http://dx.doi.org/10.21172/ijiet.102.05

RTL Design and Implementation of Erasure Code for RAID system

Chethan.K[1], Dr.Srividya.P[2], Mr.Sivashanmugam Krishnan[3]

[1]*PG Student, Department Of ECE, R. V. College Engineering, Bangalore, India.*
[2]*Associate Professor, Department Of ECE, R. V. College Engineering Bangalore, India.*
[3]*Staff Product Design Engineer Microsemi Storage Solution Pvt LTD Bangalore, India.*

Abstract-Erasure codes is Error Correction scheme that converts data into codes .It is used primarily in distributed storage system for the failure of disks and its recovery. For a set of 'n' data symbols you can have anywhere from 1 to 'k' erasure codes generated. The value of 'k' determines how many failures can be tolerated while still ensuring that no data is lost. Erasure codes are extended version of RAID 6 operation, which is supported through 4 input block descriptor. A study of throughput for raid engine supports Erasure codes for max 10 input buffers and 6 output buffers is optimal. Difference between RAID 6 and erasure code is that Raid 6 has 2 output parities and max 4 input sources whereas the erasure codes has 6 output parities and max 10 input sources. Erasure code is generated in turn using Galois Field (GF) Polynomial Multiplier equation of x8+x4+x3+x2+1and this gives us a prime polynomial of 0x11d. A fully symmetric erasure code allows for as many erasure tokens as data tokens. A "10 of 16" erasure code is synthesized using cadence RTL compiler for RAID system.

Keywords— Erasure code, RTL, GF Multiplier, RAID system, Cadence IES Simulator,SimVison

I. INTRODUCTION

Distributed storage systems represent the backbone of many cloud applications. They span geographically diverse regions in order to serve content to end users with high availability and low latency. The first property, high availability is achieved through redundancy. To that end, distributed storage systems either replicate data across multiple sites or use more elaborate approaches such as erasure coding [1]. A design for sufficient large redundant information in the transmission stream and for the receiver to allow the recover to rebuild the data, even when multiple packets are lost or corrupted in transmission. When applied to data protection, an erasure code takes a relatively large data block and mathematically manipulates it to create multiple smaller encoded blocks to allow the decoding process to recover the data from a partial set of encoded blocks. And it strips large information to create multiple smaller encoded blocks, hence allowing recovery or rebuilding of data at lower time and improving CPU's throughput. A system administrator can define two parameters based on the algorithm are (1) number of encodes blocks to be created (2) minimum number of encoded blocks that will be required to reassemble the data. Main criteria for storage system is to not to lose data when failure happens. Failures do vary from range block in sectors are corrupted; sectors are corrupted in disks or the whole disk been corrupted and unusable. The storage components themselves are protected from certain types of failures. However, when too many bits are flipped, or when physical components fail, the storage system sees this as an erasure: the storage is gone. A typical approach for distributed storage is to provide consistency in data if failures of disk because of concurrent access of disk. Also considering getting good performance with ever scaling of client requirement for storage at low cost. To overcome these difficulties are clearly understood and few are addressed for replication-based storage. Different schemes of erasure code are proposed and tried to implement into our approach [2][3].Erasure Code uses Reed Solomon code for the implementation of correcting codes.

Erasure code is an error-correcting code, which used to detect and correct errors in turn implements its coding using Reed Solomon code. Erasure codes incorporate redundancy in data 'by providing 'k' number of disk recovery as per design specifications. With this redundancy, larger "k" number of disks and rebuilding time in case of recovery will be more. It provides an additional payload to the throughput of the controller. When recovering the data in case of failure, a decoder in the controller first determines whether the received message is valid or not is error detection. Once any error is detected, the decoder provides a valid message to the controller where controller can try to recover using RAID technique or using Erasure code depending on the implementation. Number of corrupted words should not exceed a specific range if the corrupted message was able to transmit across storage systems. Thus, the decoder performs error correction. The number of errors the code can correct depends on the amount of redundancy added [4].

An Erasure code is a block code generally designated as (n, k) with n is number of input data to be stored in disks and k is the number of error correcting code to be stored in the disks. Erasure codes are designed upon on Galois fields (GFs), also called finite fields. The rules of GF arithmetic are different from the usual arithmetic rules. For

instance, GFs are finite fields. To generate and decode RS code of m-bit symbols, an m-bit wide Galois field is used [5]. A GF is a closed set. Every operation, multiplication, division, addition and subtraction of values within the field always results in a value still in the field. Nice. No loss of precision so full reproduction of missing bits is possible. A GF uses a prime polynomial to generate the field. This prime polynomial is an equation that cannot be further reduced or factored; it's in its prime state. For example x2+4x+2 can be factored to (x+2) (x+2). A prime polynomial cannot be factored in this way. There is prime polynomials for GF widths of any power of 2-field size. We choose a prime polynomial of factor 3 as that gives us a byte wide representation. The hex representation of the prime polynomial is the bit positions for each non-zero bit in the equation. We choose an equation of x8+x4+x3+x2+1 and this gives us a prime polynomial of 0x11d.

Table 1: Galois Field Default Primitive Polynomial

Symbol Size ,m	Default Polynomial	Decimal Form
3	x3+x+1	11
4	x4+x+1	19
5	x5+x2+1	37
6	x6+x+1	67
7	x7+x3+1	137
8	x8+x4+x3+x2+1	285

Solving 'n' equations for 'n' unknowns is much easier to do when real numbers are involved. The matrix math used for the solution can often involve quite a few translations and those translations may need to employ fractions to get to the needed precision. We cannot do that here as we are trying to reproduce a single bit and it has no fractional component. If we could then what we'd do is to take the coefficients of the equations and apply a series of transactions such that we can solve for one variable, then use that variable to solve for the next and so on. The generation of erasure codes is simply linear algebra. If you have 'n' unknowns then you need at least 'n' equations to solve. Erasure codes do this. For a set of 'n' data symbols you can have anywhere from 1 to 'n' erasure codes generated. The value of 'n' determines how many failures can be tolerated while still insuring that no data is lost. A elaborated study of throughput was done on giving different input and output as given in Table 2.
So when a "10 of 16" erasure code is discussed this means that there are 16 elements used to store 10 elements of data. The remaining 6 elements are the erasure codes. As there are 6 erasure codes then a 10 of 16 can tolerate up to 6 failures at any given time.

Table 2: Throughput details of various inputs Vs Output on Raid Controller.

EC scheme	6 of 12	8 of 10	10 of 12	10 of 14	10 of 16	10 of 18	10 of 20
Engine Efficiency	0.25	0.87	0.63	0.31	0.21	0.16	1.13
Throughput @all 3 Engines (GB/s)	1.73	4.62	4.33	2.17	1.44	1.08	0.87

In storage industry, with the increase in single disk capacity with fairly constant per-bit error rate, the rate of disk failures increases [6] .Concluded that even a tiny disk failure might lead to a catastrophic failure in RAID system. In addition, among typical RAID systems, RAID-0 has a zero fault tolerance due to lack of redundancy. So implementations such as RAID-3 and RAID-5 was used to tolerate single disk failure [7]. However, with the increase in the number of disks in the storage system, the disk failure probability increases. Once a disk fails in a RAID system, there is a high probability of another disk failure in the same RAID system [8]. Thus, in a storage system with a large number of disks, RAID-5 is not sufficient for reliable storage [9].An (n,k) ECC is employed over a cluster once the data is stored across the nodes. This means that k data symbols from k nodes across the stripe are encoded to get n−k parity symbols that are stored across n−k nodes.
Fault tolerance: Fault tolerance of an erasure code is defined as the number of simultaneous node failures that a DS system can tolerate. For a DS system that employs an (n,k) coding scheme, with minimum distance dmin, its fault tolerance is f = dmin −1
Repair bandwidth: It is defined as the amount of information in bits that needs to be read from the DS system to repair a failed node [10]. More formally, for a DS system storing β symbols per node, the repair bandwidth, γ is defined as

$\gamma = \nu\beta\lambda$

Where ν is the number of bits per symbols and λ is the total number of symbols read for the recovery of a failed symbol. DS systems having low repair bandwidth are preferable as they can repair faster.

Complexity: The term complexity determines the amount of resource required for the completion of a task. This resource may be time, space, operations etc. In this thesis, this resource is the number of operations. Basic tasks such as addition and multiplications of symbols of size ν, require $O(\nu)$ and $O(\nu 2)$ bit operations, respectively [11]. Here, we consider two aspects pertaining to complexity. The first being the encoding complexity, which is defined as the number of operations that needs to be performed to to obtain the parity symbols in each stripe. The second is the repair complexity, which is defined as the number of operations performed to repair a failed symbol. Complexity can be directly translated to hardware costs. Therefore, it is preferable for DS systems to have a low complexity ECC.

II. DESIGN AND ALGORITHM

RTL design does not have any operation of subtraction or addition. Instead of that, a XOR RTL engine was designed for addition and subtraction and are always done by simple XOR of two operands together. Values generated are updated into some translation tables, so that we can represent any byte wide value as a field value. With the field values, we add some additional properties to make things faster and easier. Multiplication and division are a little more difficult. Since we can't allow for overflow we use the fact that A * B can be done by taking the log of A, adding the log of B and then taking the inverse log of the product such that A * B = inverse log(log(A) + log(B)). So using the prime polynomial, we generate the log and inverse log tables. From the code, you can see that where the log and inverse log tables swap index and value.

Pseudo code implementation of GF Multiplier for the polynomial x8+x4+x3+x2+1 as shown below

```
void create_log_tables() {
   int b, log;
   int pp = 0x11d;
   int x_to_w;

   /* Max value is 255, wrap around when we get there. */
   x_to_w = 256;
   b = 1;
   for (log = 0; log < x_to_w-1; log++) {
      gflog[b] = (uint8_t) log;
      gfilog[log] = (uint8_t) b;
      b = b << 1;
      if (b & x_to_w)
         b = b ^ pp;
   }
}
```

```
irun(64): 15.20-s022: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
ncelab: *W,ARCMRA: Elaborating the ECODE_LIB.GF_MUL:RTL, MRA (most recently analyzed) architecture.
        Elaborating the design hierarchy:
        Top level design units:
                :gf_mul(rtl):
        Building instance specific data structures.
        Design hierarchy summary:
                     Instances  Unique
                Components:  2       1
                Processes:   4       4
                Signals:     6       6
        Writing initial simulation snapshot: ECODE_LIB.GF_MUL:RTL
Loading snapshot ecode_lib.gf_mul:rtl ..................... Done
ncsim> source /tools/cds/ies/ies15.20.022.lnx86/tools/inca/files/presrc.tcl
NCM: Wed May  2 16:25:31 IST 2018 1v1525258531blrbb0ab25dp02955 Host=blr1virdt178 JobID=n/a (cad_dd_00989)
Simulation SnapShot is ecode_lib.gf_mul:rtl

ncsim> run
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
Simulation time at exit is: 0 FS
```

Figure 1: Execution result of Cadence IES Simulator for GF Multiplier

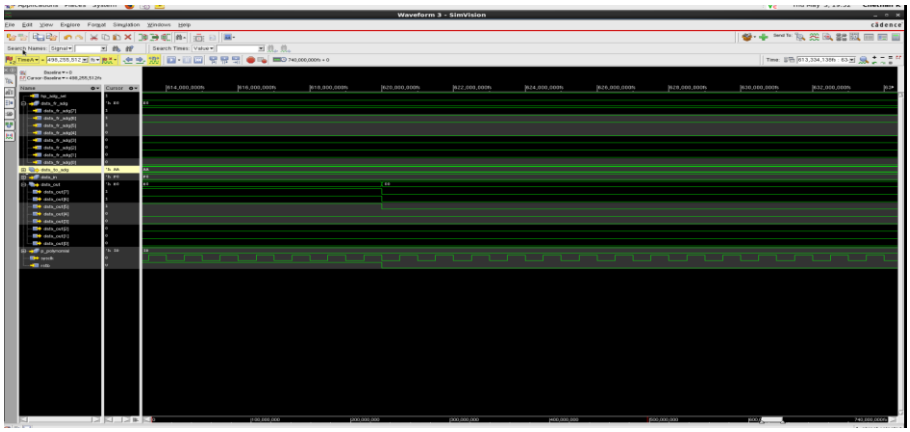

Figure 2: GF Multiplier Simulated results.

Figure 1 indicates execution output of VHDL code using cadence IES simulator. Figure 2 indicates output of GF multiplier by giving 8 input to XOR engine register with primitive polynomial. The creation of erasure codes consists of multiplying data elements by coefficients and then XOR'ing those products together to create an erasure code word. The equation for the erasure codes

ec0 = (g00$\otimes$d0)$\oplus$ (g01$\otimes$d1) $\oplus$(g02$\otimes$d2) $\oplus$ … (g09$\otimes$d9)

ec1 = (g10$\otimes$d0)$\oplus$ (g11$\otimes$d1) $\oplus$(g12$\otimes$d2) $\oplus$ … (g19$\otimes$d9)

ec2 = (g20$\otimes$d0)$\oplus$ (g21$\otimes$d1) $\oplus$(g22$\otimes$d2) $\oplus$ … (g29$\otimes$d9)

ec3 = (g30$\otimes$d0)$\oplus$ (g31$\otimes$d1) $\oplus$(g32$\otimes$d2) $\oplus$ … (g39$\otimes$d9)

ec4 = (g40$\otimes$d0)$\oplus$ (g41$\otimes$d1) $\oplus$(g42$\otimes$d2) $\oplus$ … (g49$\otimes$d9)

ec5 = (g50$\otimes$d0)$\oplus$ (g51$\otimes$d1) $\oplus$(g52$\otimes$d2) $\oplus$ … (g59$\otimes$d9)

$\otimes$: GF Multiplcation

$\oplus$: XOR operation

g'xy' : GF co-efficient

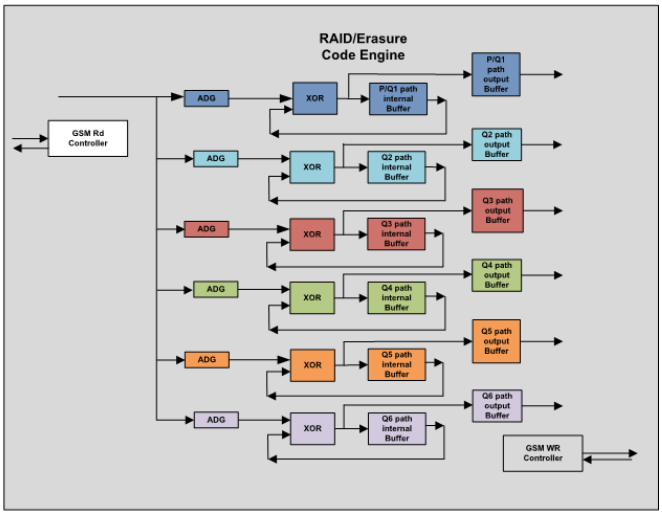

Figure 3: Block Diagram of Erasure code

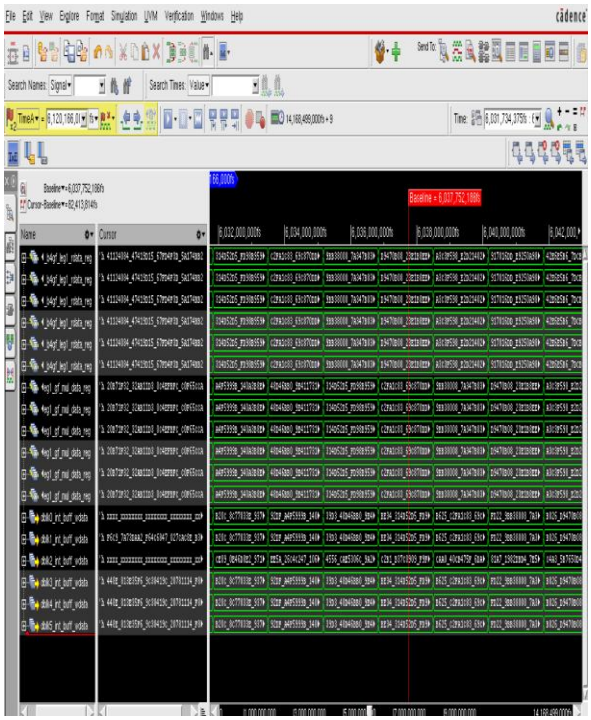

Figure 4: Erasure Code Simulated Results

III. CONCLUSION

RTL design for the erasure code is designed according to Figure 3 with input data pattern is given to ADG Block (GF Multiplier primitive polynomial [x8+x4+x3+x2+1]). An output of ADG block is XOR Block with 8 bit zeroes and saved into Input Register Buffer. Next input data is multiplied and the output is given to XOR Block to generate intermediate erasure code. At 2nd iteration, with next input data pattern is multiplied by ADG block and product generated is XOR with intermediate erasure code. In our method, same procedure repeated for 9th iteration for 10 data input, a single erasure code is generated. And with different sequence of different GF coefficient for 6 iteration, 6 erasure code is generated for same 10 input data pattern. A GF multiplier of 128 bit is generated and each GF coefficient is selected according to input data pattern selection algorithm to generate a unique erasure code. A "10-16" erasure code is verified with simulated results for 10 input data passed through our RTL module block.

VI. REFERENCES

[1] H. Weatherspoon and J. Kubiatowicz, “Erasure coding vs. replication: A quantitative comparison,” in IPTPS. Springer-Verlag, 2002

[2] S. Frolund et. al., “A decentralized algorithm for erasure-coded virtual disks,” in Proceedings of DSN, 2004

[3] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter, “Efficient byzantine-tolerant erasure-coded storage,” in Proceedings of DSN, 2004

[4] Rorabaugh, C. Britton. Error Coding Cookbook. McGraw-Hill, 1995

[5] Sweeney, Peter. Error Control Coding. John Wiley & Son,2002

[6] M. Schulze, G. Gibson, R. Katz, and D. Patterson. How Reliable is a RAID? In Digest of Papers 34th IEEE Computer Society International Conference: Intellectual Leverage, 1989

[7] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In Proceedings of the ACM SIGMOD/PODS Conference, 1988

[8] M. Blaum et al. EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures. IEEE Transactions on Computers, 1995

[9] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-Diagonal Parity for Double Disk Failure Correction. In Proceedings of the 3rd USENIX Symposium on File and Storage Technologies, 2004

[10] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” IEEE Trans. Inf. Theory, vol. 56, no. 9, pp. 4539–4551, September. 2010

[11] J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, 5th ed. CRC Press, August. 2001