Asian Journal of Information Technology 15 (17): 3366-3375, 2016
ISSN: 1682-3915

Fault Diagnosis in Ternary Content Addressable Memory with Secured and Improved Decoding Algorithm

[1]K. Mathan, [2]T. Ravichandran, [3]D. Mahesh Kumar and [3]R. Kannusamy
[1]Department of Electrical and Electronics Engineering,
Hindusthan College of Engineering and Technology, Coimbatore, India
[2]Department of Electronics and Communication Engineering,
SNS College of Technology, Coimbatore, Iindia
[3]Department of Electronics, PSG College of Arts and Science, Coimbatore, India

Abstract: In order to shield memories against MCUs (Multiple Cell Upset) as SEUs (Single Event Upset) an advanced fault detecting and correcting codes called Low Density Parity Check (LDPC) codes is devised. It is found to correct one fault per word which belongs to the family of the majority logic decoding recently projected for Ternary Content Addressable Memory (TCAM) memory application and Difference Set Cyclic Codes (DSCC). ML (Majority Logic) decodable codes measure appropriate Ternary Content Addressable Memory (TCAM) applications because of their capability to correct an outsized variety of faults. The present parity check algorithm on ML decodable codes optimizes the chip area overhead. But our projected idea for fault diagnosis algorithm makes significant Ternary Content Addressable Memory (TCAM) space overhead. Compared with the prevailing technique used for scaling back the decoding time through hybrid codes, our proposed technique offers promising choice for (TCAM) applications. It will reduce extra overhead in decoding algorithms specially designed for TCAM applications which intend to enhance the error-detection and correction capabilities. HDL (Hardware Description Language), simulation and synthesis results are enclosed herewith show that the projected techniques will be expeditiously analyzed in Spartan 6 low power FPGA for analyzing parameter in terms of TCAM, timing and power.

Key words: CAM, Difference Set Cyclic Codes (DSCC), fault correction codes, majority logic decoding, Ternary Content Addressable Memory (TCAM), Multiple Cell Upsets (MCUs), FPGA (Field Programmable Gate Array)

INTRODUCTION

RADIATION-INDUCED soft faults are a major issue for TCAM reliability. To prevent soft faults from causing data corruption, TCAM are typically protected with Fault Correction Codes (FCCs) (Reed, 1954). The most commonly used codes can correct one fault and detect two faults per TCAM memory word and are known as Single-Fault-Correction Double-fault-Detection (SFC-DFD) codes (Massey, 1963). The method used here requires few additional bits per word and that the decoding process will become very simple. A SFC-DFD code enforces a minimum distance of four, between, any two coded words, by having a distance of four. Any word that suffers a double fault would be in the worst case, at a distance of two from any valid coded word. Therefore, it cannot be mistaken for a single fault and miscorrected (Reed, 1954). The same approach is used for codes that can correct two faults; in this case, Double Fault Correction Triple Fault-Detection (DFC-TFD) codes are used. However, this increases the decoder complexity substantially. In a hierarchical approach that combines a Hamming code and a (Bose and Chaudhuri, 1960). Bose-Chaudhuri-Hocquenghem code was proposed to minimize the latency. The use of Euclidean Geometry (EG) codes has also been considered for TCAM protection. The particular EG codes studied, are one-step majority logic decodable and therefore, the decoders can be implemented with low cost. Other codes that are one-step majority logic decodable are Difference-Set (DS) codes. Their use for TCAM protection has also been studied recently, showing that the properties of the codes can be exploited to reduce the decoding time signi?cantly. The combination of a simple decoder and reduced decoding time makes DS codes an attractive option for TCAM protection (Mao and Banihashemi, 2001; Hemati and

Coresponding Author: K.Mathan, Department of Electrical and Electronics Engineering,
Hindusthan College of Engineering and Technology, Coimbatore, India

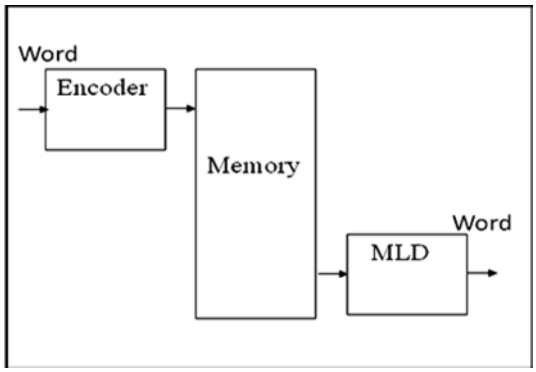

Fig. 1: Memory system schematic with MLD (Majority Logic Decoding)

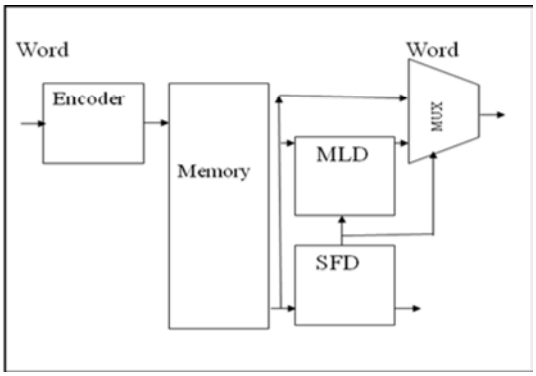

Fig. 2: Memory system schematic of an ML decoder with SFD (Syndrome Fault Detector)

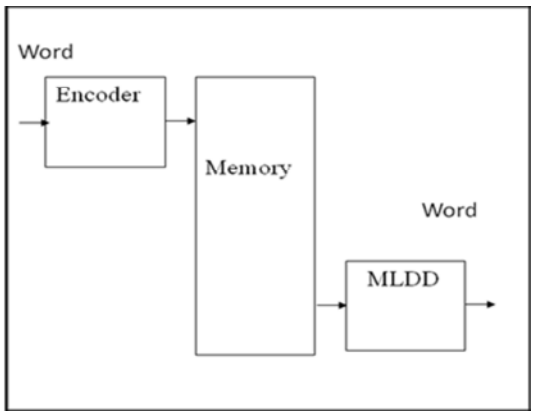

Fig. 3: Memory system schematic of an MLDD

Banihashemi, 2006) which belongs to the family of the ML decodable codes of LDPC. In this paper, the focus is on one specific type of LDPC codes, namely the Difference-Set Cyclic Codes (DSCCs), (Weldon, 1966) which is widely used in the Japanese teletext system or FM (Frequency modulation) multiplex broadcasting systems.

Profile about MLD (Majority Logic Decoding): The MLD relies on variety of redundant check equations that are orthogonal to every alternative, so that, at every iteration, every code-word bit solely participates in one redundant check equation, except the terrible first bit that contributes to all or any equations. For this reason, the majority results of these redundant check equations decide the correctness of this bit below decoding. A generic schematic of a memory system is delineated in Fig. 1 for the usage of associate ML decoder. During this methodology, at first the information words are encoded then kept within the memory of the ensuing sums and are then forwarded to the majority gate for evaluating its correctness. If the amount of 1's received is larger than the amount of 0's, it means that this bit below decoding is wrong and an indication to correct it might be triggered. Otherwise, the bit below decoding would be correct and no further operations would be required thereon and it will increase decoder. But, they need an oversized decoding time that impacts memory performance so as to enhance the decoder's performance, different designs is also used. One chance is to feature a fault detector by scheming the syndrome, so solely faulty codewords are decoded. Since most of the codewords are fault-free, no correction is required and so performance won't be affected. Though the implementation of associate SFD (Syndrome Fault Detector) reduces the typical latency of the decoding method, it additionally adds complexness to the look (Fig. 2). The SFD is associated with XOR matrix that calculates the syndrome supporting the redundant check matrix. Every bit leads to a syndrome equation. Therefore, the complexness of the syndrome calculator will increase with the scale of the code.

The scalable ML (Majority Logic) decoder that improves the styles conferred before ranging from the first style of the ML decoder introduced in ML Detector/Decoder (MLDD) has been enforced. Here, the Difference-Set Cyclic Codes (DSCCs) code is an element of the LDPC codes (Fig. 3).

In all higher than strategies though they're economical in several attributes, there are some drawbacks adding further bit for redundant check equations that successively provides further overhead to the whole method. Those strategies created use two-dimensional redundant check equations for fault detection and ML decoders for fault correction and this makes an enormous impact on the performance of the system, counting on the scale of the code. This explores the thought of mistreatment of two dimensional checksum for fault detection so as to scale back further overhead, to

(a) (b)

$$H=\begin{bmatrix}1&0&0&1&0&1&1\\0&1&0&1&1&1&0\\0&0&1&0&1&1&1\end{bmatrix} \quad H=\begin{bmatrix}0&1&0&1&1&0&0&1\\1&1&1&0&0&1&0&0\\0&0&1&0&0&1&1&1\\1&0&0&1&1&0&1&0\end{bmatrix}$$

Fig. 4: Parity check Matrix for: a) DSCC; b) LDPC

Table 1: DSCC lengths for existing and proposed algorithms

	Data (D)	Codeword (C W)	Segments (S)
Two dimensional parity	32	45	4
	64	90	8
	128	180	16
Two dimensional checksum	32	40	4
	64	72	8
	128	136	16

enhance the performance of the system and for codeword construction through hybrid codes by combining the DSCC and LDPC (Hemati and Banihashemi, 2006; Xiao *et al*., 2008; Xiao and Banihashemi, 2009) technique only for code construction. Also the corresponding DSCC lengths are delineate in Table 1. In LDPC, the sparseness of Parity check Matrix(H) which guarantees constant weight with respect to rows and columns (but less than non zero terms) but in case of (Shannon, 1948; Hamming, 1950) DSSCs, more number of non zero terms in turn increase decoding time as shown in Fig. 4.

Existing algorithm of mldd based on DSCC: The central downside of communication theory is to construct associate secret encoding and a decoding system that creates a noisy channel. The secret decoding system uses the supply knowledge to pick out a codeword from a collection of codewords. The decoding algorithm ideally infers, the given output of the channel, that codeword within the code is that it can possibly have been transmitted; for associate applicable definition of distance, this is often the nearest codeword to the received signal. A good code is one within which the Codewords (C_W) are well spaced apart, so that codewords are unlikely to be confused. In the existing algorithm a check bit or a Parity bit (P) could be a bit additional to the top of a string of computer code that indicates whether or not the amount of bits within the string with the worthy one is even or odd. Parity bits area unit is used because the simplest sort of error policy is work code. There are two variants of check bits: even check bit and odd parity bit. Just in case of even parity, the check bit is ready, if the amount of ones in a very given set of bits (not together with the parity bit) is odd, creating the amount of ones within the entire set of bits (including the parity bit) even. If the amount of ones in a

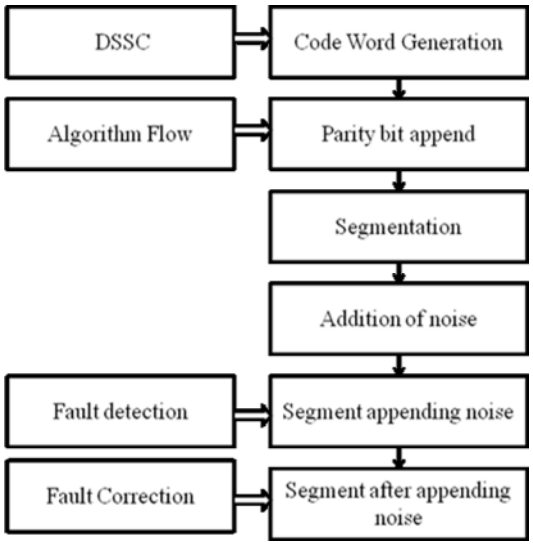

Fig. 5: Flowchart of the two dimensional parity flow

very given set of bits is already even, it's set to a zero. Once mistreatment of odd parity takes place, then the check bit is ready to one if the amount of ones in a very given set of bits (not together with the parity bit) is even, keeping the amount of ones within the entire set of bits (including the parity bit) odd. Once the amount of set bits are odd, then the odd check bit is ready to zero. In case, if the data is corrupted due to noise it must be diagnosed through the parity check analysis at the output node. If the parity matches at the output it accepts and stores data as per algorithmic flow or else it will correct until the Final Data (FD). This reflects the original transmitted Data (D) which is done through hard decision decoding (majority logc decoding) which helps to retrieve the original data through flipping based on number of fault occurred at the output (as shown in algorithm 1). But in case of existing architecture they use XOR for correction (Fig. 5). In the projected method correction part is optimized using inverter which reduces the complexity of the architecture (Fig. 6).

The MLD is based on a number of parity check equations which are orthogonal to each other, so that at each iteration, each codeword bit only participates in one parity check equation, except the very first bit which contributes to all equations. For this reason, the majority result of these parity check equations decide the correctness of the current bit under decoding. MLD was first mentioned by Xiao and Banihashemi (2009) for the Reed-Muller codes. Then, it was extended and generalized by Shannon for all types of systematic linear block codes that can be totally orthogonalized on each codeword bit. A generic schematic memory system is

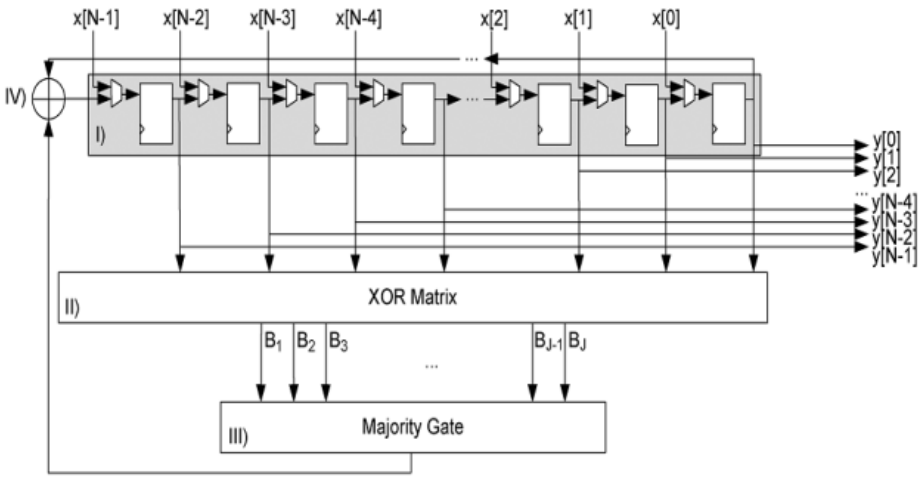

Fig. 6: Existing Schematic of an ML decoder: I) cyclic shift register; II) XOR matrix (Parity bit); III) Majority gate; IV) XOR for correction

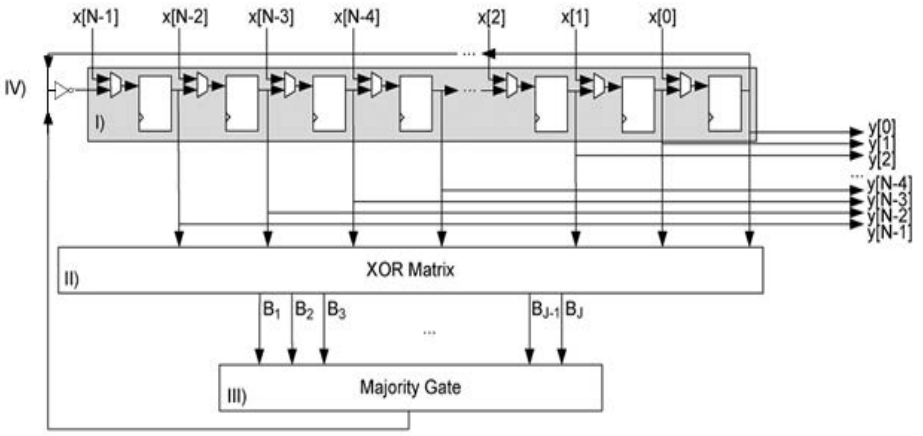

Fig. 7: Existing Schematic of an ML decoder: I) Cyclic shift register; II) XOR matrix (Parity bit); III) Majority gate; IV) NOT for correction

depicted in Fig. 7 for the usage of an ML decoder. Initially, the data words are encoded and then stored in the memory. When the memory is read, the codeword is then fed through the ML decoder before sending to the output for further processing. In this decoding process, the data word is corrected from all bit-flips that might have suffered while being stored in the memory.

Algorithm 1: Code construction and parity check equations

Initialize H, G
Initialize P, I
If (H=satisfied) then
H<=valid
Else
H<=invalid
End if
If (G=satisfied) then
G<=valid
Else
G<=invalid
End if
If (code=check node) then
s<=code append '0'
Else
s<=code append '1
End if

Thus, for a computer code with m parity-check constraint associated length n codewords the parity-check matrix is an m×n binary matrix (Fig. 4). In matrix kind a string code word C = [c1 c2 c3 c4 c5 c6] could be a valid code word for the code with parity-check

matrix H and if providing it satisfies the matrix equation CH^t = zero based on parity binary bits are appended with codeword before transmission (Algorithm 1). This method is recurrent till all the parity-check equations are satisfied or till some most range of decoder iterations has passed and also the decoder. The ML decoding decoder is instantly terminated whenever a sound codeword is found by parity-check equations if all the Checksum equations are satisfied.

MATERIALS ANDS METHODS

Proposed code construction algorithm based on hybridcodes: Within the subfield of numerical analysis, a thin matrix could be a matrix inhabited primarily with zeros as components of the table. The term itself was coined by Harry M. Markowitz. If the majority of components take issue from zero, then it's common to consult with the matrix as a dense matrix. Storing and manipulating thin matrices on a PC, is useful and sometimes necessary to use specialized algorithms and knowledge structures that cash in the thin structure of the matrix. Operations mistreat normal dense-matrix structures and algorithms are comparatively slow and consume giant amounts of memory once applied to giant thin matrices. It's knowledge is naturally simply compressed and this compression nearly always leads to considerably less PC knowledge storage usage. Indeed, some terribly giant thin matrices are unfeasible to control mistreatment in normal dense algorithms. The native organization for a matrix could be a two-dimensional array. Every entry within the array represents a component i, j of the matrix and may be accessed by the 2 indices i and j. In general, 'i' indicates the row range (top-to-bottom) whereas 'j' indicates the column range (left-to-right) of every part within the table. For associate m×n matrix, enough memory to store (m×n) entries to represent the matrix is required. Substantial memory demand reductions are complete by storing solely the non-zero entries. Counting on the amount and distribution of the non-zero entries, completely different knowledge structures is used and yield vast savings in memory in comparison to a native approach. Formats are divided into 2 groups: those that support efficient modification and those that support efficient matrix operations. The efficient modification group includes DOK (Dictionary of Keys), LIL (List of Lists) and COO (Coordinate List) and is typically used to construct the matrix. Once the matrix is constructed, it is typically converted to a format, such as CSR (Compressed Sparse Row) or CSC (Compressed Sparse Column) which is more efficient for matrix operations. From that it clearly shows sparse terms to be a crucial parameter whereas constructing a block code, however within the case of DSCCs codes isn't taken in to account whereas constructing block of codes, successively increases further overhead of the Look Up Table (LUTs) (Table 2). As a result of this it must store non-zero terms. It's to overcome via construction of code through LDPC. The projected research combines LDPC and DSCCs as a single algorithm named as hybrid codes (Naeimi and DeHon, 2009; Torrens *et al*., 2010).

Proposed two dimensional checksum: Checksum or hash sum is a small-size data sum computed from an arbitrary block of digital data for the purpose of detecting errors that may have been introduced during transmission or storage. The integrity of the data can be checked at any later time by re-computing the checksum and comparing it with the stored one. If the checksums match, the data is likely not altered. The procedure that yields the checksum from the data is called a checksum function or checksum algorithm. A good confirmation formula can yield a special result with high likelihood once the information is accidentally corrupted; if the checksums match, the information has identical high likelihood of being freed from accidental errors. In standard addition a variant of the previous formula is to feature all the "words" as unsigned binary numbers, discarding any overflow bits and append the two's complement of the overall because of the confirmation. To validate a message, the receiver adds all the words the same manner, together with the checksum; if the result is not a word filled with zeros, a mistake should have occurred. This variant too detects any single-bit error, however the likelihood that a two-bit error can go undiscovered could be a very little but 1/n. (Fig. 8). The knowledge N (Data) is split into n segment (Equally divided $N = 2^n$). The Segments (St) are extra mistreatment ones complemented arithmetic to induce the total and corresponding output is complemented to induce the Checksum, then segmental Checksum is shipped alongside knowledge segments. All the received Segments (Sr) are extra mistreatment ones complementing arithmetic to induce the total. Then, the total (Ss) must be complemented, if the result is zero, the received knowledge is accepted; otherwise rejected (Algorithm 2).

Earlier for the code construction was done through parity equations (Algorithm 1). In case of the proposed algorithm, it is noted that a generator matrix for a code with parity-check matrix H can be found by performing Gauss-Jordan elimination (Viterbi, 1967; Hemati and Banihashemi, 2006) on H to obtain it in the form:

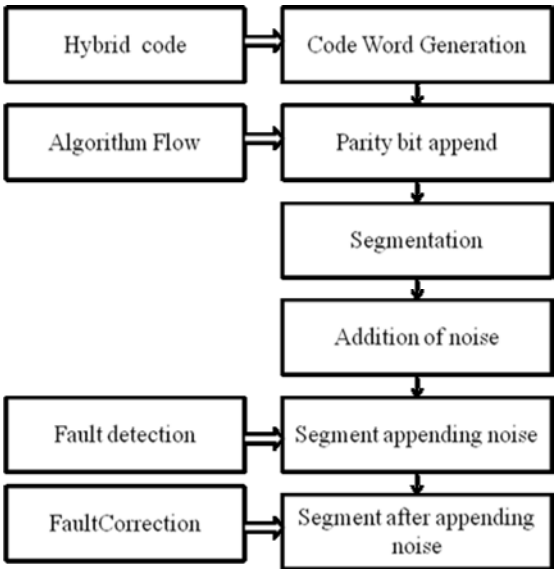

Fig. 8: Flowchart of the checksum design flow

$$H = [P^T | I_{n-k}] \quad (1)$$

where, P Parity matrix and n is size of the data in which is (n-k) (Check bit)×k (information bit) binary matrix and I_{n-k} is the identity matrix. The generator matrix (G) is then

$$G = [I_k | P] \quad (2)$$

The matrix H is called a parity-check matrix. Each row of H corresponds to a parity-check equation and each column of H corresponds to a bit in the codeword:

Algorithm 2; code construction and two dimensional check sum equations:

Initialize H, G
Initialize P, I
Check ($H=P^T|I_{n-k}$) then
H<=valid
Else
H<=invalid
End check
Check ($G=I_k|P$) then
G<=valid
Else
G<=invalid
End check
Initialize
Code(C) =GH^t=0
If (code=valid)
Else
Invalid (Repeat for valid Check)
End Check
Data segmentation
Check For: 1: N do
Segment the N
S_t (N) <=s1(n) mod 2 addition s2 (n)
Invert the checksum S_t(N)
S_r(N)<= s1(n) mod 2 addition s2 (n)
Mod 2 addition of S_t(N) and S_r(N)
S_s obtained _invert the result
Finished
End check

The ML decoding algorithm (Ibe *et al*., 2010; Liu *et al*., 2012) could be a hard-decision message-passing algorithm for hybrid codes. A binary (hard) call concerning every received bit is created by the detector and this is often passed to the decoder. For the ML decoding algorithm the messages passed on the tanner graph edges also are binary: a little node sends a message declaring if it's a 1 or a zero and every check node sends a message to every connected bit node, declaring what worth the bit relies on the knowledge obtainable to the check node. The check node determines that its Checksum equation is satisfied if the modulo-2 addition of the incoming bit values is zero. If the majority of the messages received by a little node are different from its received worth the bit node changes (fips) its current worth. This method is recurrent till all the check equations are satisfied or till some most range of decoder iterations has passed and also the decoder provides up. The Majority decoding (Ibe *et al*., 2010; Liu *et al*., 2012) decoder is instantly terminated whenever a sound codeword has been found by checking if equations are good.

In hard-decision algorithms, solely binary values are used throughout the decoding method. These algorithms are very necessary because of their straightforward implementation. Their binary structure, binary recollections and restricted wiring, makes them exceptional in hardware implementation particularly within the things wherever solely hard-decision values are obtainable at the receiver. Gallager's Algorithm (GA) is an example, a hard-decision decoder within the set of majority primarily based algorithms and is studied by the utilization of density evolution. Another example of hard-decision algorithms is Majority decoding algorithm. During this algorithm a flipping operates is outlined that counts the amount of check nodes within where every variable node participates. Every variable node contains a binary buffer to store a tough call value; the content of this buffer is flipped if the corresponding output of the flipping operate is quite a definite threshold. Decoding continues till all check node equations are happy or till most range of iterations is reached. Majority Logic Estimation (MLE) is a crucial tool in deciding the particular possibilities of the assumed model of communication.

In reality, a communication is quite complicated and a model becomes necessary to alter calculations at

decoder facet. The model ought to closely approximate the complicated communicating. There exists a myriad of ordinary applied math models that may be used for this task; mathematician, binomial, exponential, geometric, poisson, etc., a customary communication model is chosen to support empirical knowledge. Every model mentioned has a higher distinctive parameter that characterizes them. Determination of those parameters for the chosen model is critical to form them closely models the communicating at hand. Suppose a binomial model is chosen (based on observation of data) for the fault events over a selected channel, it's essential to work out the likelihood (p) of the binomial model. If a mathematician model (normal distribution) is chosen for a selected channel then estimating (mean) and (variance) are necessary so they will be applied whereas computing the contingent probability of p(y received | x sent) equally estimating lambda could be a necessity for a statistical distribution model. Most chance estimation could be a methodology to work out these unknown parameters related to the corresponding chosen models (Algorithm 3).

Algorithm 3: ML decoding for fault correction:

For i=1 to m do
S_t=S_r
End for
Repeat
For i=1 to m do
If (S_t=S_r) then
All the values known
Finished else
(S_t?S_r)
E_r belongs to S_s
Flip E_r corresponds to S_s
Finished
End if
End for

$$P(y \text{ received}|\times \text{Send}) = (1-p)^{n-d}\, p^{d} \quad (3)$$

Where:

d = The hamming distance between the received and the sent codewords
n = Number of bit sent.
p = Fault probability of the BSC
1-p = Reliability of BSC

The planned theme for fault diagnosis methodology significantly makes space overhead stripped and to scale back the decoding time through Hybrid codes than the present technique and it provides promising possibility for (Torrens *et al*., 2010) memory applications.

RESULTS AND DISCUSSION

The HDL (Hardware Description Language) simulation and synthesis results are enclosed showing that the planned techniques is expeditiously analyzed and sculptured in Spartan half-dozen low power FPGA.

Table 2: Logic utilization for various algorithms

Logic utilization	Existing method Parity (1D)	Existing method Parity (2D)	Proposed method checksum	Available resource
LUT's	40	24	17	46560
I/O Buffers	88	70	71	4896

Area/Timing analysis: The earlier subsection showed that the performance of the proposed design MLDD is much faster than the plain MLD version but somewhat lower than the design with Syndrome Fault Detector (SFD). As mentioned numerous times, this is compensated with a clear savings in area. To study this, the three designs have been implemented in VHDL and synthesized, for different values of N. The conclusions on the area results are given as follows:

- The MLD design requires a little area compared to the other two designs. However, as seen before, the performance results are not very good
- The SFD version which had the best performance, needs more area than the MLD does. Notice that the increment of area grows quicker than does
- The existing MLDD version based on parity has a very similar performance to SFD. However it requires a much lower area overhead, ranging from 20-30%
- The Proposed MLDD algorithm based on hybrid codes version has a very distinct performance to parity check based DSCC-MLDD, however, it requires a much lower area overhead, ranging from 30-40% (Table 2)

In general, the memory browse access delay of the planned MLDD (Viterbi, 1967; Hemati and Banihashemi, 2006; Naeimi and DeHon, 2009) is simply addicted to the Word Fault Rate (WER). If there are a lot of faults, then a lot of words have to be compelled to be absolutely decoded. All told the higher than strategies they created use of two dimensional parity equations it increase the total time for computation through further overhead because of the addition of redundant bit (parity). It explores the thought of two dimensional modulo sum that successively scales back the memory consumption (in terms of LUT's) of the device. Redundancy has been reduced through this algorithm and reduces the timing up to 10-20%.

The memory browse access delay of the planned MLDD rule (Mod-2 based checksum) is just keen about the WER. If there are a lot of errors, then a lot of words have to be compelled to be totally decoded. The detection section of the MLDD is code-length-independent and so has constant range of cycles for all. Timing outline of the

Table 3: Delay results of the device

	Existing method (ID)		Existing method (2D)		Proposed method (Mod 2 checksum)	
Cell: in>out	Gate delay	Net delay	1.218	1.108	1.218	0.995
IBUF	2.218	2.108	0.704	0.668	0.704	0.622
	1.704	1.668	0.704	0.455	0.704	0.455
LUT's	1.704	2.455	0.704	0.424	0.704	0.424
	1.704	0.424	0.704	0.42	0.704	0.420
	12.381ns		10.381ns		9.491ns	
Total	(7.306ns logic, 2.916 ns route, 71.5% logic, 28.5% route)		(7.306 ns logic, 3.075 ns route, 70.4% logic, 26.6% route)		(7.306 ns logic, 3.075 ns route, 70.4% logic, 26.6% route)	

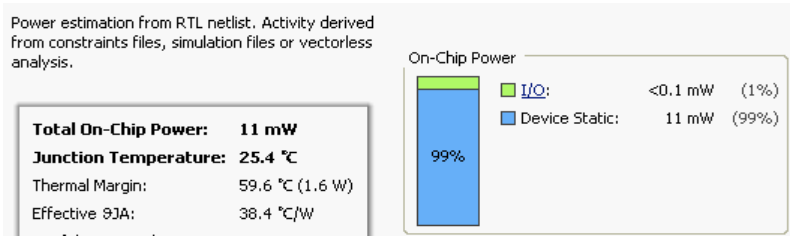

Fig. 9: Power results for two dimensional parity

performance of the three completely different styles is given in Table 3 and the proposed method outperforms timing up to 10-20% is shown in Table 3. The "detecting" column provides the particular range of timing and there might be some miscalculation within the codeword. Within the case that there are not any errors within the codeword, the styles would want, in total, the quantity of cycles given within the "no errors" column (which is that the addition of the "I/O" and "detecting" columns). On the opposite hand, the "errors" column provides the full range of cycles required by the look to correct the errors within the codeword.

The proposed algorithm on MLDD version contains a terribly similar performance to existing method, however it needs a far lower space overhead, ranging from 30-40%. In all the strategies they use two dimensional redundant check equations and increase the occupancy of gate to be declined through two dimensional modulo sum algorithmic flow so as to avoid wasting the area. The over-head introduced by proposed MLDD's are incredibly tiny, contrary to the existing case. A crucial final comment is that the area overhead of the MLDD really decreases with relevancy the MLDD method through two dimensional modulo algorithm (through Checksum rule). All the drawbacks have been overcome by the use of two dimensional checksum flow shown in the above results (Table 2 and 3).

Power analysis: Power estimation and analysis become even additionally vital as FPGAs increase in logic capability and performance by migrating to smaller method geometries. Total power in associate FPGA is that the total of two parts:

Static power: Static power results primarily from semiconductor unit outpouring the current within the device. Outpouring current is either from source-to-drain or through the gate chemical compound and exists once the semiconductor unit is logically "OFF".

Dynamic power: Dynamic power is related to style activity and change events within the core or I/O of the device. Dynamic power is set by node capacitance, offer voltage and change frequency. The accuracy of the Xilinx Power Tools depends on 2 primary components:

- Device information models and device characterization is integrated into the tools
- Inputs accurately entered by the user into the tools

Power Estimation exploitation XPE (XPower Estimator using PlanAhead 13.2) for correct estimates of your application, enter realistic info that is as complete as attainable. Modeling Diamond Statefinite an explicit facet of the de-check in a way that's too conservative or that lacks comfortable data of the planning may result in impractical estimates. The power consumption of the existing method has in turn enhanced the on-chip power due to high increase in static power, it has to be overcome by two dimensional modulo sum by reducing the switching the activity of the gate by optimizing procedure which has mentioned above in Algorithm 2. It ranges up to 20% than the existing algorithm (Fig. 9-11).

The proposed fault diagnosis algorithm significantly makes power, timing and space overhead stripped and to scale back the decoding time through Hybrid codes than the present technique and it provides promising possibility for memory applications.

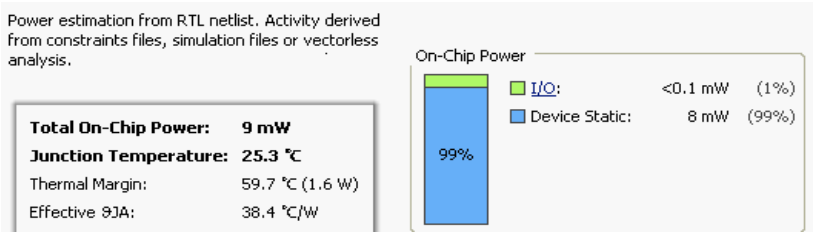

Fig. 10: Power results for two dimensional modulo sum

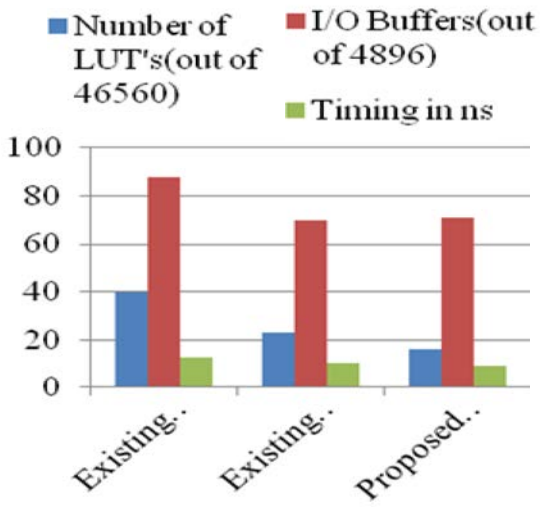

Fig. 11: Comparison on optimizing parameter (memory, timing and power)

CONCLUSION

In digital logic, an n-bit lookup table can be implemented with a multiplexer whose select lines are the inputs of the LUT and whose inputs are constants. An n-bit LUT can encode any n-input Boolean function by modeling such functions as truth tables. This is an efficient way of encoding Boolean logic functions and LUTs with 4-6 bits of input are in fact the key component of modern Field-Programmable Gate Arrays (FPGAs). Storage caches (including disk caches for files, or processor caches for either code or data) work also like a lookup table. The table is built with very fast memory instead of being stored on slower external memory and maintains two pieces of data for a sub range of bits.

In this study, an algorithmic scheme has been derived from the existing technique to effectively correct the faults caused by Multiple Cell Upsets (MCUs) as well as SEU in memories. The data placed in the memory for the identification of the faults in an MCU provides additional fault correction capabilities. Modified algorithmic methodology helps the correction of burst faults better than the existing method does and it also helps to reduce the LUTs level. Additionally, it helps to accelerate the decoding time and to effectively reduce the area occupied by the present MLDD, better than the previously proposed algorithms for DSC codes does. Thus the results show that the method is also effective in reducing LUTs, power and delay when MCUs are present. The proposed scheme has been validated by simulation (Plan Ahead 13.2) by using a large number of fault combinations and this scheme has also been analyzed by using Spartan 6 low power FPGA to evaluate its cost in terms of circuit area and speed. In future, ML decoding for fault correction followed by switching network in terms of segmentation can be proposed in order to reduce extra overheads. Here, partition via benes network helps to modules the complex data into simpler data segment which in turn reduce the complexity of the data. This can be an interesting problem for the future research. The application of the proposed technique to memories that use scrubbing is also an interesting topic which in fact was the original motivation that led to the MLDD scheme.

REFERENCES

Bose, R.C. and D.K.R. Chaudhuri, 1960. On a class of error correcting binary group codes. Inf. Control, 3: 68-79.

Hamming, R.W., 1950. Error detecting and error correcting codes. Bell. Syst. Tech. J., 29: 147-160.

Hemati, S. and A.H. Banihashemi, 2006. Dynamics and performance analysis of analog iterative decoding for Low-Density Parity-Check (LDPC) codes. Commun. IEEE. Trans., 54: 61-70.

Ibe, E., H. Taniguchi, Y. Yahagi, K.I. Shimbo and T. Toba, 2010. Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule. Electron Devices IEEE. Trans., 57: 1527-1538.

Liu, S., P. Reviriego and J.A. Maestro, 2012. Efficient maximum likelihood fault detection with difference-set codes f or memory applications. IEEE. Trans. Very Large Scale Integr. Syst., 20: 148-156.

Mao, Y. and A.H. Banihashemi, 2001. Decoding low-density parity-check codes with probabilistic schedule. Proceedings of the 2001 IEEE Pacific Rim Conference on Communications, Computers and signal Processing, August 26-28, 2001, IEEE, Victoria, British Columbia, Canada, ISBN: 0-7803-7080-5, pp: 119-123.

Massey, J.L., 1963. Threshold Decoding. MIT Press, MA., Pages: 113.

Naeimi, H. and A. DeHon, 2009. Fault secure encoder and decoder for nanomemory applications. Very Large Scale Integr. Syst. IEEE. Trans., 17: 473-486.

Reed, I.S., 1954. A class of multiple-error-correcting codes and the decoding scheme. Inf. Theory Trans. IRE. Prof. Group, 4: 38-49.

Torrens, G., B. Alorda, S. Barcelo, J.L. Rossello and S.A. Botaet al., 2010. Design hardening of nanometer SRAMs through transistor width modulation and multi-Vt combination. Circuits Syst. Express Briefs IEEE. Trans., 57: 280-284.

Viterbi, A., 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inform. Theor., 13: 260-269.

Weldon, E.J., 1966. Difference-set cyclic codes. Bell Syst. Tech. J., 45: 1045-1055.

Xiao, H. and A.H. Banihashemi, 2009. Comments on successive relaxation for decoding of LDPC codes. Commun. IEEE. Trans., 57: 2846-2848.

Xiao, H., S. Tolouei and A.H. Banihashemi, 2008. Successive relaxation for decoding of LDPC codes. Proceedings of the 24th Biennial Symposium on Communications 2008, June 24-26, 2008, IEEE, Kingston, Ontario, Canada, ISBN: 978-1-4244-1946-3, pp: 107-110.