Code Word Generation for Reconfigurable Crosstalk Elimination

K.Manasa Lakshmi [1] S.Neelima [2]

[1]PG Student (M.Tech), Dept. of ECE, Gandhiji Institute of Science & Tech., Jaggaiahpeta, AP, India

[2]Associate professor, Dept. of ECE, Gandhiji Institute of Science & Tech., Jaggaiahpeta, AP, India

Abstract— One of the greatest challenges in Deep Sub-Micron (DSM) design is inter-wire crosstalk, which becomes significant with shrinking feature sizes of VLSI fabrication processes and greatly limits the speed and increases the power consumption of an IC. This monograph presents approaches to avoid crosstalk in on-chip. The research work presented in the first part of this monograph focuses on crosstalk avoidance with bus encoding, one of the techniques that effectively mitigates the impact of on-chip capacitive crosstalk and improves the speed and power consumption of the bus interconnect . This technique encodes data before transmission over the bus to avoid certain undesirable crosstalk conditions and thereby improves the bus speed and/or energy consumption. We first derive the relationship between the inter-wire capacitive crosstalk and signal speed as well as power, and show the data pattern dependence of crosstalk.

Keywords— Cross Talk, Reconfigurable, Code word Generation, Elimination.

I. INTRODUCTION

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. Code words generated in memory-based crosstalk avoidance encoding satisfy a specified crosstalk upper-bound requirement with respect to the previous codeword onthe bus. The codeword is generated based not only on the current data word, but also on the previously transmitted codeword. Such a code is therefore called "memory based code". The decoder also uses the current received codeword and either the previous received codeword or the previous decoded data word to decode the current data word. The memory depth of CACs we consider is one. Memory less crosstalk avoidance codes are attractive due to the simplicity of their CODEC implementations. Memory-based codes, on the other hand, offer the advantage of lower bus area overhead as reported in [35, 99]. However, they generally require more complex CODECs. We first present a memory-based 4C-free CAC design that requires simple CODEC logic and 33% bus wiring overhead. We then present two generic memory-based code design methods: the pruning-based search and the ROBDD based search.

Conventional wisdom doubts that bus encoding would ever become a viable solution to the problem of inter-wire capacitive crosstalk due to its area overhead and CODEC complexity associated with such techniques, we presented some of the recent research in overcoming these obstacles and showed that such techniques can be highly efficient. Following a brief overview of techniques in crosstalk avoidance at the beginning of, we examined the phenomenon of crosstalk in on-chip busses and illustrated that capacitive coupling between adjacent parallel bus wires is the main contributor to the speed and power degradation in deep sub-micron signal busses. We introduced the notion of *effective capacitance* and showed that both delay and energy consumption of busses are linearly proportional to it. We introduced a bus classification system for binary busses to make the subsequent discussion clear. This classification system was later generalized to multi-valued busses. With the establishment of this classification system, we introduced our first set of crosstalk avoidance codes—memory less codes. These codes have the inherent advantage of requiring simpler CODECs. We showed that there exists a trade off between the area overhead and the speed-up factor for all the codes. Generally, higher the speed-up, the larger the area overhead. Some of the memory less codes we discussed include the FPF code (3C-free) [36], FTF code (3C-free code proposed by B. Victor [100]), 2C-free code and 1C-free configurations [35]. We showed that mathematical induction based approaches can be used to effectively generate code words for both 3C-free codes and the 2C-free code. Such inductive approaches also allow us to compute the code cardinality without explicitly enumerating all the code words.

We found that based on our computation, 3C-free memory less codes require44% overhead, which is significantly lower than the conventional static shielding approaches. The overhead is higher for 2C-free codes but still lower than shielding techniques. As part of a complete solution for memory less codes, we also discussed in detail some efficient CODEC design approaches for these codes. The bus-partition based solution is simple and allows us to control the

CODEC size and speed. Our solutions based on the Fibonacci Numeral System are advantageous over conventional solutions because of their significantly reduced area, simple design, deterministic mapping, reusable modular structure and most importantly, the extreme low complexity and high speed. We also investigated memory-based codes and showed that they are more efficient in terms of area overhead. A 4C-free code design was given as one of the simplest memory-based code, with a 33% overhead. We showed that graph pruning can be used to represent code words and transitions. Based on a graph representation, we showed that codeword pruning is a simple way of finding the minimum encoding overhead. It, however, requires a large amount of memory and run time. We proposed an ROBDD-based approach that represents all the illegal code words canonically and generates code words efficiently. It also suffers from the same limitations as the codeword pruning technique. As a result, realistic memory based CACs would need to partition large busses into smaller groups. Finally, we investigated the crosstalk avoidance coding for multi-valued busses. We first generalized the classification system into the multi-valued realm and showed that the speed and power of a bus are related to the effective crosstalk coefficient,*Xeff*, a parameter defined in a manner similar to the effective capacitance *Ceff* in the binary bus case. Noting that the supply voltages are scaling down continuously, using more than 4 values in a multi-valued bus is impractical. Hence, we proposed two code designs specifically for the ternary bus. These codes use a bit-to-bit direct mapping which makes the encoding and decoding trivial. By allowing flexibility onthe signal polarity and establishing a few coding rules, the encoded busses eliminate bus patterns with high crosstalk and significantly reduce the probability of other high crosstalk patterns. These codes exploit the low signal swing of the multi-valued signals. As a result, we were able to achieve a significant speed improvement and power saving.

We believe that the issue of crosstalk avoidance in on-chip interconnects to reduce delay and power will attract more and more attention as the technology marches head. Beyond the research presented in this monograph, there is room for further improvement. For example, efficient CODEC design for memory-based codes is a subject worth further investigation. Multi-valued logic based bus interconnect is another area that holds significant promise. Crosstalk in serial busses is also an important topic. Even though there have been several publications on combining crosstalk avoidance codes with other codes such as forward error correction codes, an practical, efficient design has yet to be discovered.

Some of the techniques discussed are not limited to bus crosstalk avoidance; instead, they can be applied to other areas. For example, the FPF code can be of particular interest to certain applications in digital communication, where certain patterns are designated to carry higher priority messages, such as emergency massages. Our FPF codes have 44% overhead, which is much less compared to existing schemes.

The FPF code may also be used in serial data communications, in which back to back toggles are not allowed. Another potential application is interference management in wireless communications. The transmission in a cell (or sector) is considered as Interference to an adjacent cell (or sector). The accumulated interference can greatly Summary of On-Chip Crosstalk Avoidance.

Reduce the capacity of the victim cell. It is conceivable that some of the crosstalk avoidance schemes may be applied for interference management. Another interesting property of the FPF code lies in its spectrum. Since the high frequency components are suppressed, we expect that by applying FPF encoding, high bandwidth efficiency can be achieved. This is extremely valuable for many wireless communication systems.

II. PROPOSED SYSTEM

Consider a graph where every codeword is a node and every edge represents a transition that does not involve any two neighboring lines switching in opposite direction. 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...) . The codebook is implemented as random logic. This code is referred to as opposing transitions elimination encoding (OTEE) in the following thesis.

Method designed as a transition code where every rising or falling edge is represented by a 1 while a stable logic value is represented by a 0. As an example, a codeword 101 indicates that in a three bit bus, lines on extreme ends are undergoing transitions while the middle one is stable. As a result, avoiding any codeword with consecutive 1's eliminates any simultaneous transitions on neighboring bus lines. Although code proposed 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 thesis.

The code proposed achieves the goal by eliminating any vector that contains pattern b, b_dash, b on neighbouring bus lines during any given time frame. In order to maintain simpler code book and encoder logic, authors rely on bus partitioning and group complement bits. This, in turn, results in significant overhead. As discussed in subsequent sections, it is possible to extend the proposed method to implement such code without partitions and group complement such that total routing area for the bus is significantly reduced. This kind of encoding eliminates crosstalk induced slowdown while maintaining transition speedup, this code is referred to as slowdown elimination encoding (SEE).

III. IMPLEMENTATION PROCESS

The sequence of steps followed in the proposed technique is depicted in Figure. The Process begins with making an initial attempt to identify the correlation between codewords 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.

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 codewords 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 codewords that are present in the newly formed 2-bitcode set. As an example consider the code in [19] that forbids consecutive 1's in a codeword. The codeword 11 present in class (1, 2) is invalid under this encoding technique and is flagged by the software tool. The codeword 11 is formed by appending a leading 1 to a member of class (1, 1). As a result the solid edge connecting class (1, 1) and class (1, 2).

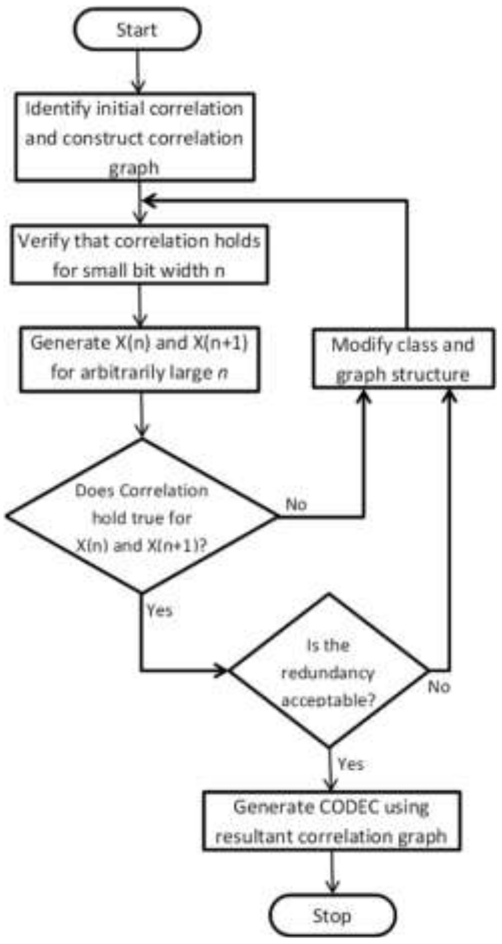
Start
Identify initial correlation and construct correlation graph
Verify that correlation holds for small bit width n
Generate X(n) and X(n+1) for arbitrarily large n
Modify class and graph structure
Does Correlation hold true for X(n) and X(n+1)?
No
Yes
Is the redundancy acceptable?
No
Yes
Generate CODEC using resultant correlation graph
Stop

Figure 1: Steps of proposed implementation strategy

Implementation of NAT, OTEE, SEES:

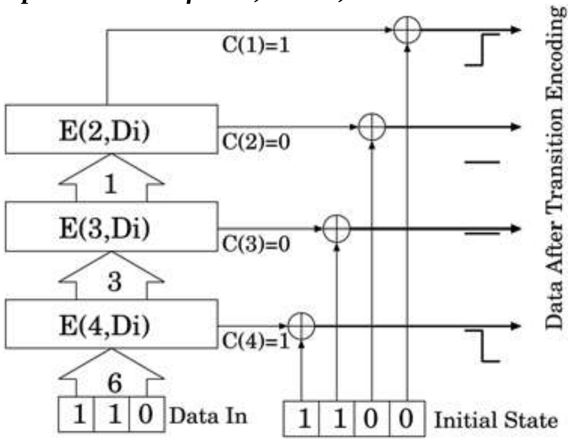

Figure 2: NAT encoder implementation

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

Table 3.1: Mapping 4-Bit NAT Code to 3-Bit Data

Th
sur
sum, the k*th*bit isset. 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:
n_i=1 (*C*(*i*) ∗ *Fb*(*i*+ 1))
where C(i) is the i*th*code bit and Fb(i+1) is the i+1*st* Fibonacci number. The 3-bit data broken-up as a sum of k*th* 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 codewords. 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 consecutives in

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

Table 3.2: Resultant Bus states

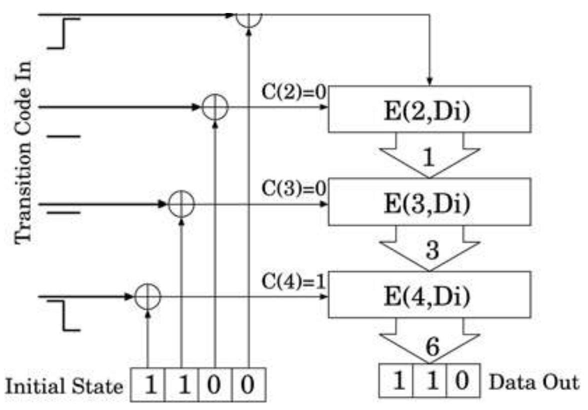

Figure 3: NAT decoder block

The codewords ensures absence of adjacent transitions thereby avoiding crosstalk induced delay. Code bits are individually led 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 depicted in Fig. 22.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 multibit adders and multiplexers. An n-bit encoder/decoder will requiren-bit adders and multiplexer.

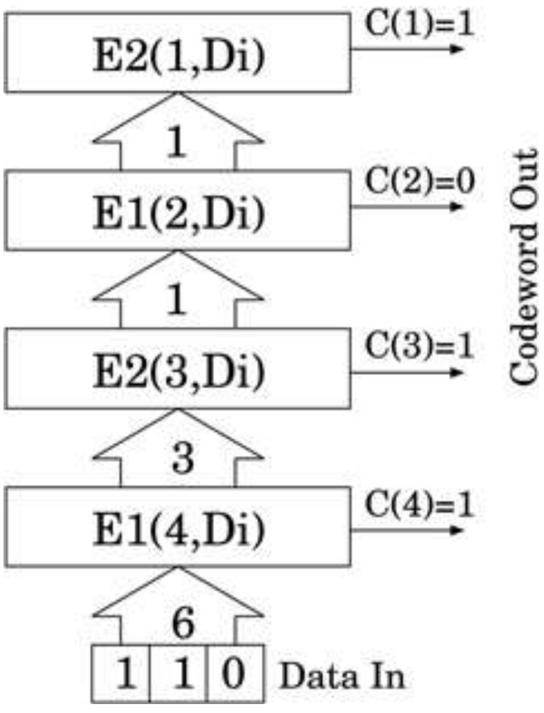

Figure 4: Otee encoder block

.

Since 3≥Fb (3), C3=1 and D*o*=3-Fb (3)=1. The third block performs E1(2, 1). Since 1_Fb(3), C2=0 and D*o*=1. Finally, the fourth block performs E2(1, 1). Since 1≥Fb (1), C1=1 and D*o*=1-Fb(1)=0. Irrespective of the input data, D*o* from the last block is always 0. The path traversed on the correlation graph that corresponds to input data 110 is shown in bold. 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 thencoder, if Fb(k)is subtracted from the input data, the k*th*bit is set. The decodermust recover the original data by adding Fb(k) for every k*th*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:
n_i=1(*C*(*i*) ∗ *Fb*(*i*))
where C(i) is the i*th* code bit and Fb(i) is the i*th* Fibonacci number. The breakdown of 3-bit data into

Fb(4), Fb(3), Fb(2), Fb(1)and the resultant mapping is shown in Table I.
TABLE I Mapping 4-Bit OTEE Code to 3-Bit Data Fig. 18. OTEE decoder block. NAT code bit generation. The block diagram of the decoder using D(k,D*i*) function blocks is given in Fig. 17. Function D(k,D*i*) 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). The remaining data words are mapped to the first (2d -|(0, n)|) elements of (1, n). There maining elements of (1, n) are unmapped. For (4,3,2)-NAT code mapping shown in Table II (|(1, n)|+|(0, n)|)=2d ; hence, there are no unmapped codewords.

Hardware architecture for the reconfigurable crosstalk elimination

The selection line is used for selecting OTEE, NAT, SEE

The truth table is as follows

Inp 1	Inp 2	Inp 3	S1	S 2	Opt 1	Opt 2	Opt 3	Opt 4
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	0	1	1	0	0	0	0	1
0	1	0	0	0	0	0	1	0
0	1	0	0	1	0	1	0	0
0	1	0	1	0	0	0	1	1
0	1	1	0	0	0	1	0	0
0	1	1	0	1	0	1	0	1
0	1	1	1	0	0	1	1	0
1	0	0	0	0	0	1	0	1
1	0	0	0	1	0	1	1	1
1	0	0	1	0	0	1	1	1
1	0	1	0	0	1	0	0	0
1	0	1	0	1	1	1	0	0
1	0	1	1	0	1	0	0	0
1	1	0	0	0	1	0	0	1
1	1	0	0	1	1	1	0	1
1	1	0	1	0	1	0	0	1
1	1	1	0	0	1	0	1	0
1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	0	0

The above truth table is the functionality of the otee see and nat encoding and decoder continues as the same in conventional .the outputs mentioned are k map designed to obtain the equation and then to hardware architech.

Output1 kmap

Ab\cs1 s2	00 0	00 1	01 1	01 0	11 0	11 1	10 1	10 0
00								
01								
11	1	1		1	1		1	1
10					1		1	1

Kmap equation

$$ab\overline{s2}+ab\overline{cs1}+c\overline{s1}a+a\bar{b}cs1\overline{s2}$$

For output 2

Kmap

Ab\cs1s2	000	001	011	010	110	111	101	100
00								
01		1			1		1	1
11		1			1		1	
10	1	1		1			1	

Equations

$$a\bar{b}(\overline{cs\,2})+\mathrm{a}\,\bar{\mathrm{b}}(\overline{\mathrm{s}1}\mathrm{s}\,2)+\overline{\mathrm{s}1}s2b+cs1\overline{s2}$$

For output3

Ab\cs1s2	000	001	011	010	110	111	101	100
00								
01	1			1	1			
11							1	1
10		1		1				

Equations

$$\bar{a}b(\overline{cs\,2})+ab(\overline{cs\,1})+\mathrm{a}\,\bar{\mathrm{b}}\overline{cs1}s2+a\overline{bc}s1\overline{s2}+\bar{a}b(cs\,1\overline{s\,2})$$

For output 4

Ab\cs1s2	000	001	011	010	110	111	101	100
00	1				1		1	1
01				1			1	
11	1	1		1			1	
10	1	1		1				

Equation

$$a\overline{cs1}+ab\overline{cs2}+ab\overline{s1}s2+\bar{c}s1\overline{s2}b+\bar{c}s1\overline{s2}a+a\bar{b}c\overline{s2}$$

IV. RESULTS & DISCUSSIONS

Number of luts	Correlation graph	Fibonacci
OTEE	5	8
NAT	43	90
SEE	8	12

Table 4.1. LUT report for correlation graph and Fibonacci

References

1. Kedar Karmarkar, Spyros Tragoudas, “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 free 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. Berkeley predictive technology modeling homepage. http://www-device.eecs.berkeley.edu/bptm/
11. BSIM4 offcial release site. http://www-evice.eecs.berkeley.edu/bsim3/bsim4.html
12. Fibonacci number (FromWikipedia) http://en.wikipedia.org/wiki/Fibonacci_number
13. Moore’s law. http://en.wikipedia.org/wiki/Moore´s_law
14. Numeral system. http://en.wikipedia.org/wiki/Numeral_system.

Authors Profile:

K.Manasa Lakshmi is pursuing her M. Tech in Department of Electronics and Communication Engineering at Gandhiji Institute of Science & Technology, Jaggaiahpeta.
Her specialization is VLSID.

Mrs. S.Neelima is an Associate professor in the Department of Electronics and Communication Engineering at Gandhiji Institute of Science & Technology, Jaggaiahpeta. She is persuing her Ph.D in the field ov VLSI. She has published several papers on her interested area of VLSI signal processing.