be stored and retrieved efficiently have also risen rapidly. To achieve this, main memory systems have been scaled for maximum information density. Moore's Law has been the primary driver behind the phenomenal advances in computing capability of the past several decades. However, with technology scaling having reached the nanoscale era, integrated circuits, especially memory systems, are becoming increasingly sensitive to process variations leading to reliability and yield concerns.

1.2 MEMORY RELIABILITY IS BECOMING A KEY CONCERN

Memories have become the limiting factor in reliability of computing systems [3] because they are primarily designed to maximize bit storage density; this makes memories particularly sensitive to manufacturing process variation, environmental operating conditions, and aging-induced wearout [4, 5]. Unfortunately, errors in computing memories have also increased. In warehouse-scale computers, these errors have become expensive culprits that cause machine crashes, corrupted data, security vulnerabilities, service disruption, and costly repairs and hardware servicing [3, 6]. Google has observed 70000 failures in time (FIT)/Mb in commodity on-chip DRAM memory, with 8% of modules affected per year [3], while Facebook has found that 2.5% of their servers have experienced memory errors per month [7]. The Blue Waters supercomputer had 8.2% of the dual in-line memory modules (DIMMs) (modules that contain multiple RAM chips) encounter an error over the course of a 261 day study [8]. These trends are expected to continue to rise.

Moreover, with IoT devices increasingly becoming part of critical infrastructure and being deployed in failure-intolerant modes (e.g., cars), development of inexpensive fault tolerance schemes for them has become important [9]. Also, with sensing and data-processing being one of the most important use cases for edge devices, these devices are seeing increasing use of large memories. SRAM based scratchpad memories are often the choice of memory architecture used in IoT devices. As demand for higher memory density increases, memory cells are shrunk using advanced technology nodes which in turn makes the memory cells more susceptible to both soft and hard faults. Need for low-power and hence lower operating voltage exacerbates the error rates further. These trends indicate that memory failures are likewise going to be critical for emerging edge/IoT computing devices as well.

ISSN: 2278-4632
Vol-11 Issue-01 2021

Error-Correcting Codes (ECCs)

ECCs are mathematical techniques that transform message data stored in memory into codewords using a hardware encoder to add redundancy for added protection against faults. When soft faults affect codewords, causing bit flips, the ECC hardware decoder is designed to detect and/or correct a limited number of errors. ECCs used for random-access memories are typically based on linear block codes.

The encoder implements a binary generator matrix G and the complementary decoder implements the parity-check matrix H to detect/correct errors. To encode a binary message ~m, one multiplies its bit-vector by G to obtain the codeword ~c: ~mG =~c. To decode, one multiplies the stored codeword (which may have been corrupted by errors) with the parity-check matrix to obtain the syndrome ~s, which provides error detection and correction information: $H\sim c^T$ =~s. Typical ECCs used for memory have the generator and parity-check matrices in systematic form, i.e., the message bits are directly mapped into the codeword and the redundant parity bits are appended to the end of the message. This makes it easy to directly extract message data in the common case when no errors occur.

1.3 APPROACH

We propose SED-DEC that together form a novel hybrid approach to low-cost embedded memory fault-tolerance. They specifically address the unique challenges posed by SPMs.

The high-level concept is illustrated in Fig. 2.1. At fabrication time, process variation and defects may result in hard faults in embedded memories. During test-time, these are characterized and maintained in a per-chip fault map that is stored in a database for later. When the system developer later deploys the application software onto the devices, Difference Set is used to customize the binary for each individual chip in a way that avoids its unique hard fault locations. Finally, at run-time, unpredictable soft faults are detected, localized, and recovered heuristically using SED-DEC.

Note that Difference Set is not heuristic and therefore does not induce errors. On the other hand, SED-DEC has a chance of introducing silent data corruption (SDC) if recovery turns out to be incorrect; this consideration will be revisited later in the discussion. We briefly explain the approaches of the SED-DEC steps before going into greater detail for each.

1.4 SED-DEC

We describe the SED-DEC architecture, the concept of DS-SEC codes, and two SED-DEC recovery policies for instruction and data memory. Architecture

The SED-DEC architecture is illustrated in Fig. 2.6 for a system with split on-chip instruction and data SPMs (each with its own DS-SEC code) and a single-issue core that has an in-order pipeline. We assume that hard faults are already mitigated using Difference Set.

When a codeword containing a single-bit soft fault is read, the DS-SEC decoder detects and

ISSN: 2278-4632
Vol-11 Issue-01 2021

localizes the error to a specific chunk of the codeword and places error information in a Penalty Box register (shaded in gray in the figure). A precise exception is then generated, and software traps to a handler that implements the appropriate SED-DEC recovery policy for instructions or data, which we will discuss shortly.

Once the trap handler has decided on a candidate codeword for recovery, it must correctly commit the state in the system such that it appears as if there was no memory control flow disruption. For instruction errors, because the error occurred during a fetch, the program counter (pc) has not yet advanced. To complete the trap handler, we write back the candidate codeword to instruction memory. If it is not accessible by the load/store unit, one could use hardware debug support such as JTAG. We then return from the trap handler and re-execute the previously-trapped instruction, which will then cause the pc to advance and re-fetch the instruction that had been corrupted by the soft error. On the other hand, data errors are triggered from the memory pipeline stage by executing a load instruction. We write back the chosen candidate codeword to data memory to scrub the error, update the register file appropriately, and manually advance pc before returning from the trap handler.

Fault-Tolerant Caches

There is an abundance of prior work on fault-tolerant and/or low-voltage caches. Examples include PADded Cache [47], Gated-VDD [48], Process-Tolerant Cache [49], Variation-Aware Caches [50], Bit Fix/Word Disable [51], ZerehCache [52], Archipelago [53], FFT-Cache [54], VS-ECC [55], Correctable Parity Protected Cache (CPPC) [56], FLAIR [57], Macho [58], DPCS [59], DARCA [60], and others (see related surveys by Mittal [61, 4]). These fault-tolerant cache techniques tolerate hard faults/save energy by sacrificing capacity or remapping physical data locations. This affects the software-visible memory address space and hence they cannot be readily applied to SPMs.

Although they are cache-specific, some of the above techniques can be roughly compared with Difference Set in terms of min-VDD. For instance, DPCS [59] achieves a similar min-VDD to Difference Set of around 600 mV, while FLAIR [57] achieves a lower min-VDD (485 mV). We emphasize that the above techniques cannot be applied to SPMs and are therefore not a valid comparison.

Similar to SED-DEC, CPPC [56] can recover random soft faults using SED parity. However, CPPC requires additional hardware bookkeeping mechanisms that are in the critical path whenever data is added, modified, or removed from the cache (and again, their method is not applicable to SPMs).

Performance Overheads

Difference Set does not add any performance overheads because it is purely a link-time solution, while its impact on code size is less than 1%. SED-DEC recovery of soft faults, however, requires about

ISSN: 2278-4632
Vol-11 Issue-01 2021

1500 dynamic instructions, which takes a few ms on a typical microcontroller (the number of instructions varies depending on the specific recovery action taken and the particular DS-SEC code). However, for low-cost IoT devices that are likely to be operated in low-radiation environments with only occasional soft faults, the performance overhead is not a major concern. Simple recovery policies could be implemented in hardware, but then software-defined flexibility and application-specific support would be unavailable.

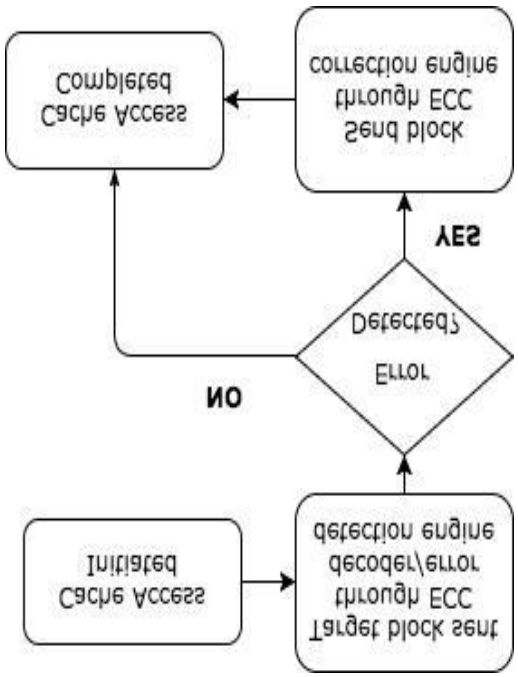
Memory Reliability Binning

Difference Set could bring significant cost savings to both IoT manufacturers and IoT application developers throughout the lifetime of the devices. Manufacturers could sell chips with hard defects in their on-chip memories to customers instead of completely discarding them, which increases yield. Customers could run their applications on commodity devices with or without hard defects at lower-than-advertised supply voltages to achieve energy savings. Fault maps for each chip at typical min-VDDs are small (bytes to KBs) and could be stored in a cloud database or using on-board flash. Several previous works have proposed heterogeneous reliability for approximate applications to reduce cost [70, 71, 72, 73].

Table Error! No text of specified style in document.**-1 Fraction of Special Messages per Benchmark Within Suite**

	Top Two Most Freq Opcodes (Data Memory)		First 6 bits are 0 (Instruction Memory)	
Benchmark Suite	Max	Mean	Max	Mean
AxBench	0.51	0.46	0.92	0.86
SPEC CPU2006	0.56	0.37	0.99	0.89

MS-OLS ERROR CORRECTION CODE



Flow of a read operation in a cache with ECC protection

ISSN: 2278-4632
Vol-11 Issue-01 2021

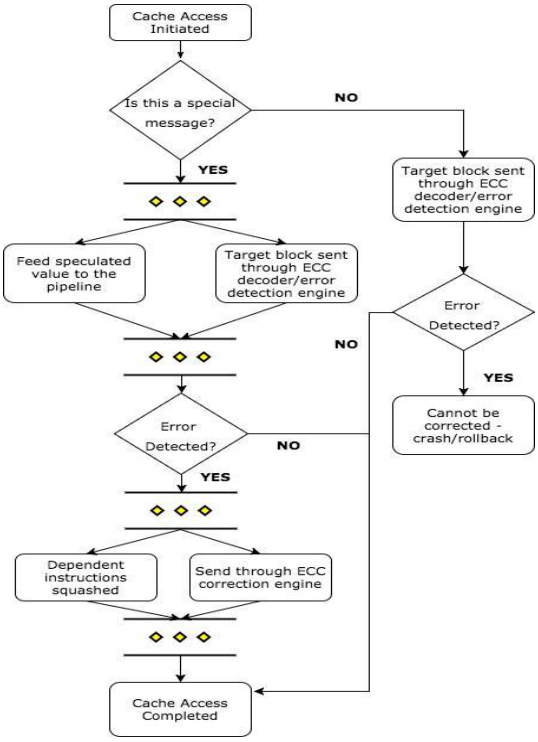


Fig 1.1 Flow of read operation in cache with memory speculation and MS-OLS-ML protection schemes

Additional Cache Support for Speculation

Figure 3.3 depicts the additional circuitry that needs to be added to a traditional cache to support the memory speculation scheme with MS-OLS-ML.

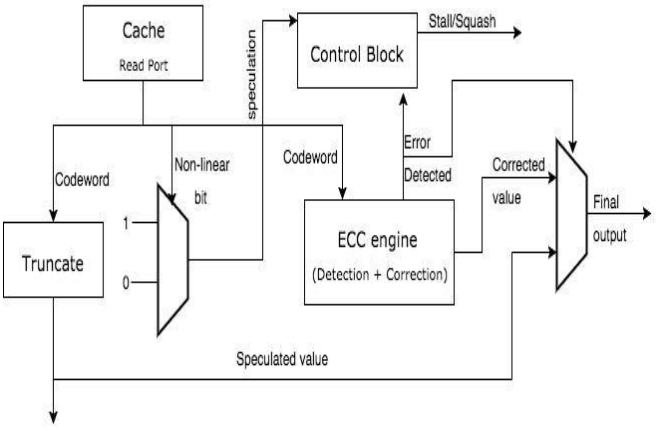


Fig1.2 Cache architecture to implement MS-OLS-ML with memory speculation

The non linear bit is first checked. If it is a special message, then speculation is triggered and the speculated value is forwarded to the next stage. This speculated value comprises of the lower 26-bits of the received codeword to which the special prefix is separately appended. Meanwhile, the decoding and the error detection circuitry works in parallel. If an error is detected, the control module initiates a squash operation to squash all the dependant instructions that used the mis-predicted data and the ECC correction engine provides the correct output. The control module also stalls the pipeline when the non linear bit indicates that the message is not special and hence, the codeword is not systematic. Therefore, speculation cannot be used and the pipeline needs to be stalled for one cycle till the original message is decoded. The stall latency is, of course, greater than one cycle when an error is detected and the ECC correction engine needs to be triggered. This additional control module is simple and has minimal overhead in terms of area and energy.

Storage Overhead

Single-error detection requires only a single parity bit; our Pairty++ scheme adds an additional parity-bit for a total of 2. The most efficient SEC code is the Difference Set code. Assuming our message length, k, is a power of 2, then the number of redundancy bits required for the (shortened) Difference Set code is log(k) +1. Since the Difference Set code has a minimum distance of 3, we can create a SECDED code—the extended Difference Set

ISSN: 2278-4632
Vol-11 Issue-01 2021

code—with the addition of a single parity bit, yielding a total of log(k) + 2 redundancy bits. Similarly, we can use a (shortened) extended BCH code as a DECTED code, with 2 log(k) + 3 redundancy bits.

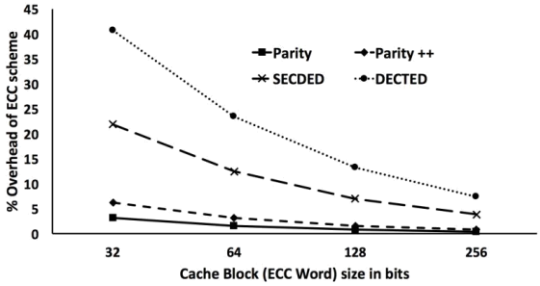


Fig 1.3 Storage overhead of different commonly used ECC schemes along with our scheme MS-OLS-ML

1.5 Experimental Methodology

We evaluated MS-OLS-ML over applications from the SPEC 2006 benchmark suite. Two sets of core micro-architectural parameters (provided in Table 3.3) were chosen to understand the performance benefits in both a MS-OLS in-order(InO) processor and a larger out-of-order(OoO) core. Per-formance simulations were run using Gem5 [99], fast forwarding for 1 billion instructions and executing for 2 billion instructions.

The first processor is a MS-OLS single in-order core architecture with a 32kB L1 cache for instruction and 64kB L1 cache for data. Both the instruction and data caches are 4-way associative. The LLC is a unified 1MB L2 cache which is also 8-way associative. The second processor is a dual core out-of-order architecture. The L1 instruction and data caches have the same configuration as the previous processor. The LLC comprises of both L2 and L3 caches. The L2 is a shared 512kB SRAM based cache while the L3 is a shared 2MB cache which is 16-way associative. For both the baseline processors it is assumed that the LLCs (L2 for the InO processor and L2 and L3 for the OoO processor) have SECDED ECC protection.

The performance evaluation was done only for cases where there are no errors. Thus, latency due to error detection is taken into consideration but not error correction as correction is rare when compared to the processor cycle time and doesn't fall in the critical path. In order to compare the performance of the systems with MS-OLS-ML against the baseline cases with SECDED ECC protection, the size of the LLCs were increased by 10% due to the lower storage overhead of Parity as provided in Section 3.3.4. We call this iso-area since the additional area coming from reduction in redundancy is used to increase the total capacity of the SRAM. The iso-area evaluation was done for both with and without memory speculation. The analysis was also done for the iso-capacity where the memory capacity of the systems with MS-OLS-ML and SECDED remain same and their performances are measured. As mentioned before, SECDED allows speculation in all cases and thus, incurs no additional read latency due to error detection when there is no error. But for MS-OLS-ML, only the special messages are systematic and thus, for all non-special messages, there is an additional one cycle read latency due to the error detection circuitry. This additional latency for

ISSN: 2278-4632
Vol-11 Issue-01 2021

non-special messages was also taken into consideration for our simulations.

1.6 Results & Discussion

In this section we discuss the performance results obtained from the Gem5 simulations (as mentioned in Section 3.4). Figures 3.5 and 3.6 show the comparative results for the two different sets of core micro-architectures across a variety of benchmarks from the SPEC2006 suite when using memory speculation. In both the evaluations, performance of the system with MS-OLS-ML was compared against that with SECDED. The evaluation was further split into iso-area

Simulation results of MS-OLS-MLD are shown in figure 5.2. Here IC(0:7),ID(0:31),R,S(0:7) are inputs and OD(0:31) is the output. The parameters considered for the designed architecture are delay, power and area. Through this approach the delay,area and power consumption successfully reduced.

RTL is an acronym for register transfer level. This implies that Verilog HDL code written based on the architecture describes how data is transformed and how it is passed from register to register. If the simulation and synthesis is done, we have to check for the RTL schematic. We have to click on the RTL schematic double times, and then we will get the basic block diagram of the schematic or our module.

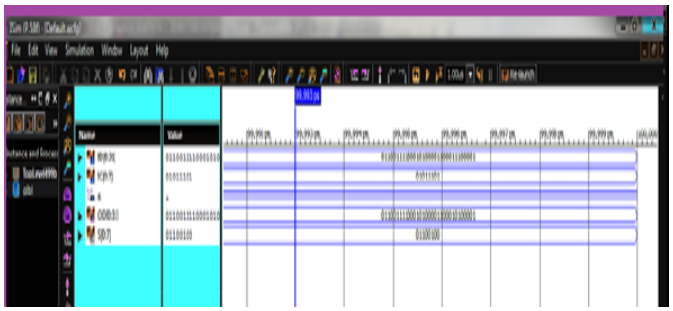


Fig1.4 Simulation Result for MS-OLS-MLD

The detailed view of the RTL schematic of MS_OLS_MLD is shown in Fig. 5.3 (a) and Fig. 5.3 (b). It indicates the internal blocks and connection between the blocks.

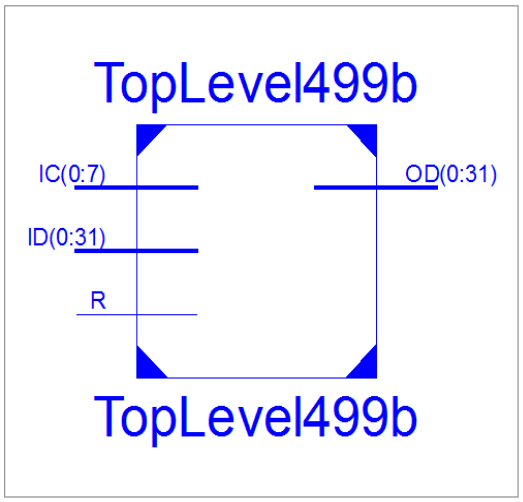


Fig1.5 Top Figure bloc

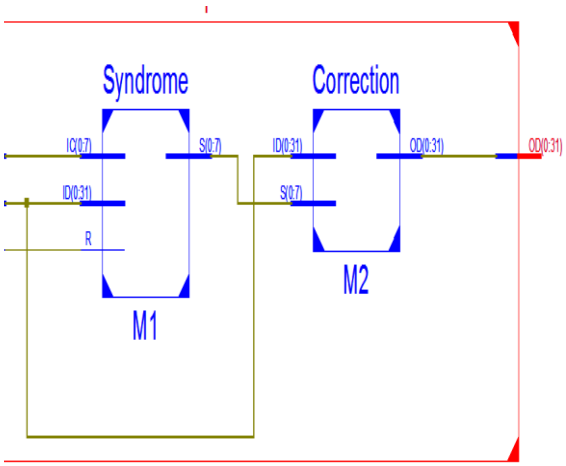


Fig1.6 internal architecture

ISSN: 2278-4632
Vol-11 Issue-01 2021

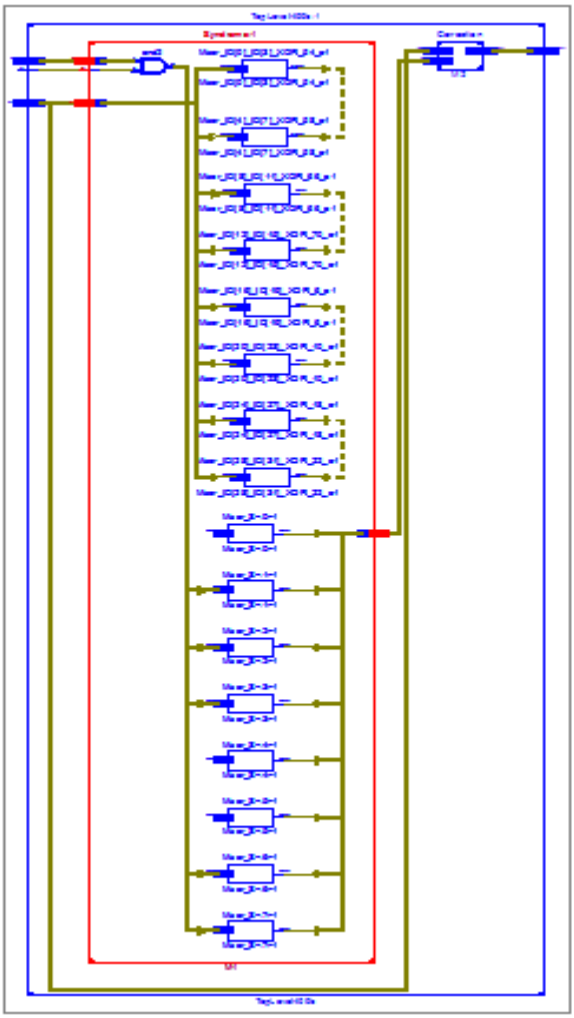


Fig1.7 Complete internal architecture

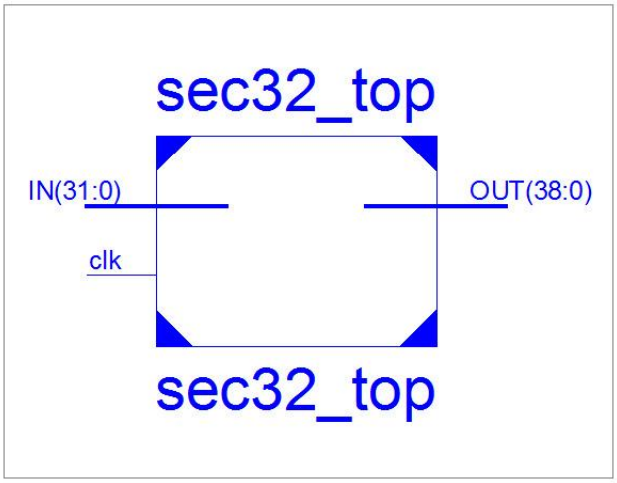
Simulation Results of A Double Error Correction Code For 32-Bit Data Words With Efficient Decoding

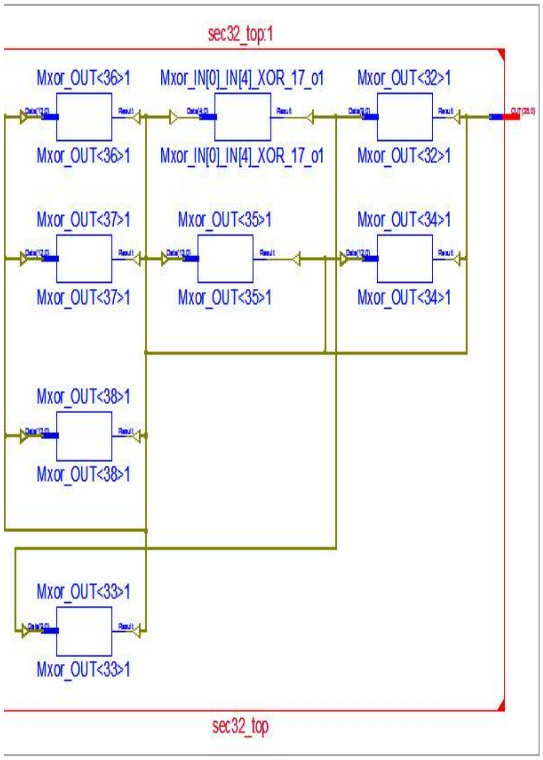
6.3.1 Simulation Results of Encoder

Simulation results of encoder are shown in fig.5.4. Here IN(31:0) are the input and OUT(38:0) is the output. . These are synthesized and simulated using Xilinx ISE 14.7 tool for vertex family device and simulation results as well as synthesis reports are presented.

Fig 1.8 Simulation Results of encoder

The RTL view of the encoder is shown in below fig.5.5. It shows the internal blocks and connections of the architecture.





ISSN: 2278-4632
Vol-11 Issue-01 2021

Fig 1.9 Simulation Results of Decoder

In this section simulation results of a double error correction code for 32-bit data words with efficient decoding shown in fig.5.8. Here IN(38:0), clk, are the inputs and OUT(38:0), SYN(6:0),DBL,ERR,SGL are the output.

ig1.10 Simulation Results of Deco

RTL schematic view of the decoder is shown in below Fig. 5.9 (a) and (b). RTL schematic gives the detailed details of the architecture and internal blocks connections.

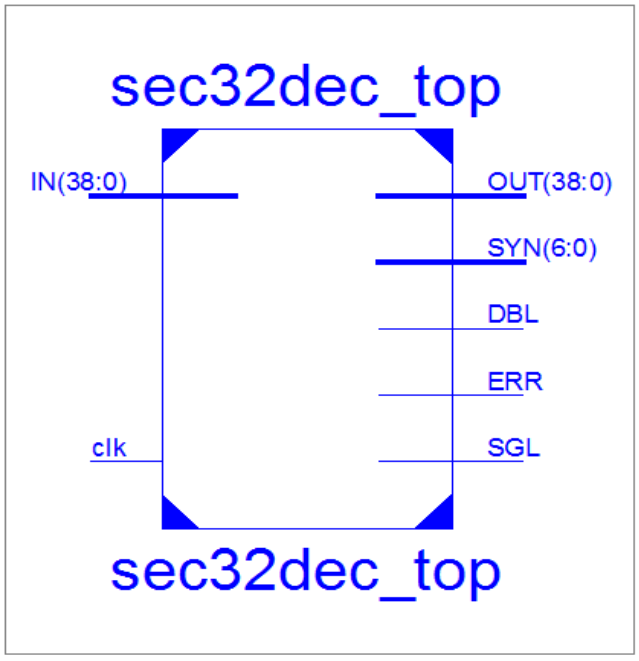


Fig1.11RTL view diagram of decoder

S.No	Parameters	Adouble error correction code for 32- bit data words with efficient decoding	MS-OLS-MLD correction code
1	No:of slice LUT's	76	55
2	No:of occupied slices	43	26
3	No:of Bonded IOB's in %	22%	18%
4	No:of LUT'S Flipflop pairs used	76	55
5	Delay	2.889ns	2.603ns
6	Power	15.311mw	11.484mw

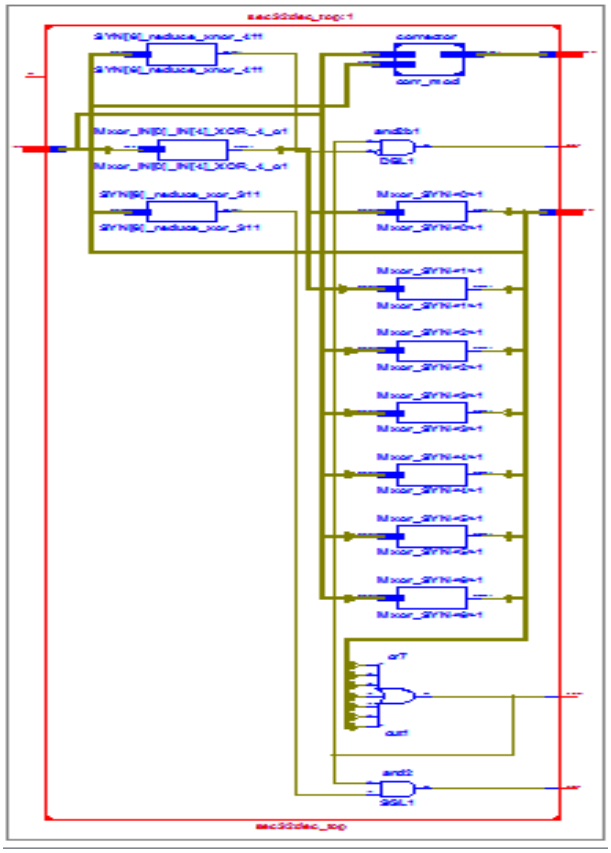
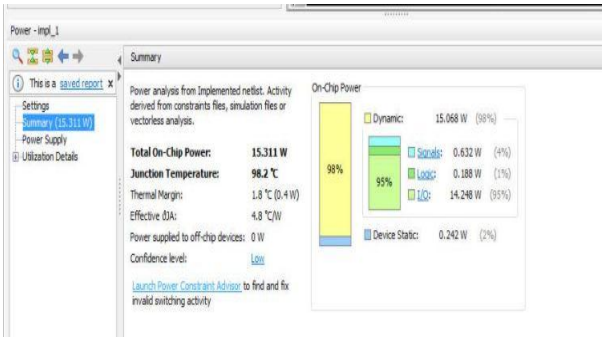


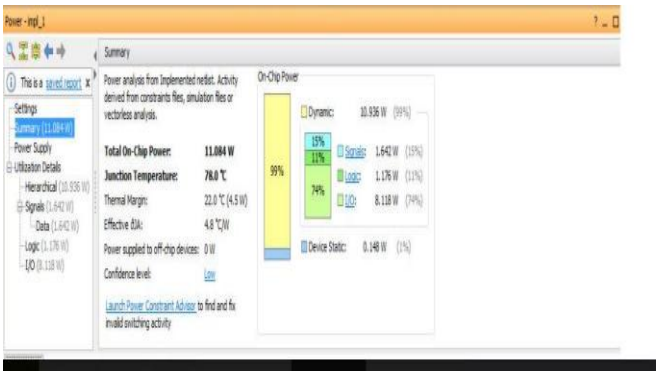
Fig1.12Detailed RTL view diagram of decoder

ISSN: 2278-4632
Vol-11 Issue-01 2021

Table Error! No text of specified style in document.**-2 Design summary of MS-OLS-MLD with A double error correction**

Name of the system	Power(mw)
A double error correction code for 32_bit data words with efficient decoding	15.311
MS_OLS-MLD based double error detection and correction	11.484





code for 32- bit data words with efficient decoding

Table 5.1. Here Number of Slice LUTS , Number of occupied Slices, Number of bonded IOBs , No:of LUT'S flipflop pairs ,Delay, Power are discussed.

Power Report

Table Error! No text of specified style in document.**-3 Power report comparison of existing and proposed MS-OLS-MLD code**

Table.6.2 shows the power comparison between the existing and proposed architecture. Proposed architecture reduces the 3.827mw power than existing architecture, so the performance of the system is increased.

```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 2.889ns
```

Comparisons

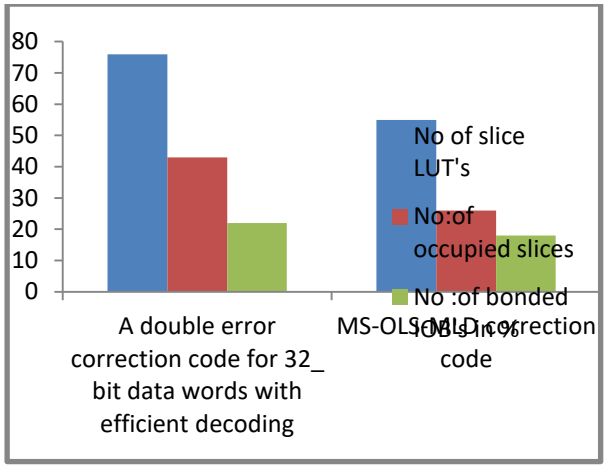
Table 5.3 shows the MS-OLS-MLD correction code is compared with A double error correction code with 32-bit data words with efficient decoding in various parameters like number of slice LUTs, number of occupied slices, number of bonded IOBs, dynamic power, Quiescent power, total power .The implementation results are almost give the same output but power, area is less when

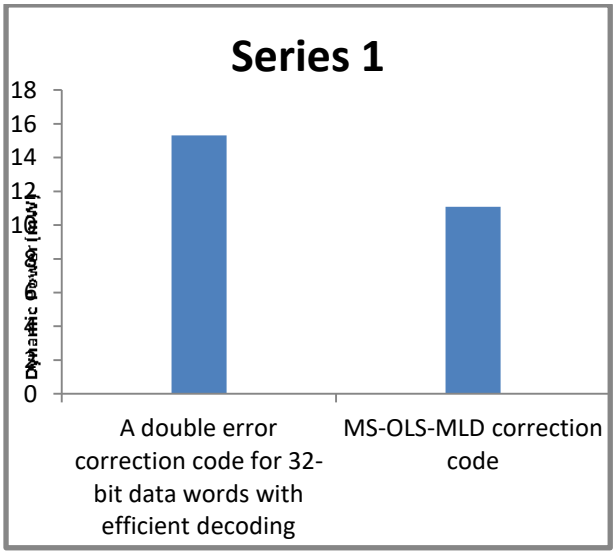
ISSN: 2278-4632
Vol-11 Issue-01 2021

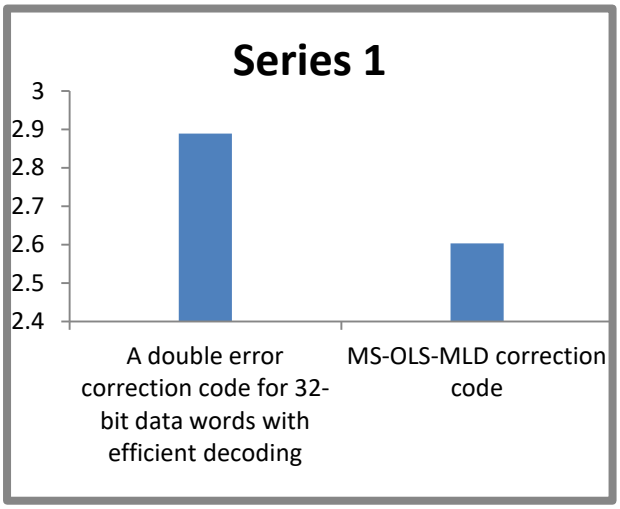
compared to existing work.

Fig.5.12 explains the power comparison between MS-OLS-MLD correction code compared with A double error correction code with 32-bit data words with efficient decoding. Finally it states that the MS-OLS-MLD correction code performance is increased.

Fig.5.13 states the Delay comparision for MS-OLS-MLD correction code and A double error correction code with 32-bit data words with efficient decoding. Finally absorbed that MS-OLS-MLD correction code is less than the existing work .







REFERENCES

M. Gottscho, I. Alam, C. Schoeny, L. Dolecek, and P. Gupta, "Low-cost memory fault tolerance for iot devices," ACM Trans. Embed. Comput. Syst., vol. 16, pp. 128:1–128:25, Sept. 2017.

C. Schoeny, F. Sala, M. Gottscho, I. Alam, P. Gupta, and L. Dolecek, "Context-aware resiliency: Unequal message protection for random-access memories," in 2017 IEEE Information Theory Workshop (ITW), pp. 166–170, Nov 2017.

L. A. D. Bathen and N. D. Dutt, "E-RoC: Embedded RAIDs-on-Chip for Low Power Dis-tributed Dynamically Managed Reliable Memories," in Design, Automation, and Test in Europe (DATE), 2011.

L. A. D. Bathen, N. D. Dutt, A. Nicolau, and P. Gupta, "VaMV: Variability-Aware Memory

ISSN: 2278-4632
Vol-11 Issue-01 2021

Virtualization," in Design, Automation, and Test in Europe (DATE), 2012.

M. Gottscho, L. A. D. Bathen, N. Dutt, A. Nicolau, and P. Gupta, "ViPZonE: Hardware Power Variability-Aware Memory Management for Energy Savings," IEEE Transactions on Computers (TC), vol. 64, no. 5, pp. 1483–1496, 2015.

A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A Mul-tiplatform Benchmark Suite for Approximate Computing," IEEE Design and Test, vol. 34, no. 2, pp. 60–68, 2017.

M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in Proceedings of the IEEE International Workshop on Workload Characterization (IWWC), 2001.

S. Hamdioui, A. J. van de Goor, and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults," in International Workshop on Memory Technology, Design, and Testing (MTDT), 2002.

P. P. Shirvani and E. J. McCluskey, "PADded Cache: A New Fault-Tolerance Technique for Cache Memories," in Proceedings of the VLSI Test Symposium, 1999.

M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2000.

A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy, "A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 1, pp. 27–38, 2005.

M. Mutyam and V. Narayanan, "Working with Process Variation Aware Caches," in Design, Automation, and Test in Europe (DATE), 2007.

C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," in Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA), 2008.

A. Ansari, S. Gupta, S. Feng, and S. Mahlke, "ZerehCache: Armoring Cache Architec-tures in High Defect Density Technologies," in Proceedings of the ACM/IEEE International Symposium on Microarchitecture (MICRO), 2009.

A. Ansari, S. Feng, S. Gupta, and S. Mahlke, "Archipelago: A Polymorphic Cache Design for Enabling Robust Near-Threshold Operation," in Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2011.

A. BanaiyanMofrad, H. Homayoun, and N. Dutt, "FFT-Cache: A Flexible Fault-Tolerant

ISSN: 2278-4632
Vol-11 Issue-01 2021

Cache Architecture for Ultra Low Voltage Operation," in Proceedings of the ACM/IEEE International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2011.

A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes," in Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA), 2011.

M. Manoochehri, M. Annavaram, and M. Dubois, "CPPC: Correctable Parity Protected Cache," in Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA), 2011.

M. K. Qureshi and Z. Chishti, "Operating SECDED-Based Caches at Ultra-Low Voltage with FLAIR," in Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013.

T. Mahmood, S. Hong, and S. Kim, "Ensuring Cache Reliability and Energy Scaling at Near-Threshold Voltage with Macho," IEEE Transactions on Computers (TC), vol. 64, no. 6, pp. 1694–1706, 2015.

M. Gottscho, A. BanaiyanMofrad, N. Dutt, A. Nicolau, and P. Gupta, "DPCS: Dynamic Power/Capacity Scaling for SRAM Caches in the Nanoscale Era," ACM Transactions on Architecture and Code Optimization (TACO), vol. 12, no. 3, p. 26, 2015.

M. Mavropoulos, G. Keramidas, and D. Nikolos, "A Defect-Aware Reconfigurable Cache Architecture for Low-Vccmin DVFS-Enabled Systems," in Design, Automation, and Test in Europe (DATE), 2015.

S. Mittal, "A Survey of Architectural Techniques for Improving Cache Power Efficiency," Sustainable Computing: Informatics and Systems, vol. 4, no. 1, pp. 33–43, 2014.

F. J. Aichelmann, "Fault-Tolerant Design Techniques for Semiconductor Memory Applica-tions," IBM Journal of Research and Development, vol. 28, no. 2, pp. 177–183, 1984.

R. van Rein, "BadRAM: Linux Kernel Support for Broken RAM Modules," 2016.

M. M. Sabry, D. Atienza, and F. Catthoor, "OCEAN: An Optimized HW/SW Reliability Mitigation Approach for Scratchpad Memories in Real-Time SoCs," ACM Transactions on Embedded Computing Systems (TECS), vol. 13, no. 4s, 2014.

H. Sayadi, H. Farbeh, A. M. H. Monazzah, and S. G. Miremadi, "A Data Recomputation Approach for Reliability Improvement of Scratchpad Memory in Embedded Systems," in Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014.

A. M. H. Monazzah, H. Farbeh, S. G. Miremadi, M. Fazeli, and H. Asadi, "FTSPM: A Fault-Tolerant ScratchPad Memory," in

ISSN: 2278-4632
Vol-11 Issue-01 2021

Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013.

F. Li, G. Chen, M. Kandemir, and I. Kolcu, "Improving Scratch-Pad Memory Reliability Through Compiler-Guided Data Block Duplication," in Proceedings of the IEEE/ACM Inter-national Conference on Computer-Aided Design (ICCAD), 2005.

H. Farbeh, M. Fazeli, F. Khosravi, and S. G. Miremadi, "Memory Mapped SPM: Protecting Instruction Scratchpad Memory in Embedded Systems against Soft Errors," in Proceedings of the European Dependable Computing Conference (EDCC), 2012.

D. P. Volpato, A. K. Mendonca, L. C. dos Santos, and J. L. Guntzel," "A Post-Compiling Approach that Exploits Code Granularity in Scratchpads to Improve Energy Efficiency," in Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 127–132, 2010.

A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate Storage in Solid-State Mem-ories," in Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO), 2013.

Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu, "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory," in Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2014.

M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting Partially-Forgetful Memories for Approximate Computing," IEEE Embedded Systems Letters (ESL), vol. 7, no. 1, pp. 19–22, 2015.

A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Approximate Storage for Energy Efficient Spintronic Memories," in Proceedings of the ACM/IEEE Design Automation Conf