International Journal of Scientific Engineering and Technology Research

ISSN 2319-8885
Vol.03,Issue.43
December-2014,
Pages:8739-8745

www.ijsetr.com

An Efficient Avoidance of On-Chip Codeword Generation to Cope with Crosstalk

NAGISETTY SUBRAHMANYAM[1], G. NAGESWARAO[2], CH.VENKATRAMAIAH[3]

[1]PG Scholar, Dept of ECE, Audisankara Institute of Technology, Gudur, AP, India, Email: subbu.manyam60@gmail.com.
[2]Assoc Prof, Dept of ECE, Audisankara Institute of Technology, Gudur, AP, India, Email: bhavani.g24@gmail.com.
[3]Asst Prof, Dept of ECE, R.M.D. Engineering College, Kavaraipettai, Tamil Nadu, India, Email:venkat.chadalavada@gmail.com.

Abstract: Capacitive and inductive coupling between bus lines results in crosstalk induced delays. Many bus encoding techniques 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. Codeword mapping relies on graph-based representation that is unique to the given encoding technique. 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 overhead 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 [2]–[4], [6], [11]. 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. Encoding techniques such as those presented in duplicate each data line such that there is always at least one transition in the same direction as that on the line under consideration. Both of these solutions essentially double the routing area of the bus, which may not always be acceptable. Another 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. Some of the techniques that are most relevant to this paper have been discussed in the subsequent section.

Lin [10] studies, in detail, the bus invert coding and its effectiveness in reducing coupling and energy consumption on a bus line. It has been observed that the bus invert coding is most effective when the bus is divided into smaller groups, which implies increased redundancy. Such method reduces average number of couplings and/or switching activity thereby reducing the energy consumption. However, such method would not be effective against crosstalk induced delay as the capture mechanism at the receiver end still has to wait for the worst case delay transmission to settle. The introduction of static delay between adjacent bus lines has been discussed in [8]. 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. The bottom line of all of these encoding techniques is that they eliminate simultaneous opposing transitions on coupled lines by means of adding redundancy. Encoding techniques, such as those presented in [6] and achieve this goal by prohibiting certain bit patterns from appearing in the transmitted codeword. 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, such as those presented in [5], [7], and rely on mathematical modeling that applies only to the code proposed in them. The proposed paper presents a generalized framework to generate code words in a non-enumerative and scalable manner. Hardware overhead is measured in terms of the number of redundant bits. However, it is observed that as the number of lines in a bus increases, the size of combinational logic in the encoder/decoder increases exponentially. This also gives rise to optimization issues. A real-time on-chip encoder implementation method is needed to make the encoding scheme scalable for wide buses. The term real-time is used to describe non-enumerative codeword generation. The method is referred to as real-time because the code words are never explicitly enumerated and stored in memory. The goal of the

Copyright @ 2014 IJSETR. All rights reserved.

proposed method is to provide with a generalized strategy for scalable codeword generation for encoding techniques that allow recursive codeword generation. Most of the coupling aware encoding techniques in the literature fall in this category. In deep sub micrometer, the performance of interconnects is critical. As on-chip interconnects are placed closer and closer together, the coupling induced delays become more and more important. As the use of tightly coupled and wider buses become more common, a scalable codec implementations strategy becomes increasingly important.

This paper provides details of the application of the proposed framework for effective real-time codeword generation to existing encoding techniques, such as [6]. 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 has been explained. This paper is organized as follows. Section II describes Codeword Correlation. On-Chip Implementation of (N, D)-Nat Encoder in Section III Section IV presents experimental results. Three performance metrics, namely, hardware overhead, execution time for design automation tools, and power, are used to evaluate the effectiveness of the proposed method. Section V concludes the paper.

II. CODEWORD CORRELATION

A. Overview

Code words designed to eliminate crosstalk show a highly structured behavior. Code words with common properties can be classified into various sets. The cardinality of each set can be mathematically determined.

Definition 1: A valid codeword is a bit vector that does not contain a prohibited bit pattern.

Definition 2: Class is a set of all valid code words that contain a predetermined most significant bit pattern. Classes are labeled as *(s, n)* where *s* is the most significant bit pattern and *n* is the number of code bits. For example, Class *(00, 5)* is a set of all 5-bit valid code words that have *00* as most significant bit pattern.

Theorem 1: Any (n+1)-bit codeword is valid only if the n-bit code segments contained within it are valid n-bit code words themselves.

Proof: Consider a valid n-bit codeword, $x= (b_n,.., b_2, b_1)$. Since x is valid, by definition, does not contain the prohibited bit pattern. Consequently two (n-1)-bit vectors contained within x, namely $x_1=(b_n,..,b_2)$ and $x_2=(b_{n-1},..,b_2,b_1)$ do not contain the prohibited bit pattern. Hence, by definition, both x_1 and x_2 are valid code words. Similarly, if $y= (b_n,..,b_2 ,b_1)$ is not a valid n-bit codeword, by definition, it must contain the prohibited bit pattern. As a result (n+1)-bit vectors, namely $y_1=(b_{n+1},b_n,..,b_2,b_1)$ and $y_2=(b_n,..,b_2,b_1,b_0)$, must contain the prohibited bit pattern as well. Hence, by definition, y_1 and y_2 are not valid code words.

In other words, if x is not a valid codeword for a given code, then neither *0x* nor *1x* is a valid codeword. This is a necessary but not sufficient condition. As a result, in order 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 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 iterative procedure. Starting with set of 1-bit code words, a set of n-bit code words is formed in n iterations as shown in Fig. 1. Each level of the graph in Fig. 1 corresponds to an iteration of the while loop. Class A is a parent of Class B if some elements in Class B are formed by adding a leading 1 or 0 to every element in Class A. Also Class B is a child of Class A. As an example, in Fig. 1 Class (0, 2) is a parent Class to Classes (0, 3) and (1, 3) while a child Class of (0, 1) and (1, 1). Appending a 0 or 1 is depicted by dotted and solid arrows respectively. Arrow points from parent to child class. Every codeword within a Class contributes towards forming one or more code words in the next level. Being a structured graph the parent-child relationships are well defined. Union of all Classes in the bottom most levels of the graph form an ordered set of n-bit code words.

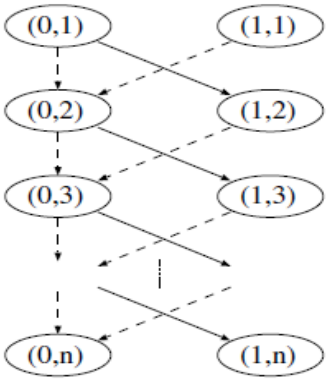

Fig.1. Codeword Correlation Graph.

Cardinality of such set $|X(n)| \geq 2^d$ determines the number of data bits *(d)* that can be transmitted using n-bit code words.

Definition 3: In an ordered set of code words, offset of a codeword is defined as the number of code words occurring before it within that set.

Definition 4: In an ordered set of code words, offset of a Class is defined as the offset of the first codeword of that Class.

The first codeword in the ordered set is mapped to smallest data (namely decimal *0*), the second codeword is mapped to next smallest data (namely decimal *1*) etc. Hence the decimal value of the data determines the offset of corresponding

codeword and consequently the Class of codeword in the final level. As the classification is done based on the most significant bits, determination of Class also determines the most significant bit (n^{th} bit) of the codeword. Parent Class of the codeword is determined using cardinality of Classes and the parent child relationships. Determination of parent Class determines the n-1*st* bit of the codeword. Determination of Class of a codeword and the parent Classes is based solely on the cardinality of the Classes involved and not the contents of these Classes. Since the cardinality of these Classes can easily be calculated mathematically, it is not required to explicitly enumerate all the code words. In particular, a topological traversal of the multilevel codeword correlation graph from bottom to top determines the codeword corresponding to a particular data word. The non-enumerative nature of the proposed method makes it scalable for wide buses and suitable for automation.

B. Codeword correlation in the (n, d, t)-NAT code

The code proposed it has been designed as a transition code where a *1* indicates a transition (either rising or falling) on a bus line while *0* means a stable value (stable 0 or a stable 1). The (n, d, t)-NAT code is an n-bit code that transmits d-bit data and has at most *t* 1's in each codeword. Thus more number of 1's corresponds to increased dynamic power consumption. Authors propose to limit power consumption by limiting the maximum number of 1's (the weight) to value *t*. For simplicity of explanation, the code without limitations on maximum weight is considered. The maximum weight of the code can be limited by dividing Classes into subclasses based

TABLE I: (N, D)-NAT CODEWORD CORRELATION (FIRST 4 ITERATIONS)

Code Bits *(n)*	*(0,n)*	*(1,n)*	*|(0,n)|*	*|(1,n)|*	*|X(n)|*	Data Bits *(d)*	Redundancy *(r)*
1	0	1	1	1	2	1	0
2	00, 01	10	2	1	3	1	1
3	000, 001, 010	100, 101	3	2	5	2	1
4	0000, 0001, 0010, 0100, 0101	1000, 1001, 1010	5	3	8	3	1

on the weights of the code words. Due to the nature of the code to avoid any consecutive 1's, a codeword with maximum weight consists of alternate 1's and 0's. Thus the maximum possible weight is $\lceil n/2 \rceil$. In this paper a simpler notation (n, d)-NAT is used instead of (n, d, $\lceil n/2 \rceil$)-NAT. Any further increase in the weight will cause more than one consecutive 1's thereby forming an invalid codeword. In order to avoid transition on neighboring lines, every pattern with more than one 1's on neighboring lines is discarded.

To construct a multilevel code correlation graph, consider a set of valid 2-bit code words X (2) = {00, 01, 10}. In order to form valid 3-bit code words, every element in X (2) must be augmented with either a leading *0* or a *1*. Adding a *0* as an MSB does not result in invalid code words. While adding a *1* however one must check the MSB of the element in X (2). For example appending *1* to *10* will form an invalid pattern 110. To simplify the procedure further set X (2) is divided into two Classes, namely (0, 2) and (1, 2*)*. As a result, X (2) = (0, 2) U (1, 2) and (0, 2) ∩ (1, 2) =φ. It is clear that a leading *0* can be added to all the elements of set X (2) while a leading *1* can only be added to all elements of Class (0,2). This is a recursive procedure. Results of first four iterations are presented in Table I. Thus Class *(*0, 3) is a child of (0, 2) and (1, 2) both, whereas (1, 3) is a child of (0, 2) only. The resulting codeword correlation graph is shown in Fig. 2.

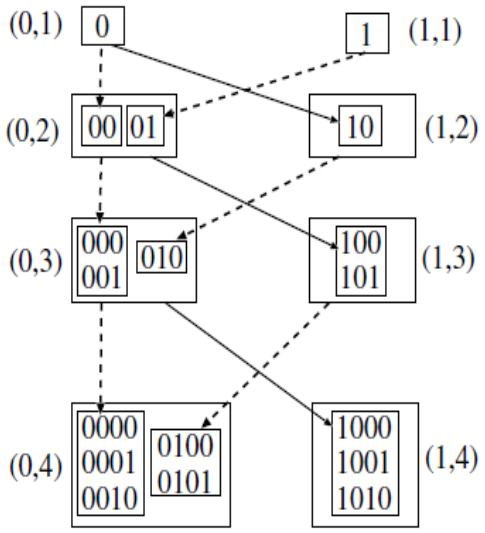

Fig.2. (n, d)-NAT Codeword Correlation Graph.

III. ON-CHIP IMPLEMENTATION OF (N, D)-NAT ENCODER

First | (0, *n*)| data words are mapped to all of the elements in the Class (*0*, *n*). Remaining data words are mapped to first (2*d*-| (*0*, *n*)|) elements of (*1*, *n*). Remaining elements of (*1*, *n*) are unmapped. For (4, 3)-NAT code mapping shown in Table II (| (*1*, *n*)|+| (*0*, *n)*|) =2*d* hence there are no unmapped code words. Encoder essentially traverses the codeword correlation graph depicted in Fig. 2 from bottom to top. It is observed that $|X(k)|=Fb(k+2)$, $|(0,k)|=Fb(k+1)$ and $|(1,k)|=Fb(k)$. Here Fb (k) is the k^{th} Fibonacci number. Starting with the last iteration, comparison of decimal value of the input data (D_i) with *|(0,n)|* determines whether the code word belongs to Class *(1,n)* or *(0,n)*. Hence the n*th* code bit C (n) is *1* if $D_i \geq Fb(n+1)$, *0* otherwise. Since parent and child Classes may have different offsets, input data scaling may be required while migrating from a child Class to a parent class. As seen in Fig. 3, if the codeword belongs to Class *(0, k)*, scaling of input data is not required since both child Class and the first parent Class, namely *(0, k-1)* have same offset (i.e. zero). Class *(1,k)* however, has an offset of Fb(k+1) while that of its parent, namely *(0,k-1)*, is zero. As a result in order to direct the encoder to proper parent Class, the input data D*i* must be scaled by subtracting Fb(k+1) from it. In other words, if code word belongs to *(1, k)* then $D_o=D_i$-Fb (k+1). The functionality of the k^{th} encoder block with input data D*i* labeled E (k, D*i*) is given in Fig. 4. Block E (1, D_i) can be eliminated since C (1) is always equal to the input D*i* to that block and D*o* is always zero.

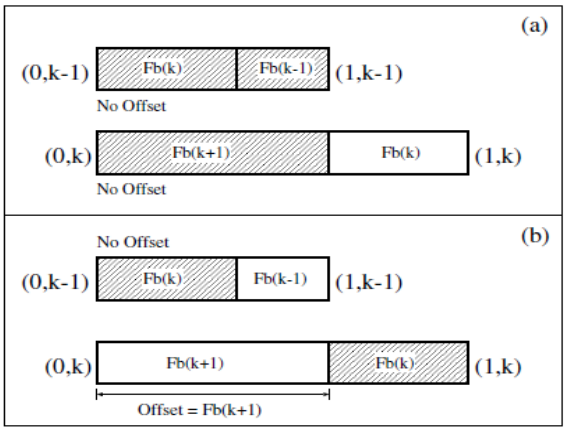

Fig.3. Scaling in NAT Encoder Block.

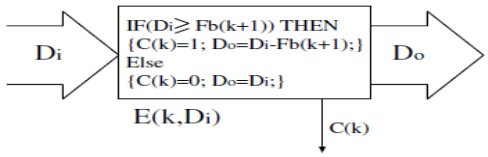

Fig.4. Functionality of kth Encoder Block.

The (4, 3)-NAT encoder is shown in Fig. 5. In the example in Fig. 5, input data is 6 *(110)*. Since 6 is grate than Fb (5), output D_o of E (4, 6) is 6-5=1 and C (4) is 1. Next comparison is done with Fb (4) =3. Since 1 is not greater than or equal to Fb (4), *Do* is 1 and C (3) is 0. This determines the Class of the codeword in the second last level. At the end of the process, code word 1001 is formed. As a result, bus lines at extreme ends undergo transition while middle lines are stable. For every kth code bit that is set (1≤k≤n), Fb (k+1) is subtracted from the input data and for every kth code bit that is reset, *0* is subtracted from the input data. In other words, the 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. If C (i) is the ith code bit and D_i is the input data to the i*th* decoder block then recovered data D_r from an n-bit codeword is given by:

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

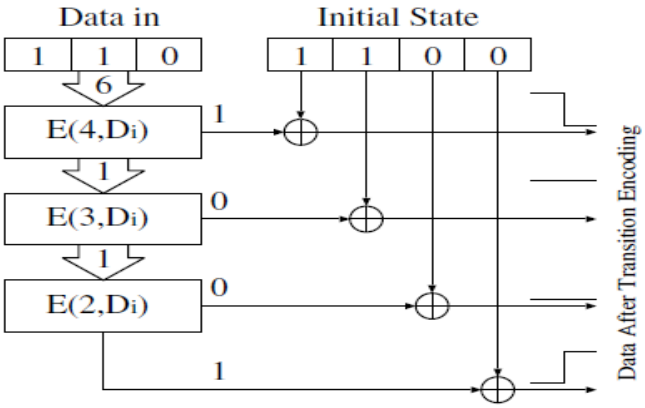

Fig.5. Overall NAT Encoder Implementation.

As an example, 3-bit data broken-up as a sum of products of the ith bit and Fb (i+1) as well as corresponding mapping is shown in Table II. As shown in Fig. 7 array of exclusive or gates translate the transitions on the bus lines to code words. 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, D_i) is shown in Fig. 6.

TABLE II: Mapping 4-Bit Nat Code To 3-Bit Data

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

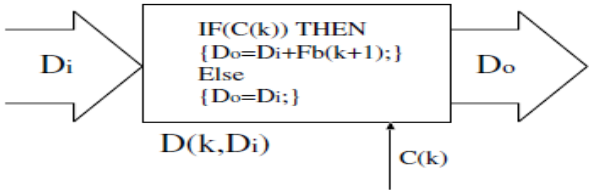

Fig.6. Functionality of kth Decoder Block.

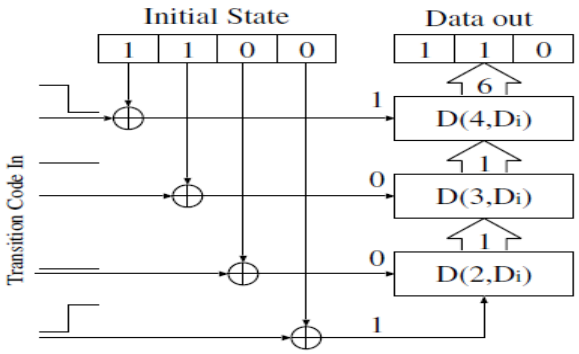

Fig.7. NAT Decoder Implementation.

Length of the codeword is determined by the number of function blocks. For example, to encode 4-bit data, code length of 6-bits is required. Therefore 5 function blocks are needed (E (1, D_i) is unnecessary). For very large codes, throughput of the encoder and decoder can be improved by using pipelined architecture.

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. These VHDL codes were then synthesized using Cadence encounter tool using 180nm technology and circuit statistics were analyzed. The HDL codes were written and synthesized in a generic 180nm technology using cadence RTL compiler and Cadence encounter to generate the

schematics and layout for the codec. Power consumption of the bus lines is based on post extraction figures (9 to 11) provided by the ASU predictive model for 180nm node. Each function block terminates with a register where the calculated value is stored for the next clock cycle. The synthesis tool is constrained to achieve the delay of each pipeline stage of the codec to be no more than 1nS. Since the codec produces one codeword each cycle, the average energy consumption is reduced to a reasonably small value for sufficiently large amount of data transfer.

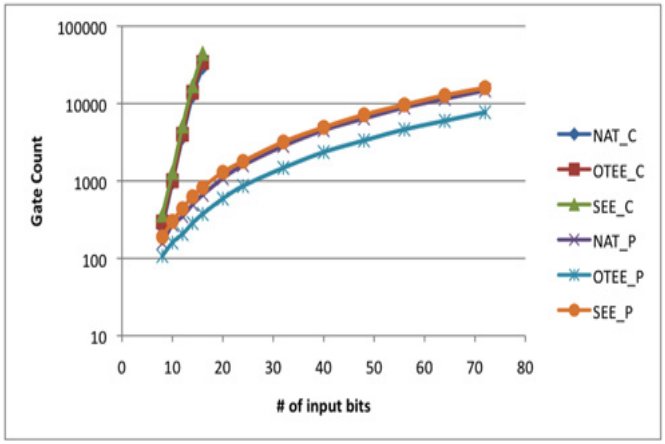

Fig.8. Hardware requirement.

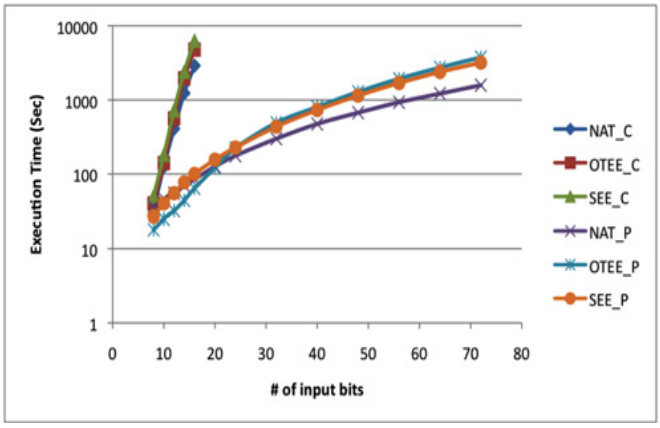

Fig.9. Execution time.

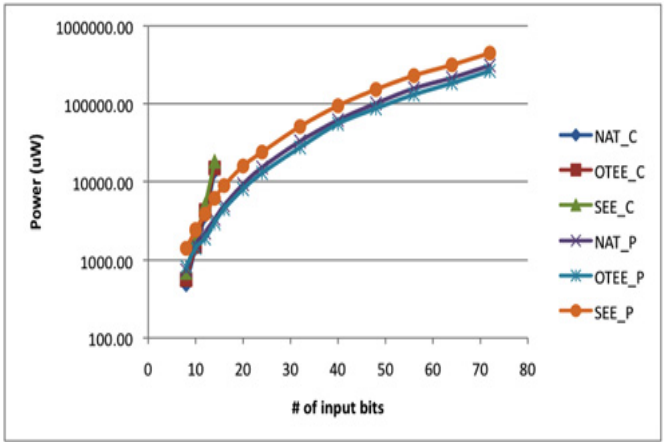

Fig.10. Power consumption.

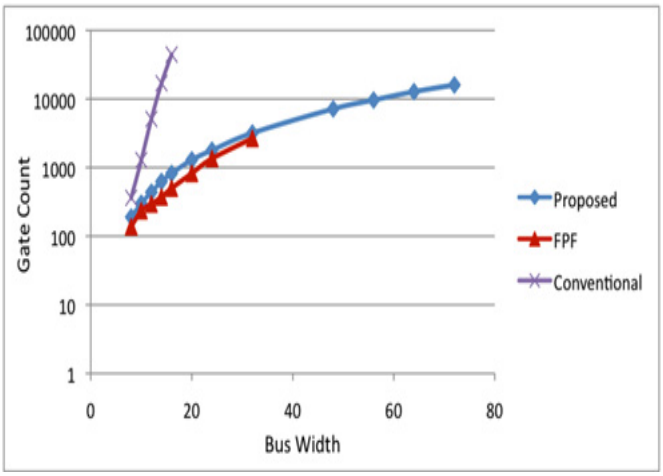

Fig.11. Hardware requirement of code in [6].

For the encoding techniques that do not provide the scalable implementation, it is assumed that the codec is implemented as a truth table based random logic circuit. Such circuit has d-bit data input and an n-bit code output that is transmitted over the bus. Give the data word as an input, the circuit produces the corresponding codeword as an output. Table VI describes the hardware overhead of implementation of OTEE for both the methods. The number of functional blocks required is linearly proportional of number of code bits and the functionality of each block is fairly simple. As a result, for wider buses, the proposed method requires much less hardware. A similar trend is observed in synthesis of (n, d, $\lceil n/2 \rceil$)-NAT code as shown in Table VII and that of SEE as seen in Table VIII. The synthesis tool failed to synthesize the conventional encoder/ decoder implementation for buses beyond 16-bits due to memory requirement violation. 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.

Fig.8 compares the three encoding techniques with respect to gate count. The curves labeled NAT C, OTEE C and SEE C represent the gate counts of conventional random logic implementation of the three encoding techniques while the curves labeled NAT P, OTEE P and SEE P represent the gate counts of implementations using the proposed strategy. The same naming convention is used in Figs. 9 and 10. The conventional method also exhibits a considerable computational overhead and consequently time required for code generation as well as synthesis of the encoder/decoder circuitry is considerably high as observed in Fig. 9. The execution time of the software tool to synthesize a 16-bit conventional encoder is comparable to the time for synthesis of a 72-bit encoder using proposed implementation strategy. A similar trend is observed in case of power consumption of the encoder/decoder implementation as seen in Fig. 10. The power consumption is estimated using the Cadence RTL compiler.

The power estimation for 16-bit and beyond conventional encoders is not shown as the RTL compiler terminated due to memory requirement violation. Table III depicts the power consumption on a 2500 micrometer bus capable of transmitting 14-bit data. As such, the bus without encoding has 14 lines while the bus with encoding has 20 lines. The bus was modeled using π-model based on using the same technology as the encoder/decoder. Despite having more number of lines, the worst case dynamic power consumption on the bus without encoding is higher since the coupling capacitance exhibits at Miller-like effect. Since bus encoding eliminates simultaneous opposing transitions, the Miller-like effect is not observed in encoded bus. As a result despite having more number of lines, the total power consumption on a Non encoded bus is slightly less. As observed in Table III. The total power consumption of the proposed technique is considerably less as compared to the conventional one. The main goal of crosstalk avoidance encoding is to improve the performance of the bus. The proposed implementation strategy improves the performance while consuming between 11% to 38% more power as compared to a non encoded bus.

TABLE III: Power Consumption With and Without Encoding

Encoding Technique	Non-encoded bus Worst case power (uW)	Encoded bus Worst case power (uW)	codec Power Conventional (uW)	codec Power Proposed (uW)	Total Power Conventional (uW)	Total Power Proposed (uW)
OTEE	11534.2	9872.6	14840.78	2962.16	24713.38	12834.76
NAT	11534.2	9872.6	13463.57	3305.12	23336.17	13177.72
SEE	11534.2	9872.6	19037.97	6088.52	28910.57	15961.12

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. This can be achieved by reducing the coupling capacitance between neighboring lines. This can be done by separating the bus wires over a larger area while keeping the wire width constant. The inverse relationship between the line separation and coupling capacitance and consequently power, results in drastic increase in area when power budget is tight. Table VII represents the overhead of encoding in terms of redundancy as well as actual routing area. The results show that to bring the power to the level of non encoded bus, the area increases by about 61% to 67% for some codes.

TABLE IV: Performance Improvement using Encoding

Encoding Technique	Performance Improvement		
	1500um	2000um	2500um
OTEE	37.1%	37.07%	37%
NAT	37.1%	37.07%	37%
SEE	37.1%	37.07%	37%

TABLE V: Overhead With Respect To Non Encoded Bus for No Extra Power

Encoding Technique	Redundancy	Area Overhead
OTEE	42.86%	61.7%
NAT	42.86%	67.8%
SEE	42.86%	194%

TABLE VI: 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	-

In addition, we observe that the codec is necessary. Due to relatively higher power consumption for the codec for SEE, the routing overhead of the technique is considerably higher compared to NAT and OTEE. 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 SPICE simulations performed using the π-model on buses from 1500um to 2500um, the delay of the encoded bus is approximately 37% better compared to an un encoded bus as seen in Table IV. 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.

TABLE VII: Hardware Overhead of (N, D, ⌈N/2⌉)-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	-

The rest of the section focuses on comparing the proposed method to the one presented in [7]. The comparisons are only meaningful for the code proposed in [6]. The experimental results provided in [7] are interpreted from the chart published in the paper without having first hand data. Fig. 11 plots the results of the proposed method and those interpreted from [7] represented by the curve labeled forbidden pattern free (FPF). The vertical axis represents the gate count while the horizontal axis represents the bus width. Unfortunately, the results provided extend only up to 32-bit buses. Although the gate count is a good indication of hardware overhead, the actual transistor count depends upon the technology and libraries used for implementation. It can be seen that the hardware overhead of proposed strategy is of the same order

as compared to that of the technique proposed in [7]. It is seen in Fig.11 that with increase in bus width, the hardware overhead of [7] appears to increase more rapidly than the proposed method. Therefore, for buses larger than 32-bits, the cost is expected to be higher.

TABLE VIII: 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	-

V. CONCLUSION

The conventional implementation strategy is based on explicit enumeration of code words. In case of very wide buses, such codeword enumeration results in exponential hardware overhead. Maintaining a codebook is not a practical approach. Proposed implementation strategy takes advantage of the iterative structure of the encoding technique. The encoder/decoder is described as structural HDL models that use multi-bit 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 encoding 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] Kedar Karmarkar, Student Member, IEEE, and Spyros Tragoudas, Member, IEEE, "On-Chip Codeword Generation to Cope With Crosstalk", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 33, No. 2, February 2014.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] C. Duan, A. Tirumala, and S. P. Khatri, "Analysis and avoidance of cross-talk in on-chip buses," in Proc. Hot Interconnects, vol. 9. Aug. 2001, pp. 133–138.

[7] C. Duan, C. Zhu, and S. P. Khatri, "Forbidden transition frees crosstalk avoidance CODEC design," in Proc. ACM/IEEE Design Autom. Conf., Jun. 2008, pp.986–991.

[8] 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.

[9] K. Karmarkar and S. Tragoudas, "Scalable codeword generation for coupled buses," in Proc. Design Autom. Test Eur., Mar. 2010, pp. 729–734.

[10] R.-B. Lin, "Inter-wire coupling reduction analysis of bus-invert coding," IEEE Trans. Circuits Syst. I Reg. Papers, vol. 55, no. 7, pp. 1911–1920, Aug. 2008.

[11] 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.

[12] M. Mutyam, "Fibonacci codes for crosstalk avoidance," IEEE Trans. Very Large Scale Integr. Syst., vol. 20, no. 10, pp. 1899–1903, Oct. 2012.

Author's Profile:

Nagisetty Subrahmanyam, Pursuing his M.Tech in VLSI Stream from Audisankara Institute of Technology, Gudur, AP, India.
Email: subbu.manyam60@gmail.com.

G. Nageswarao Working as Associate Professor Dept of ECE in Audisankara Institute of Technology Gudur, AP, India.
Email: bhavani.g24@gmail.com.

Ch.Venkatramaiah, Received his M.Tech Degree in VLSI Stream From Sathyabama University, Chennai, Tamilnadu And Working As Assistant Professor Dept Of Ece In R.M.D Engineering College Chennai, Tamilnadu, India.
Email: venkat.cha dalavada@gmail.com.