National Conference on Innovative Research & Industrial Applications

www.worldconferences.org

Generation of On-Chip Code with Crosstalk

T. AVANIJA[1], A. ANASUYAMMA[2]

[1]PG Scholar, Dept of ECE, Audisankara College of Engineering and Technology, JNTU Anantapur, AP, India.

[2]Assistant Professor, Dept of ECE, Audisankara College of Engineering and Technology, JNTU Anantapur, AP, India.

Abstract: Capacitive and inductive coupling between bus lines results in crosstalk induced delays. Many bus encoding technique have been proposed to improve the performance. Existing implementation techniques and mapping algorithms in the literature only apply the specific encoding. This paper presents the first generalized framework for a stall-free on-chip codeword generation strategy that is scalable and easy to automate. It is applicable to the coupling aware encoding techniques that allow recursive codeword generation. The proposed implementation strategy iteratively generates code-words without explicitly enumerating them. The code-words are calculated on-chip using basic function blocks, such as adders and multiplexers. Three encoding techniques were implemented using the proposed strategy. Experimental results show significant reduction in the area and power dissipation over the existing method that uses random logic to implement the codec.

Keywords: Codeword Generation, Crosstalk, Encoding.

I. INTRODUCTION

Every Bus line exhibits capacitive and inductive coupling with its neighboring lines. Opposing transitions on coupled bus lines causes slowed down transitions resulting in reduced performance. Over the years many techniques have been proposed to tackle this problem, such as repeater insertion, use of shield lines, and bus encoding. The worst case crosstalk induced delay is observed when tightly coupled bus lines undergo simultaneous opposing transitions. The simplest solution to this problem would be to insert shield lines in between each pair of bus lines. An alternative is to use crosstalk avoidance codes. Many crosstalk avoidance codes have been proposed in past that rely on adding redundancy to eliminate crosstalk induced delays. By delaying the lines that exhibit worst case transition, the technique reduces the Miller like effect observed in the mutual coupling capacitance. This type of transmission scheme is effective in reducing the energy consumption but less effective in reducing the crosstalk induced delay.Many of these encoding techniques Enumerate valid code-words and map them to the data words. Such mapping becomes impractical for wider buses. For wider buses, an on-chip memory based implementation is impractical as well because it suffers from the same scalability issues arising from enumerative mapping. Non enumerative encoding techniques are presented.

The method is referred to as real-time because the code words are never explicitly enumerated and stored in memory. The goal of the proposed method is to provide with a generalized strategy for scalable codeword generation for encoding techniques that allow recursive codeword generation. This paper provides details of the application of the proposed framework for effective real-time codeword generation to existing encoding techniques. Due to the recursive nature of codes discussed in this paper the Fibonacci sequence is a common occurrence in all three. Even though the proposed strategy utilizes Fibonacci numbers for convenience of explanation, it is not limited to encoding techniques based strictly on Fibonacci sequence. As an example, the correlation graph of the weight limited version of the code proposed and has been explained in Section III-A. This paper is organized as follows. Section II-A describes related work, provides rational. Section II-B provides some definitions and describes properties an encoding scheme must possess to be implementable using proposed strategy. The details of the steps involved in the proposed implementation strategy are provided in Section III-A. Section III-B provides details of scalable real-time implementation based on the correlation graph. Section IV presents experimental results. Three performance metrics namely, Area, execution time for design automation tools, and power, are used to evaluate the effectiveness of the proposed method. Section V concludes the paper.

II. PRELIMINARIES

Code words designed to eliminate crosstalk show a highly structured nature. Code words with common most significant bit patterns can be classified into various sets. The process is formally introduced in Algorithm1. Real-time on-chip implementation of encoder and decoder can be achieved by back-tracking such a procedure so that the code words can be formed and mapped in a fast and effective manner without explicitly enumerating them. Implementation of three such codes is described in this paper.

A. Related work

One widely recognized way to avoid crosstalk induced delays is to avoid all opposing transitions on every pair of lines in a bus. This problem was studied in detail. Two types of codes are discussed: 1) memory-based and 2) memory less. Memory-based codes rely on code-words from two consecutive time frames while memory less codes rely on the codeword

transmitted during single time frame. The following focus on published work on memory less codes, the latter is discussed in this paper. It is mathematically proved that the largest clique is formed by all of the neighbors of a codeword that is in the form of alternating 1's and 0's (...101010...) [20].The codebook is implemented as random logic. This code is referred to as opposing transitions elimination encoding (OTEE) in the following discussion. As a result, avoiding any codeword with consecutive 1's eliminates any simultaneous transitions on neighboring bus lines. Although code proposed in [19] relies on code-words from two consecutive time frames, it can be considered as a memory less code because a simple exclusive-or array can be used to translate the transitions to 1's and 0's. This way for an n bit bus, the decoder logic has only n inputs and not 2n as in memory based codes. This code is referred to as no adjacent transitions (NAT) in the following discussion.

It is possible to have comparatively more efficient code that allows some of the opposing transitions if they are compensated by an assisting transition. This, in turn, results in significant overhead.

This kind of encoding eliminates crosstalk induced slowdown while maintaining transition speedup. In the following discussion, this code is referred to as slowdown elimination encoding (SEE). It is a scalable method and can easily be implemented on-chip. All three of the encoding techniques eliminate the worst case crosstalk induced delay by avoiding simultaneous opposing transitions on neighboring lines. The worst case delay of a bus line is the same for all three techniques. In other words, the choice of the encoding technique does not change the crosstalk induced delay on the bus. [5] has slightly less redundancy as compared to [20] and [19. The key difference between the proposed and the existing work is that the introduction of the codes allows implementation of a wider variety of codes, such as codes with additional restrictions on the maximum weight of the code-words. In that sense, this paper is a framework that generalizes on-chip code generation and is not limited to a specific code as our previous work in [8] or the work in [4], [6], and [18]. The results show scalability of the proposed approach when applied to a variety of codes. For the code in [5], the results from [6] are also given. The goal of this paper is to propose a more generalized framework that can be applied to a number of existing codes. Implementation details of three such codes are presented in this paper.

B. Definitions and Terminology

Definition 1: A valid codeword is a bit vector that does not contain bit pattern that causes undesired effects, such as crosstalk induced delays anywhere within it. Such a bit pattern is referred to as prohibited bit pattern.

Definition 2: A k-bit segment of a codeword is a k-bit binary bit string that is a subset of the given codeword

Definition 3: Class is a set of all valid code-words that contain a fixed most significant bit pattern

Example 1: Consider a 5-bit codeword 10100. Under a particular encoding scheme, 10100 is a valid codeword only if two 4-bit segments (namely, 1010 and 0100), three 3-bit segments (namely, 101, 010, and 100), four 2-bit segments (Namely, 10, 01, 10, and 00) are valid code-words themselves. This is a necessary but not sufficient condition since a 6-bit codeword 110100, formed by appending a leading 1 to 10100 may not necessarily be a valid codeword. As a result, to build a set of all valid (k+1)-bit code-words one must work with a set of valid k-bit code-words. A set of (k+1)-bit code-words can be formed by appending a leading 0 or 1 to certain k-bit code-words. Newly formed set of (k+1) bit code-words can be classified according to the MSB of the code-words. This is a recursive procedure. The nodes of the graph on the left side list the contents of the class. As an example, (1, 3) represents a set of all valid 3-bit code words with MSB equal to 1, namely, {100, 101}. Consider a code that prohibits any consecutive 1's from appearing in the codeword. Any k-bit vector that contains consecutive 1's is not However, it is not absolutely necessary that the cardinality of individual sets follow Fibonacci series. As an example, the set sizes for the weight limited NAT code are not members of Fibonacci series as explained in Section III. Mapping is done in such a way that first codeword in the ordered set is mapped to smallest data (namely, decimal 0), second codeword is mapped to next smallest data (namely, decimal 1) so on and so forth.

III. PROPOSED IMPLEMENTATION STRATEGY

The sequence of steps followed in the proposed technique is depicted in Fig2. The Process begins with making an initial attempt to identify the correlation between code words of different sizes by classifying them into various classes according to the most significant bit. This section describes how the codec generation process can be automated.

A. Generation of Correlation Graph

Consider an initial classification of a set X(1) consisting of 1-bit code into two classes, namely, (0, 1) and (1, 1). In absence of any encoding technique, the set of 2-bit code-words X(2) is formed by appending a leading 0 and leading 1 to both of the aforementioned classes. The resultant section of graph has four nodes and four edges as shown in Fig. 3(a).During the first iteration of the graph generation flow depicted in Fig. 2, a small software module identifies and lists any invalid code-words that are present in the newly formed 2-bitcode set. The correlation between X(n) and X(n+1) is said to hold true if all elements of X(n+1) are valid (n+1) bit code-words. If the correlation between X(n) and X(n+1) holds true and if the redundancy is acceptable, HDL descriptions of the codec is generated based on the resultant correlation graph. The resultant correlation graph for the NAT code in [19] is depicted in Fig. 5.For some codes such as OTEE, the relationship between X(1) and X(2) is not the same as that between X(2) and X(3)as seen in Fig. 6. In such a scenario, the procedure described above is repeated till a recursive pattern is observed as seen in Fig. 7. The sections of the correlation graph must fit one on top of the other like identical building blocks. There are three scenarios in which it becomes necessary to change the initial classification of code-words during the runtime of the proposed algorithm. They are as follows. First scenario occurs when the correlation between X(n) and X(n+1) does not hold true for reasonably large n and X(n+1)contains invalid code-words.

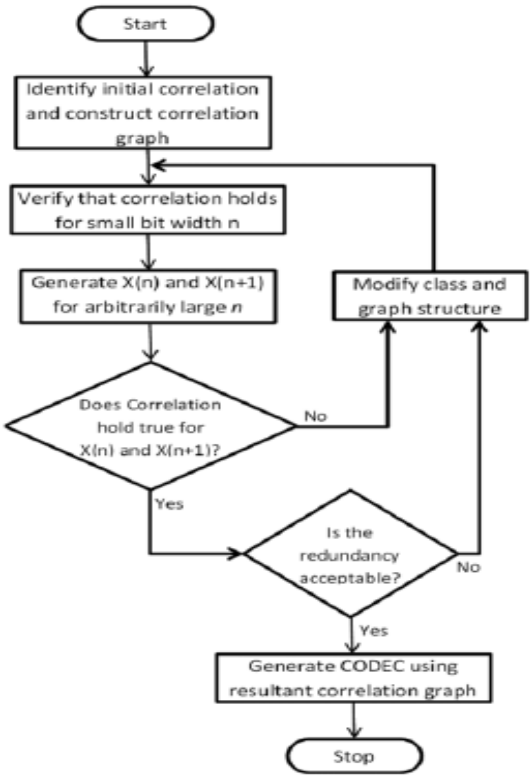


Fig.1 Steps for proposed implementation strategy

This implies that the correlation established in the a forementioned procedure is incomplete. The second scenario occurs when a recursive pattern in the correlation graph is not observed after testing a reasonably large number of levels in the graph. As discussed before, the correlation for NAT is established by checking three levels of the graph while the correlation is established for OTEE after checking five levels of the graph. For more complicated code the number could be higher. The two cliques [20] for the OTEE code are depicted in Fig. 8.The third scenario occurs when the redundancy of the resultant code in not acceptable. It is worth noting that the algorithm described in this section only removes edges in the graph that result in invalid code-words. Such removal may potentially remove a few valid code-words. The algorithm does not check for missing valid code-words.

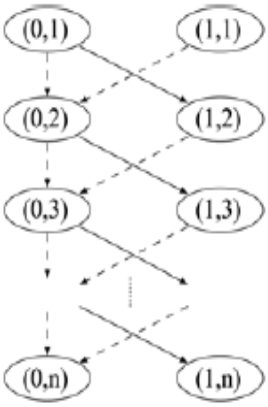


Fig2. (n,d,[n/2])- NAT correlation graph.

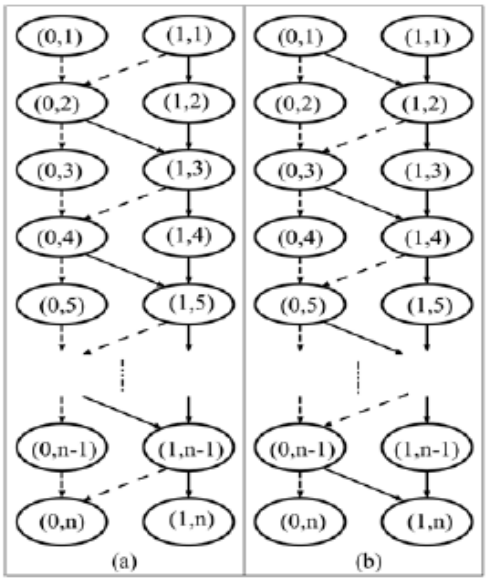


Fig. 3 OTEE Correlation graph (2 cliques).

This is acceptable as long as the redundancy of the resultant code is acceptable. As an example consider the correlation graph for the SEE code that prohibits patterns 101 and 010. If the initial classification based on the most significant bit is adopted the resultant graph is depicted in Fig. 9. The set X(n) will always have four code-words. Consequently, the redundancy is deemed unacceptable. The correlation graph is modified by splitting the classes based on two most significant bits and repeating the correlation graph formation procedure described above Using four classes, namely, (00, n), (01, n), (10, n), and (11,n), for SEE code, the resultant correlation graph is depicted in Fig. 10.

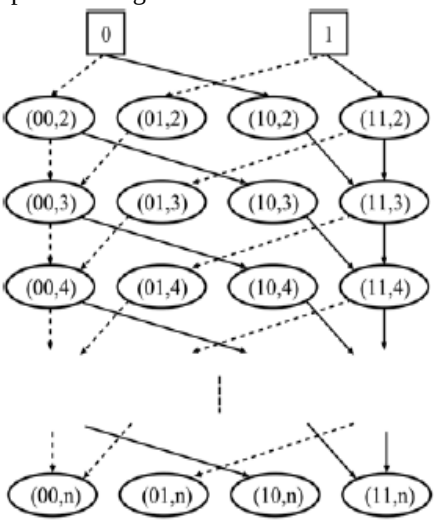


Fig.4. See correlation graph.

For example, class (0, 3, 1) will contain all valid3-bit code words with most significant bit 0 and weight equal to 1. It is worth noting that classes (1, n, 0) are always empty sets as any codeword with most significant bit 1 has minimum weight of 1. Following similar logic, the classes represented by shaded nodes in Fig. 11 are empty sets. As an example, appending 1 to

a 3-bit codeword001 with weight one will result in a new codeword 1001with weight two. Appending a leading 0 does not change the weight of a codeword. The software module that verifies the validity of the code words may flag a codeword as invalid either because it contains a forbidden bit pattern or because its weight exceeds the desired limit. Regardless of the reason, the edge in the graph that results in the invalid codeword is removed. It is worth noting that due to the additional constraint of maximum weight, the cardinality of sets of codewords does not follow the Fibonacci sequence. The weight limited NAT code is one of the examples of non-Fibonacci encoding techniques that can be implemented using the proposed implementation strategy.

B. Scalable Implementation

The code-words are calculated bit by bit by traversing the correlation graph from bottom to top. If n-bit code is used to transmit d-bit data, the ordered set X(n) formed by union of all classes on the nth level of the correlation graph are mapped to the 2d data words. Consider for example the OTEE correlation graph depicted in Fig8(a). The first data word, namely, decimal 0 is mapped to the first member of class (0,n), the second data word, namely, decimal 1 is mapped to the second member of the class (0, n) and so on. As a result, the decimal value of the data word determines the most significant bit of the corresponding codeword.This is achieved using if-else structure in the HDL that is implemented using multiplexers.|(0, n)|=Fb (n+1). If d-bit input data is Di then the nth code bit C(n) is generated as shown in Fig. 12.

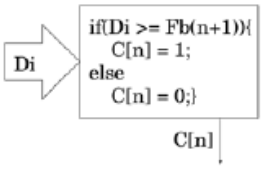


Fig.5. OTEE Code bit generation.

The comparison between input data Di and the comparison threshold t is achieved using multibit adders and the two's complement of t. If t inside the if-statement is a k-bit number, then the sum of t and the two's complement of t is 2k. If the two's complement of t is added to Di using a k-bit adder, the carry-out of the adder is set if Di is greater than or equal to the t. The data scaling discussed in the rest of this subsection ensures that the input data Di is less than 2k + 1.

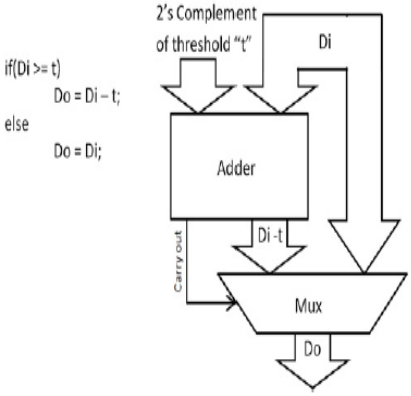


Fig.6. Encoder block Implimentation.

Consider as an example that the input data is to be compared to threshold of decimal value 5. The two's complement of 5 is decimal value 3. When Di is added to 3, the sum is greater than 7 ifDi is greater than or equal to 5. As a result the carry-out of the adder acts as a comparator output. Using advanced adders such as carry select adders or carry look-ahead adders, comparison can be achieved significantly fast. The structure of the HDL modules is depicted in Fig13. Since 3≥Fb (3), C3=1 and Do=3-Fb (3)=1. The third block performs E1(2, 1). Since 1_Fb(3), C2=0 and Do=1. Finally, the fourth block performs E2(1, 1). Since 1≥Fb (1), C1=1 andDo=1-Fb(1)=0. Irrespective of the input data, Do from the last block is always 0. The path traversed on the correlation graph that corresponds to input data 110 is shown in bold.

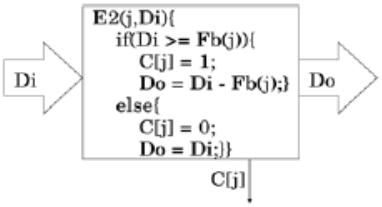


Fig.7. OTEE Encoder block E2.

A closer observation of functions E1 & E2 reveals that the encoder essentially breaks down the input data into sum of Fb(k). According to the functionality of the encoder, if Fb(k) is subtracted from the input data, the kth bit is set. The decoder must recover the original data by adding Fb(k) for every kth code bit that is set. The decoder simply adds up the Fb(k) corresponding to every bit that is set. Binary equivalent of the original data DR is given by

$$\sum_{i=1}^{n}(C(i) * Fb(i))$$

where C(i) is the ith code bit and Fb (i) is the ith Fibonacci number.

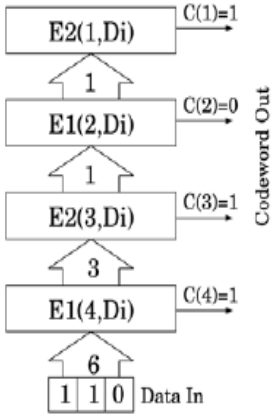


Fig. 8. OTEE Encoder.

The block diagram of the decoder using D(k,Di) function blocks is given in Fig. 17. Function D(k,Di) is given in Fig. 18.Following similar logic, real-time implementation of the NAT and SEE codes is achieved as discussed in the reminder of this section. The first |(0, n)| data words are mapped to all of the elements in the class (0, n).

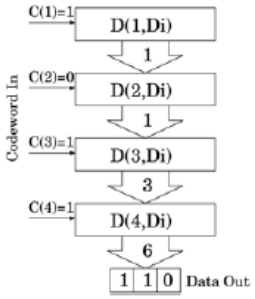


Fig.9. OTEE Decoder.

Table 1. Mapping 4 bit OTEE code to 3 bit data

Data	Data Word	Σ (k^{th}bit*Fb(k))	Code Word
0	000	0*3+0*2+0*1+0*1	0000
1	001	0*3+0*2+0*1+1*1	0001
2	010	0*3+1*2+0*1+0*1	0100
3	011	0*3+1*2+0*1+1*1	0101
4	100	0*3+1*2+1*1+1*1	0111
5	101	1*3+1*2+0*1+0*1	1100
6	110	1*3+1*2+0*1+1*1	1101
7	111	1*3+1*2+1*1+1*1	1111

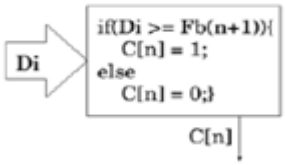


Fig.10. OTEE Decoder block

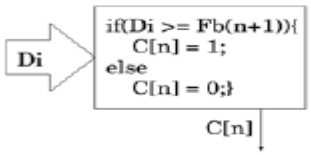


Fig.11. NAT Code bit Generation.

NAT code mapping shown in Table II (|(1, n)|+|(0, n)|)=2d ; hence, there are no unmapped code-words. Encoder essentially traverses the correlation graph in Fig. 5from bottom to top. It is observed that |X(k)|=Fb(k+2),|(0, k)|=Fb(k+1), and |(1, k)|=Fb(k). Starting with the last iteration, comparison of decimal value of the input data (Di)with |(0, k)| determines whether the codeword belongs to class(1, n) or (0, n). The resultant functionality is depicted in Fig. 19. Since |X(n)| - |X(n − 1)|=Fb(n), while migrating from nth level to n-1st, Fb(n) must be subtracted from the data input to the nth level. The parent–child relationship between every pair of levels is identical. The resultant function block E(k,Di) is shown in Fig. 20.The (4,3,2)-NAT encoder is shown in Fig. 21

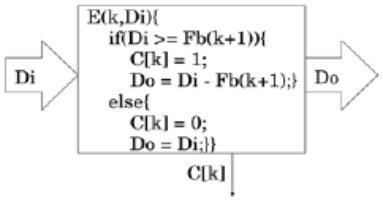


Fig.12. NAT Encoder block.

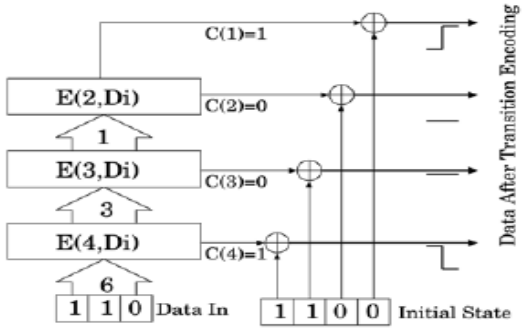


Fig.13. NAT Encoder Implementation.

The NAT encoder breaks down the input data into sum of Fb(k+1) values. If Fb(k+1) is present in the sum, the kth bit is set. Decoder simply adds the Fb(k+1) values corresponding to every code bit that is set. The original data Dr from an n-bit codeword can be given by:

$$\sum_{i=1}^{n}(C(i) * Fb(i+1))$$

where C(i) is the ith code bit and Fb(i+1) is the i+1st Fibonacci number.

The 3-bit data broken-up as a sum of kth bit*Fb(k) and the corresponding mapping is shown in Table II. As shown in Fig. 23 array of exclusive or gates translate the transitions on the bus lines to code-words. According to Table II the data '2' is mapped to code 0010, data '4' is mapped to0101 while data '7' is mapped to 1010. It is worth noting that the actual bits transmitted over the bus depend upon the initial state. As an example, consider data '2', '4' and'7' transmitted over the bus with initial state 0000 during consecutive time frames t0, t1 and t2. The resultant bus states are enumerated in Table III. Absence of consecutive 1's in the code-words ensures absence of adjacent transitions thereby avoiding crosstalk induced delay. Code bits are individually fed to each of the functional blocks. As long as proper code bits are fed to proper blocks, the order in which the addition takes place is irrelevant. Functionality of block D(k, Di) is depicted in Fig.

Table 2. Mapping 4 bit NAT code to 3 bit data

Data	Data Word	$\Sigma(k^{th}bit * Fb(k+1))$	Code Word
0	000	0*5+0*3+0*2+0*1	0000
1	001	0*5+0*3+0*2+1*1	0001
2	010	0*5+0*3+1*2+0*1	0010
3	011	0*5+1*3+0*2+0*1	0100
4	100	0*5+1*3+0*2+1*1	0101
5	101	1*5+0*3+0*2+0*1	1000
6	110	1*5+0*3+0*2+1*1	1001
7	111	1*5+0*3+1*2+0*1	1010

The length of the codeword is determined by the number of function blocks. The number of function blocks (E and D)is proportional to the number of code bits. An n-bit encoder has n (or n-1 function blocks, depending upon the type of the code). As an example, the 4-bit encoder/decoder circuits in Figs. 16, 17, and 28 have four blocks. A 5-bit encoder/decoder will have five blocks. Each function block consists of multi bit adders and multiplexers.

Table 3. NAT Transmission with initial state 0000

Time Frame	Data to be Transmitted	Mapped transition codeword	Initial State	Next State
t_0	010	0010	0000	0010
t_1	100	0101	0010	0111
t_2	111	1010	0111	1101

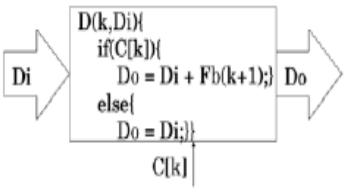


Fig.14. NAT Encoder block.

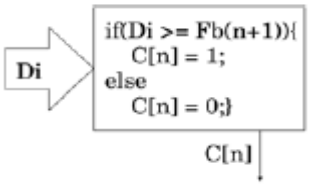


Fig.15. SEE Code bit generation.

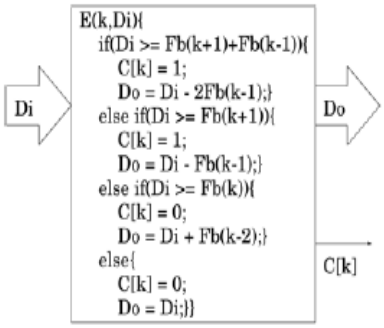


Fig.16. SEE Encoder block.

The proposed encoder and decoder are shown in Figs. 28and 29, respectively. The functionality of the decoder block is more complicated. Since there are four classes at each level, each decoder block requires two code bits, namely, k and k-1,to recreate the data. In the encoder block, if the code word belonged to class (11,k), then 2Fb(k-1) is subtracted from the input data; hence, to recover the original data, 2Fb(k-1) must be added. Applying similar logic to classes (00, k), (01, k),and (10, k) the functionality of decoder block can be derived as shown in Fig. 27. The functionality of the encoder and decoder blocks of the weight limited NAT code is more complex due to the complicated nature of the correlation graph.

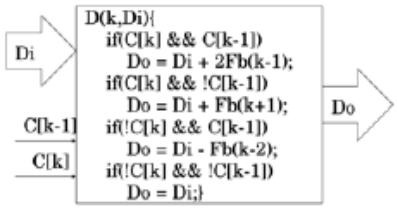


Fig.17. SEE Decoder block.

IV. EXPERIMENTAL RESULTS

In order to verify the effectiveness of the proposed implementation method, VHDL codes were written for encoder and decoder as random logic, as well as proposed method. As a result, for wider buses, the proposed method requires much less hardware. The experimental results for up to 72-bit bus are provided to emphasize the fact that the proposed method is in fact capable of completing the task of encoder/decoder synthesis as opposed to the random logic implementation, which runs out of memory beyond 16-bit bus. The execution time of the software tool to synthesize a 16-bit conventional encoder is comparable to the time for synthesis of a 72-bitencoder using proposed implementation strategy.

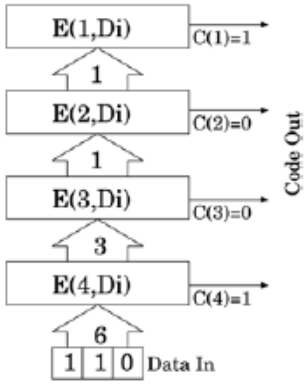


Fig.18. SEE Encoder.

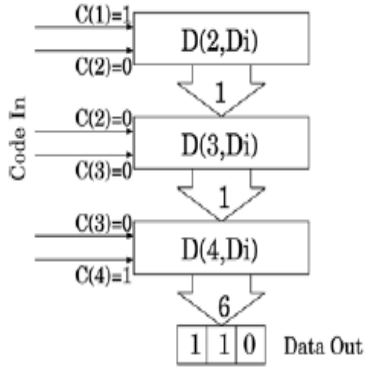
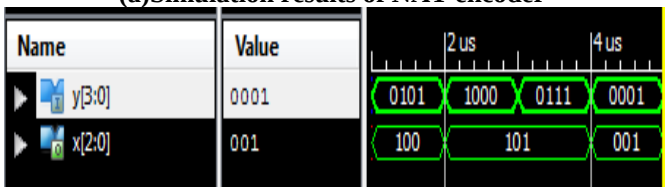


Fig.19. SEE Decoder.

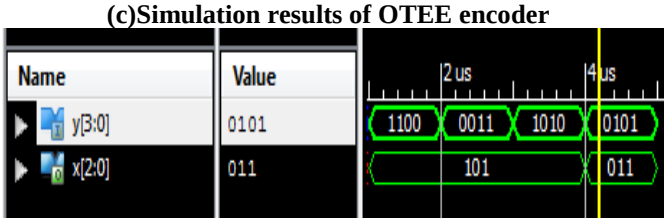
The proposed implementation strategy improves the performance while consuming between 11% to 38% more power as compared to a Non-encoded bus. The total power consumption of the encoded bus is the sum of power dissipated on the bus and that on the logic. If the total power of encoded bus is restricted to that of the non encoded bus, the power consumption on the bus must be reduced to accommodate the additional power consumed by the logic circuit. The primary goal of crosstalk avoidance encoding is performance enhancement in terms of speed. This can be achieved by either treading off power or routing area. It is worth noting that increasing routing area also improves the delay on the bus. The goal of using bus encoding is to improve the performance using redundant bits. Based on xilinx simulations performed the delay of the encoded bus is approximately 37%better compared to an un-encoded bus. It is worth noting that despite having different redundancies, the worst case delay for all three codes is the same as a result the performance improvement for SEE, OTEE, and NAT is the same. The comparisons are only meaningful for the code proposed in [5].

(a)Simulation results of NAT encoder

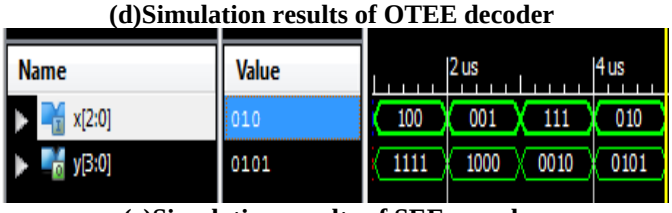


(b)Simulation results of NAT decoder

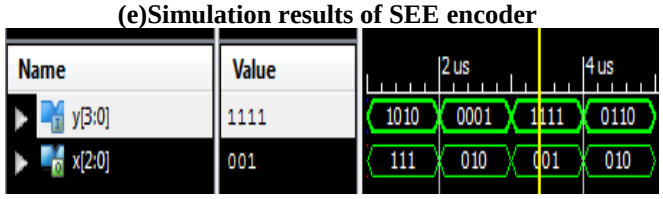
(c)Simulation results of OTEE encoder



(d)Simulation results of OTEE decoder



(e)Simulation results of SEE encoder



(f)Simulation results of SEE decoder

Fig.20.Simulation results.

The experimental results provided in [6] are interpreted from the chart published in the paper without having first hand data. Unfortunately, the results provided extend only up to 32-bit buses.

Table 4. Hardware Overhead of OTEE

Data bits (d)	Gate Count (proposed)	Gate Count (Conventional)
8	107	295
10	160	1007
12	206	4006
14	285	13949
16	374	34084
20	590	-
24	859	-
32	1476	-
48	3334	-
56	4627	-
64	5984	-
72	7716	-

Table 5. Hardware Overhead of NAT Encoder

Data bits (d)	Gate Count (proposed)	Gate Count (Conventional)
8	157	253
10	270	960
12	348	3648
14	496	12593
16	662	29375
20	1080	-
24	1595	-
32	2828	-
48	6424	-
56	8892	-
64	11531	-
72	14778	-

Table 6. Hardware Overhead of SEE

Data bits (d)	Gate Count (proposed)	Gate Count (Conventional)
8	190	362
10	301	1285
12	438	5112
14	627	16959
16	822	44473
20	1295	-
24	1790	-
32	3189	-
48	7163	-
56	9641	-
64	12772	-
72	16029	-

It is seen in Fig20 that with increase in bus width, the hardware overhead of [6] appears to increase more rapidly than the proposed method. Therefore, for buses larger than 32-bits, the cost is expected to be higher.

V. CONCLUSION

The conventional implementation strategy is based on explicate numeration of codewords. Proposed implementation strategy takes advantage of the iterative structure of the encoding technique. The encoder/decode rare described as structural HDL model that uses multibit adders, comparators, and multiplexers as building blocks. This makes the strategy scalable for very wide buses. The proposed strategy has been applied to a variety of en coding techniques. The properties an encoding technique must possess to be implementable using the proposed strategy are described in this paper. Three of the existing encoding techniques that fit the criteria were implemented using proposed strategy with encouraging outcomes. All three encoding techniques exhibit similar scalable trends in areas such as hardware overhead, power consumption, memory requirements and time complexity.

VI. REFERENCES

[1] M. Anders, N. Rai, R. K. Krishnamurthy, and S. Borkar, “A transition-encoded dynamic bus technique for high-performance interconnects,” IEEE J. Solid-State Circuits, vol. 38, no. 5, pp. 709–714, May 2003.

[2] R. Ayoub and A. Orailoglu, “A unified transformational approach for reductions in fault vulnerability, power, and crosstalk noise & delay on processor buses,” in Proc. ASP-DAC, vol. 2. Jan. 2005, pp. 729–734.

[3] K.-C. Cheng and J.-Y.Jou, "Crosstalk-avoidance coding for low-power on-chip bus," in Proc. 15th IEEE Int. Conf. Electron. Circuits Syst., Aug.–Sep. 2008, pp.1051–1054.
[4] C. Duan, V. H. C. Calle, and S. P. Khatri, "Efficient on-chip crosstalk avoidance CODEC design," IEEE Trans. Very Large Scale Integr. Syst., vol. 17, no. 4, pp.551–560, Apr. 2009.
[5] C. Duan, A. Tirumala, and S. P. Khatri, "Analysis and avoidance of cross-talk inon-chip buses," in Proc. Hot Interconnects, vol. 9. Aug. 2001, pp. 133–138.
[6] C. Duan, C. Zhu, and S. P. Khatri, "Forbidden transition free crosstalk avoidance CODEC design," in Proc. ACM/IEEE Design Autom. Conf., Jun. 2008, pp.986–991.
[7] M. Ghoneima and Y. Ismail, "Delayed line bus scheme: A low-power bus scheme for coupled on-chip buses," in Proc. ISLPED, Aug. 2004, pp. 66–69.
[8] K. Karmarkar and S. Tragoudas, "Scalable codeword generation for coupled buses," in Proc. Design Autom. Test Eur., Mar. 2010, pp. 729–734.
[9] R.-B. Lin, "Inter-wire coupling reduction analysis of bus-invert coding," IEEETrans. Circuits Syst. I Reg. Papers, vol. 55, no. 7, pp. 1911–1920, Aug. 2008.
[10] C.-G. Lyuh and T. Kim, "Low power bus encoding with crosstalk delay elimination,"in Proc. 15th Annu. IEEE Int. ASIC/SOC Conf., Sep. 2002, pp. 389–393.
[11] M. Mutyam, "Fibonacci codes for crosstalk avoidance," IEEE Trans. Very Large Scale Integr. Syst., vol. 20, no. 10, pp. 1899–1903, Oct. 2012.
[12] M. Mutyam, "Preventing crosstalk delay using fibonacci representation," in Proc.17th Int. Conf. VLSI Design, 2004, pp. 685–688.
[13] M. Mutyam, "Selective shielding: A crosstalk-free bus encoding technique," in Proc. IEEE/ACM Int. Conf. Comput. Aided Design, Nov. 2007, pp. 618–621.
[14] D. Pamunuwa and H. Tenhunen, "Repeater insertion to minimise delay in coupled interconnects," in Proc. Int. Conf. VLSI Design, Jan. 2001, pp. 513–517.
[15] D. Rossi, C. Metra, A. K. Nieuwland, and A. Katoch, "New ECC for crosstalk impact miimization," IEEE Design Test Comput., vol. 22, no. 4, pp. 340–348,Jul.–Aug. 2005.
[16] D. Rossi, A. K. Nieuwland, A. Katoch, and C. Metra, "Exploiting ECC redundancy to minimize crosstalk impact," IEEE Design Test Comput., vol. 22, no. 1, pp.59–70, Jan. 2005.
[17] S. Salerno, E. Macii, and M. Poncino, "Crosstalk energy reduction by temporal shielding," in Proc. ISCAS, vol. 2. 2004, pp. 49–52.

Author's Profile:

T.Avanija is currently PG scholar of ECE, In VLSI Designs, Audisankara College of Engineering and Technology, Affiliated to JNTU Ananthapur, AP, India.

Mrs.Anasuyamma, M.Tech., Presently She is working as Assistant Professor in Dept. of ECE, Audisankara College of Engineering and Technology ,Affiliated to JNTU Ananthapur, AP, India.