Lavanya Thunuguntla, Bindu Madhavi K, Ramesh Kumar Reddy C / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622
www.ijera.comVol. 2, Issue 2,Mar-Apr 2012, pp.292-299

Secure Transmission for Nanomemory using EG-LDPC

Lavanya Thunuguntla * , Bindu Madhavi K **, Ramesh Kumar Reddy C ***

* (Associate Professor, Department of ECE, HITAM, Hyderabad, India)
** (Assistant Professor, Department of ECE, HITAM, Hyderabad, India)
*** (M.Tech (VLSI System design) II Year, Department of ECE, HITAM, Hyderabad, India,)

Abstract

Most of the Memory cells have been protected from soft errors for more than a decade; due to the increase in soft error rate in logic circuits, the encoder and decoder circuitry around the memory blocks have become susceptible to soft errors as well and must also be protected .This project propose a new approach to design fault-secure encoder and decoder circuitry for memory designs. The key novel contribution of this project is identifying and defining a new class of error-correcting codes whose redundancy makes the design of fault-secure detectors (FSD) particularly simple. In this project, a fault-tolerant nano-memory architecture is implemented which tolerates transient faults both in the storage unit and in the supporting logic (i.e., encoder, decoder (corrector), and detector circuitries).In this project, the Euclidean Geometry low density parity check (EG-LDPC) codes have the fault-secure detector capability will be proved. Using some of the smaller EG-LDPC codes, we can tolerate bit or nanowire defect rates of 10% and fault rates of 10^{-18} upsets/device/cycle**,** An unified approach is presented to tolerate permanent defects and transient faults. This unified approach reduces the area overhead.

Keywords: **Decoder, encoder, fault tolerant, memory**

I. Introduction and Motivation

Nanotechnology provides smaller,faster, and lower energy devices which allow more powerful and compact circuitry; however, these benefits come with a cost—the nanoscale devices may be less reliable. Thermal and shot-noise estimations [1], [2] alone suggest that the transient fault rate of an individual nanoscale device (e.g., transistor or nanowire) may be orders of magnitude
higher than today's devices. As a result, we can expect combinational logic to be susceptible to transient faults in addition to storage cells and communication channels. Therefore, the paradigm of protecting only memory cells and assuming the surrounding circuitries (i.e., encoder and decoder) will never introduce errorsis no longer valid.A mathematical definition of ECCs which have simple FSD which do not requiring the addition of further redundancies in order to achieve the fault-secure property. Identification and proof that an existing LDPC code (EG-LDPC) has the FSD prooperty. A detailed ECC encoder, decoder, and corrector design that can be built out of fault-prone circuits when protected by this fault-secure detector also implemented in fault-prone circuits and guarded with a simple OR gate built out of reliable circuitry .

To further show the practical availability of these codes, we work through the engineering design of a nanoscale memory system based on these encoders and decoders including the Memory banking strategies and scrubbing, Reliability analysis, Unified ECC scheme for both permanent memory bit defects and transient upsets.

II Fault-Tolerant Computing Using Hybrid Nano-CMOS Architecture

Architectures based on nanoscale molecular devices are attracting attention for replacing CMOS architectures at the end of the semiconductor roadmap. The two most promising nanotechnologies, according to ITRS, are silicon nanowires and carbon nanotubes. Although they offer unmatched densities for building logic, interconnect and memory, they suffer from very defect-prone manufacturing processes. This is further exacerbated by testing complexities where it is nearly impossible to detect all defects in a large nanoscale chip. Furthermore, the small structures in nanoscale architectures are susceptible to transient faults which can produce arbitrary soft errors. As a result, fault tolerance is necessary to make nanoscale architectures practical and realistic.

We propose an architecture that can tolerate a large number of undetected manufacturing faults as well as a large rate of transient faults. Our architecture is characterized by multiple levels of redundancy and majority voting to correct errors caused by such faults. A key aspect of the architecture is that it contains a judicious balance of both nanoscale and traditional CMOS components. A companion to the architecture is a compiler with heuristics tailored to quickly and compactly map logic onto partially defective components. Experimental results demonstrate the efficacy of the architecture. Low-Density Parity-Check Codes Based on Finite Geometries. Later,a geometric approach to the construction of low-density parity-check (LDPC) codes has were found.Four classes of LDPC codes

are constructed based on the lines and points of Euclidean and projective geometries over finite fields. Codes of these four classes have good minimum distances and their Tanner graphs have girth 6. Finite-geometry LDPC codes can be decoded in various ways, ranging from low to high decoding complexity and from reasonably good to very good performance. They perform very well with iterative decoding. Furthermore, they can be put in either cyclic or quasi-cyclic form. Consequently, their encoding can be achieved in linear time and implemented with simple feedback shift registers.

III Coding Scheme:

The technique introduced in this work exploits the existing structure of the ECC to guarantee the fault-secure property of the detector unit without adding redundant computations. We start with ECC definition for our fault-secure detector capable codes. its parity-check matrix and generator matrix are the cyclic shifts of their first rows. The checking or detecting operation is the following vector-matrix multiplication.

$$S = C \times H^T$$

Where, H is an (n−k) ×n Parity-Check matrix. (n − k)-bit vector S is called syndrome vector.

A syndrome vector is zero if C is a valid codeword and non-zero if C is an erroneous codeword.

3.1 Creating a parity check matrix:

The parity check matrix for a given code can be derived from its generator matrix and (vice-versa). If the generator matrix for an [*n*, k]-code is in standard form

$$G = [I_k \mid P]$$

Then the parity check matrix is given by

$$H = [-P^T \mid I_{n-k}]$$

Where 'I' is a (n-k) identity matrix.

'P' is a k x (n-k) matrix that generates parity bits.Because, $GH^T = P - P = 0$.

Like the general parity-prediction technique [4] concurrently generates (predicts) the parity-bits of the encoder

outputs (encoded bits) from the encoder inputs (information bits).For any valid codeword *x*, $H_x = 0$. For any invalid codeword, the syndrome vector *S* satisfies. The rows of a parity check matrix are parity checks on the code words of a code. That is, they show how linear combinations of certain digits of each codeword equal zero.

3.2 ECC with Fault Secure Detector:

In this proposed system the encoder is protected with parity-prediction and parity checker. The decoder is protected by adding a code checker (detector) block. If the code checker detects a non-codeword, then the error in the decoder is detected. Here we propose a multiple-error fault tolerant decoder and encoder that are general enough for any decoder and encoder implementation and for any kind of ECC that satisfies the restricted ECC definition. The restricted ECC definition which guarantees a fault-secure detector capable ECC is as follows:

Let C be an ECC with minimum distance d. C is FSD-ECC if it can detect any combination of overall (d – 1) or fewer errors in the received codeword and in the detector circuitry.

3.3 LDPC codes:

LDPC codes have several advantages, which have made them popular in many communication applications like low density of the encoding matrix,Easy iterative decoding, generating large code words that can approach Shannon's limit of coding.

An LDPC code is defined as the null space of a parity check matrix H that has the following properties like 1. Each row has ρ number of 1's and each column has γ number of 1's, the number of 1's that are common between any two columns (λ) is no greater than 1, i.e., λ = 0 or 1 and both ρ and γ are small compared to the length of the code and the number of rows in H.

As both ρ and γ are very small compared to the code length and the number of rows in the matrix H, H has a low density of 1's. Hence H is said to be a low density parity check matrix and the code defined by H is said to be a low-density parity check code. The density of H (r) is defined to be the ratio of the total number of 1's in H to the total number of entries in H in this case r = ρ/n = γ/J, where J is the number of rows in H. This kind of LDPC code is said to be a (γ, ρ)-regular LDPC code. If the weights of all the columns or rows in H are not the same, then it is called an irregular LDPC code.

3.4 Euclidean Geometry LDPC Codes:

Let EG be a Euclidean Geometry with n points and j lines. EG is a finite geometry that is shown to have the following fundamental structural properties:

1) every line consists of points;
2) any two points are connected by exactly one line;
3) every point is intersected by lines;
4) two lines intersect in exactly one point or they are parallel;

i.e., they do not intersect.

Let **H** be a **j x n** binary matrix, whose rows and columns corresponds to lines and points in an Euclidean geometry respectively, where $\mathbf{h_{ij}} = 1$ if and only if the i^{th} line of EG contains the $\mathbf{j^{th}}$ point of **EG**, and $\mathbf{h_{ij}} = 0$ otherwise.

A row in **H** displays the lines that intersect at a specific point in **EG** and has weight **ρ** . A column in **H** displays the lines that intersect at a specific point

in **EG** and has a specific weight γ. The rows of **H** are called the incidence vectors of the lines in **EG** and the columns of **H** are called the intersecting vectors of the points in **EG**. Therefore, **H** is the incidence matrix of the lines in **EG** over the points in **EG**. It is shown in **[3]** that **H** is a LDPC matrix, and therefore the code is an LDPC code.

Aspecial subclass of **EG-LDPC** codes, type-I 2-D EG-LDPC,is considered here. It is shown in **[3]** that type-I 2-D EG-LDPC have the following parameters for any positive integer t > 2:

- Information bits, $k=2^{2t} - 3^t$
- Length, $n =2^{2t} - 1$
- Minimum distance, $d \min =2^t +1$
- Dimensions of the parity-check matrix, n x n
- Row weight of the parity-check matrix, $p =2^t$
- Column weight of the parity-check matrix, $y =2^t$

Data bits stay in memory for a number of cycles and, during this period, each memory bit can be upset by a transient fault with certain probability. Therefore, transient errors accumulate in the memory words over time. In order to avoid accumulation of too many errors in any memory word that surpasses the code correction capability, the system must perform memory *scrubbing*.

Memory scrubbing is the process of periodically reading memory words from the memory, correcting any potential errors, and writing them back into the memory (e.g., [22]). This feature is shown in the revised system overview in Fig. 10. To perform the periodic scrubbing operation, the normal memory access operation is stopped and the memory performs the scrub operation.

The information bits are fed into the encoder to encode the information vector, and the fault secure detector of the encoder verifies the validity of the encoded vector. If the detector detects any error, the encoding operation must be redone to generate the correct codeword. The codeword is then stored in the memory. During memory access operation, the stored codeword's will be accessed from the memory unit. Codeword's are susceptible to transient faults while they are stored in the memory; therefore a corrector unit is designed to correct potential errors in the retrieved codeword's.

In our design all the memory words pass through the corrector and any potential error in the memory words will be corrected. Similar to the encoder unit, a fault-secure detector monitors the operation of the corrector unit. All the units shown in figure are implemented in fault-prone, nanoscale circuitry; the only component which must be implemented in reliable circuitry are two OR gates

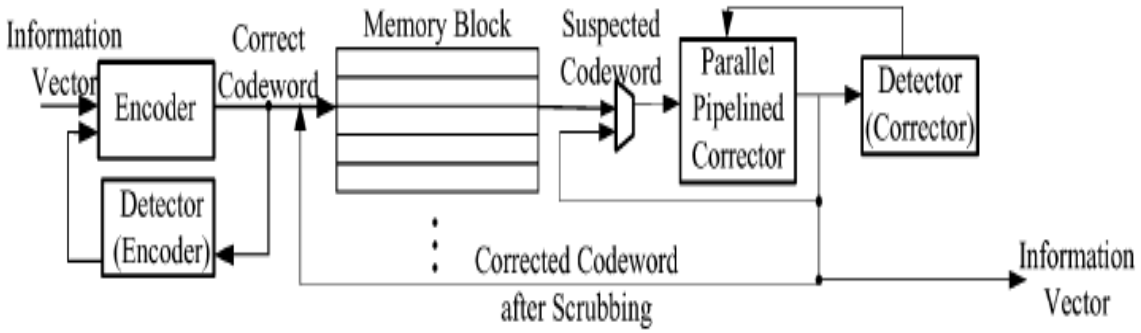

Fig 3.1 Fault tolerant memory architecture, with pipeline corrector

that accumulate the syndrome bits for the detectors. Data bits stay in memory for a number of cycles and, during this period, each memory bit can be upset by a transient fault with certain probability. Therefore, transient errors accumulate in the memory words over time.

3.5 Encoder

The encoder structure of a systematic code calculates the parity function of each parity bit based on the information bits. Each parity function is a xor gate similar to the detector. Therefore the encoder circuitry consists of (n − k) xor gates. An n -bit codeword c, which encodes a k-bit information vector i is generated by multiplying the k-bit information vector with a kxn bit generator matrix G; i.e., c=i.G. EG-LDPC codes are not systematic and the information bits must be decoded from the encoded vector, which is not desirable for our fault-tolerant approach due to the further complication and delay that it adds to the operation. However, these codes are cyclic codes. We convert the cyclic generator matrices to systematic generator matrices for all the EG-LDPC codes under consideration.

The code under consideration is a (15, 7, 5) EG-LDPC code. We used this code as an example to concretely illustrate the concept of the fault secure encoder, decoder, and checker; and the implementation of these units. We used the procedure

presented in [3] and [5] to convert the cyclic generator matrices to systematic generator matrices for all the EG-LDPC codes under consideration. The (15, 7, 5) EG-LDPC code has the generator polynomial

$$1+x^4+x^6+x^7+x^8 \quad \ldots\ldots \quad (1)$$

This generator polynomial will result in the generator matrix, shown in Figure 4.2(a) below. We perform linear row operations to make this cyclic non-systematic generator matrix into systematic form. We perform the following operations:

$$i_0=i_0+i_4+i_6 \quad \ldots. \quad (2)$$
$$i_1 = i_1 + i_5 \quad \ldots. \quad (3)$$
$$i_2=i_2+i_6 \quad \ldots. \quad (4)$$

This systematic form is presented in Figure 3.2. This is the correct representation of this systematic format.Based on this new generator matrix encoder shown in Figure 3.3

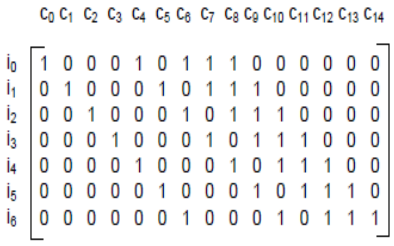

Fig 3.2 The generator matrix of (15, 7, 5) EG-LDPC code in cyclic format

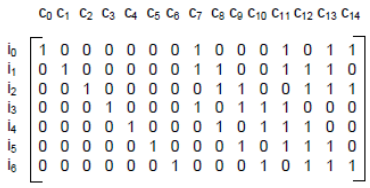

Fig 3.3 The generator matrix of (15, 7, 5) EG-LDPC codes in systematic format

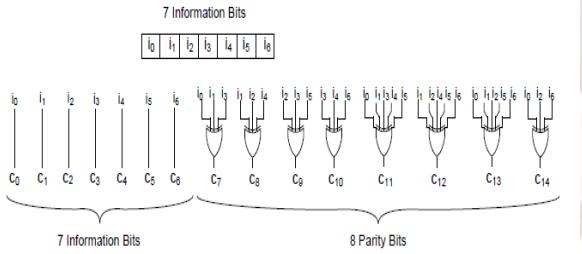

Fig 3. 4 Systematic encoder circuit of (15, 7, 5) EG-LDPC codes

3.6 Fault Secure Detector

The core of the detector operation is to generate the syndrome vector, which is basically implementing the following vector-matrix multiplication on the received encoded vector c and parity-check matrix H: s = c×H^T, and therefore each bit of the syndrome vector is the product of the following vector-vector multiply: si = c · hi^ T , where hi^T is the transposed of the ith row of the parity-check matrix. The above product is a linear binary sum over digits of c where the corresponding digit in hi is 1. This binary sum is implemented with a xor gate. Since the row weight of the parity-check matrix is p, to generate one digit of the syndrome vector we need a p-input xor gate, or (p− 1) 2-input xor gates in a tree structure. For the whole detector, it takes n(p − 1)2-input xor gates. An error is detected if any of the syndrome bits has a nonzero value.

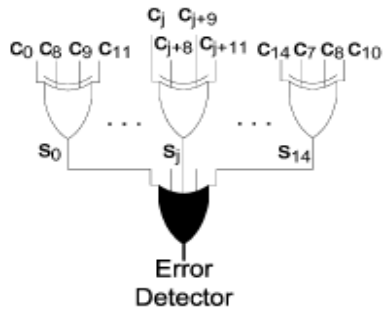

Fig 3.5: Fault-secure detector for (15, 7, 5) EG-LDPC code

The final error detection signal is implemented by an OR function of all the syndrome bits. The output of this n-input or gate is the error detector signal. In order to avoid a single point of failure, we must implement the OR gate in a reliable substrate i.e., we use a litho graphic scale wired-or. This n-input wired-or is much smaller than implementing the entire n (p − 1) 2-input XOR at the lithographic scale.

IV Memory

A method for operating a memory cell array, the method comprising assigning word lines of a memory cell array as addresses for writing sets of data thereto from a cache memory, and scrambling addresses of the sets of data by writing a first chunk of the particular set of data from the cache memory to a first word line of the array, and writing a second chunk of the particular set of data from the cache memory to a second word line of the array, the first chunk comprising a first subset of the particular set of data and the second chunk comprising a second subset of the particular set of data.

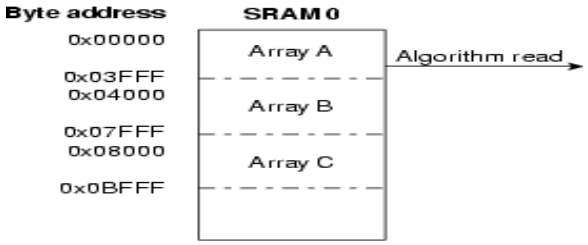

Fig 4.1: **Memory block diagram**

4.4.1 Nano-Memory Architecture Model:

Here Nano-Memory and *NanoPLA* architectures to implement the memory core and the supporting logic, respectively. Nano-Memory and Nano PLA are based on nanowire crossbars. The Nano-Memory number of cycles and, during this period, each memory bit can be upset by a transient fault with certain probability. Therefore, transient errors accumulate in the memory words over time. In order to avoid accumulation of too many errors in any memory word that surpasses the code correction capability, the system must perform memory scrubbing. Memory scrubbing is the process of periodically reading memory words from the memory, correcting any potential errors, and writing them back into the memory. Architecture developed in can achieve greater than 10^{11} b/cm^2 density even after including the lithographic-scale address wires and defects.

This design uses a nanowire crossbar to store memory bits and a limited number of lithographic scale wires for address and control lines. Fig. 3 shows a schematic overview of this memory structure. The fine crossbar shown in the centre of the picture stores one memory bit in each crossbar junction. To be able to write the value of each bit into a junction, the two nanowires crossing that junction must be uniquely selected and an adequate voltage must be applied to them. The nanowires can be uniquely selected through the two address decoders located on the two sides of the memory core.

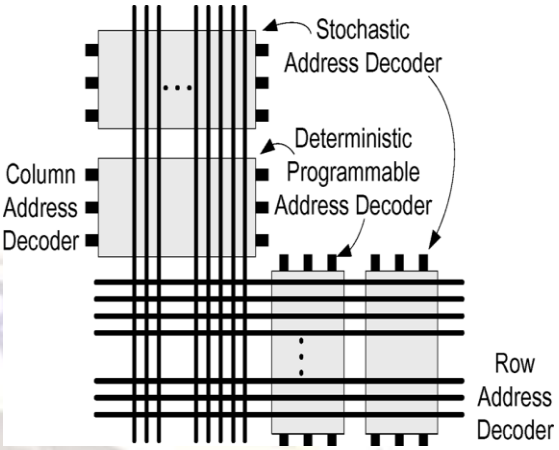

Fig 4.2 Structure of NanoMemory core

4.1 Corrector

One-step majority-logic correction is a fast and relatively compact error-correcting technique. There is a limited class of ECCs that are one-step-majority correctable which include type-I two-dimensional EG-LDPC.

4.2 One-Step Majority-Logic Corrector:

One-step majority logic correction is the procedure that identifies the correct value of a each bit in the codeword directly from the received codeword; this is in contrast to the general message-passing error correction strategy which may demand multiple iterations of error diagnosis and trial correction. Avoiding iteration makes the correction latency both small and deterministic. This technique can be implemented serially to provide a compact implementation or in parallel to minimize correction latency.

This method consists of two parts: 1) generating a specific set of linear sums of the received vector bits and 2) finding the majority value of the computed linear sums. The majority value indicates the correctness of the code-bit under consideration; if the majority value is 1, the bit is inverted, otherwise it is kept unchanged. The circuit implementing a serial one-step majority logic corrector for (15, 7, 5) EG-LDPC code is shown in Fig.4.3.

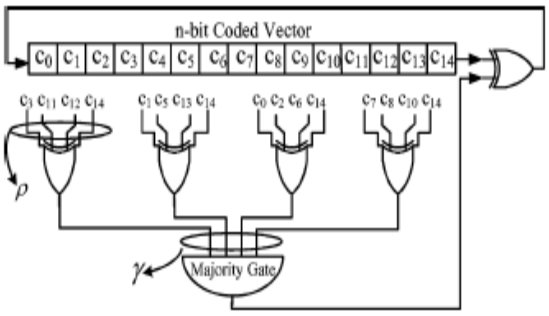

Fig 4.3 Serial one-step majority logic corrector structure to correct last bit (bit 14th) of 15-bit (15, 7, 5) EG-LDPC code.

V Circuit Implementation

Here majority circuit implementation gate use Sorting Networks the majority gate has application in many other error-correcting codes, and this compact implementation can improve many other applications. We use binary Sorting Networks to do the sort operation of the second step efficiently. An -input sorting network is the Structure that sorts a set of n bits, using 2-bit sorter building blocks. Fig.5.1 shows a 4-input sorting network. Each of the vertical lines represents one comparator which compares two bits and assigns the larger one to the top output and the smaller one to the bottom [see Fig.5.1 (b)]. The four-input sorting network, has five comparator blocks, where each block consists of two two-input gates; overall the four-input sorting network consists of ten two-input gates in total.

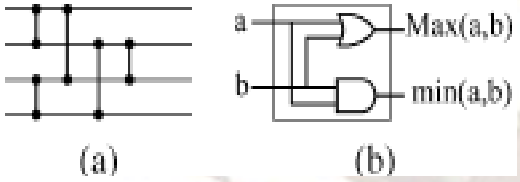

Fig 5.1 Four-input sorting network; (a) each vertical line shows a one-input comparator. (b) One comparator structure

5.1 Serial Corrector:

As mentioned earlier, the same one-step majority-logic corrector can be used to correct all the n bits of the received codeword of a cyclic code. To correct each code-bit, the received encoded vector is cyclic shifted and fed into to the XOR gates as shown in Fig 4.5.1. The serial majority corrector takes cycles to correct an erroneous codeword. If the fault rate is low, the corrector block is used infrequently; since the common case is error-free codewords, the latency of the corrector will not have a severe impact on the average memory read latency. The serial corrector must be placed off the normal memory read path.The memory words retrieved from the memory unit are checked by detector unit. If the detector detects an error, the memory word is sent to the corrector unit to be corrected, which has the latency of the detector plus the round latency of the corrector.

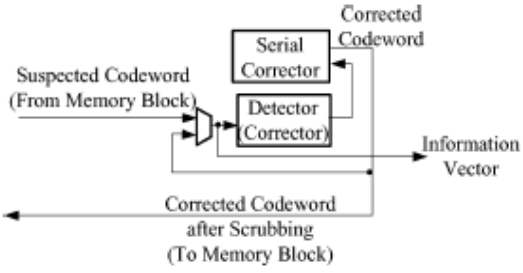

Fig 5.2 Partial system overview with serial corrector

5.2 Parallel Corrector:

For high error rates, the corrector is used more frequently and its latency can impact the system performance. Therefore we can implement a parallel one-step majority corrector which is essentially copies of the single one-step majority-logic corrector. All the memory words are pipelined through the parallel corrector. This way the corrected memory words are generated every cycle. The detector in the parallel case monitors the operation of the corrector, if the output of the corrector is erroneous; the detector signals the corrector to repeat the operation. Note that faults detected in a nominally corrected memory word arise solely from faults in the detector and corrector circuitry and not from faults in the memory word. Since detector and corrector circuitry are relatively small compared to the memory system, the failure rate of these units is relatively low.

However, when the detector observers an error, the memory word must pass through the corrector to be corrected. The total number of cycles that will be lost is equal to the latency of the corrector and the detector for our serial corrector design, the main latency is due to the corrector. For larger codes, where the serialized corrector can take a long time, multiple copies of the corrector can be used to reduce the throughput loss. Longer scrubbing intervals increase the number of errors accumulated in the memory and therefore more retrieved memory words have to go through the correction operation.

5.3 FPGA:

A Field-programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable".

The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping is the partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC design, offer advantages for many applications.

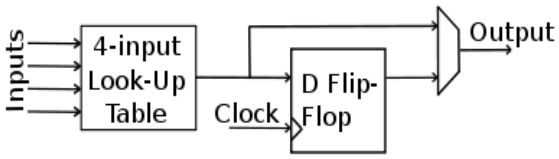

Fig 5.3 Common FPGA Architecture

The Spartan®-3E family of Field-Programmable Gate Arrays (FPGAs) is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The five-member family offers densities ranging from 100,000 to 1.6 million system gates.

These Spartan-3E FPGA enhancements, combined with advanced 90 nm process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

VI Theoretical Calculations

After successfully compiling an FPGA design using the Xilinx development software, the design can be downloaded using the iMPACT programming software and the USB cable.

The information bits are fed into the encoder to encode the information vector, and the fault secure detector of the encoder verifies the validity of the encoded vector.7-bit information vector is applied to encoder module as shown below:

Input message vector = $i_0\ i_1\ i_2\ i_3\ i_4\ i_5\ i_6$
= 000 0010

The encoded vector consists of information bits followed by parity bits.

Codeword = $[C_0\ C_1\ \ldots.\ C_{14}]$

C = [I : P];

I = Message Part;

P= Parity Part; P = $[P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7]$

$P_0 = i_0$ xor i_1 xor i_3 =0 + 0 + 0 =0;

$P_1 = i_1$ xor i_2 xor i_4 = 0 +0+0 = 0;

$P_2 = i_2$ xor i_3 xor i_5 = 0 +0+1 = 1;

$P_3 = i_3$ xor i_4 xor i_6 = 0 +0+0 = 0;

$P_4 = i_0$ xor i_1 xor i_3 xor i_4 xor i_5 = 0 +0+0+0+1 = 1;

$P_5 = i_1$ xor i_2 xor i_4 xor i_5 xor i_6 = 0+0 +0+1+0 = 1;

$P_6 = i_0$ xor i_1 xor i_2 xor i_5 xor i_6 = 0+0+0+1+0 = 1;

$P_7 = i_0$ xor i_2 xor i_6 = 0 +0+0 = 0;

Code word = (000 0010 0010 1110)

The checking or detecting operation is the following vector-matrix multiplication

$$S = C \times H^T$$

Syndrome vector is zero if c is a valid codeword and non-zero if c is an erroneous codeword.

$$S = \begin{bmatrix} 0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,1\,1\,1\,0 \end{bmatrix} \begin{bmatrix} 1\,0\,0\,0\,1\,0\,1\,1 \\ 1\,1\,0\,0\,1\,1\,1\,0 \\ 0\,1\,1\,0\,0\,1\,1\,1 \\ 1\,0\,1\,1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,1\,1\,0\,0 \\ 0\,0\,1\,0\,1\,1\,1\,0 \\ 0\,0\,0\,1\,0\,1\,1\,1 \\ 1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,0\,0\,0\,1 \end{bmatrix}$$

S = 0000 0000

Hence received codeword is valid code word.

Original Codeword = (000 0010 0010 1110)

- For 1st cycle, Error Codeword = (000 0011 0010 1111),
- 1st xor gate inputs: 0111,Output: 1
- 2nd xor gate inputs: 0111,Output:1
- 3rd xor gate inputs: 0011, Output:0
- 4th xor gate inputs: 0001,Output:1

Majority gate inputs are: 1101, for this condition majority gate output = 1

Majority gate value = 1 then invert the bit present in 14th position

Majority gate value = 0 then bit position is unchanged. After 15 th cycle, Codeword: (000 0010 0010 1110) i.e. Correct Codeword.

VII Conclusion

In this paper, fault-tolerant memory system is presented and it is capable of tolerating errors not only in the memory bits but also in the supporting logic including the ECC encoder and corrector using Euclidean Geometry low density parity check (EG-LDPC) codes. These codes have the fault-secure detector capability (FSD). Using these FSDs, a fault-tolerant encoder and corrector is designed, where the fault-secure detector monitors their operation. An unified approach is presented to tolerate permanent defects and transient faults. This unified approach reduces the area overhead. Without this technique to tolerate errors in the ECC logic, reliable encoders and decoders are required.

References

[1] M. Forshaw, R. Stadler, D. Crawley, and K. Nikolic´, "A short review of nanoelectronic

architectures," *Nanotechnology*, vol. 15, pp. S220–S223, 2004.

[2] J. Kim and L. Kish, "Error rate in current-controlled logic processors with shot noise," *Fluctuation Noise Lett.*, vol. 4, no. 1, pp. 83–86, 2004.

[3] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.

[4] S. J. Piestrak, A. Dandache, and F. Monteiro, "Designing fault-secure parallel encoders for systematic linear error correcting codes," *IEEE Trans. Reliab.*, vol. 52, no. 4, pp. 492–500, Jul. 2003.

[5] R. J. McEliece, *The Theory of Information and Coding*. Cambridge, U.K.: Cambridge University Press, 2002.

Authors

Ms Lavanya Thunuguntla has 6 years of Teaching experience and presently working as an Associate Professor in the Department of ECE in Hyderabad Institute of Technology and Management (HITAM), Hyderabad, AP(India).She received her B.Sc degree in Computer Science from Acharya Nagarjuna University in 2002, M.Sc degree in Physics from University of Hyderabad, Hyderabad in 2004 and M.Tech from Indian Institute of Technology (IIT) Kharagpur in 2007. She has Professional Membership in IEEE. She has various journal publications in International journals in the field of Nano Technology etc. She has guided several M.Tech and B.Tech projects. She has strong motivation towards research in the fields of Nano Technology, Microwave and Optical & Analog Communications ,VLSI system design etc.

Mrs Bindu Madhavi K has 7 years of Teaching experience and presently working as an Assistant Professor in the Department of ECE in Hyderabad Institute of Technology and Management (HITAM), Hyderabad, AP(India).She received her B.Tech degree in ECE from JNTU in 2003, M.Tech in VLSI System Design from JNTU ,Hyderabad. She has guided several M.Tech and B.Tech projects.Her areas of interests are VLSI System Design and Communication Systems.

Mr Ramesh Kumar Reddy C, doing his M.Tech II Year in VLSI System design in Hyderabad Institute of Technology and Management. His area of interest is VLSI & System Design