Error-Correction and Crosstalk Avoidance in DSM Busses

Ketan N. Patel and Igor L. Markov
Department of EECS, University of Michigan, Ann Arbor 48109-2122
{knpatel,imarkov}@eecs.umich.edu

ABSTRACT

Aggressive process scaling and increasing clock rates have made crosstalk noise an important issue in VLSI design. Switching on adjacent wires on long bus lines can increase delays and lead to logic faults, particularly when adjacent lines switch with opposite transitions. At the same time system-level interconnects have also become more susceptible to other less predictable forms of interference such as noise induced by power grid fluctuations, electromagnetic interference, and alpha particle radiation. Previous work has treated these systematic and non-systematic forms of noise separately.

In this paper we propose to make system level interconnects more robust using encoding that simultaneously addresses error correction requirements and crosstalk noise avoidance. This is more efficient than satisfying these requirements separately. We give algorithms for obtaining optimal encodings, and present a practical class of codes called boundary shift codes. We evaluate the overhead of our method and make comparisons to using error correction with simple shielding.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*VLSI*

General Terms

Design

Keywords

bus encoding, error-correction, crosstalk noise, DSM busses

1. INTRODUCTION

As packing densities and clock rates continue to increase in VLSI circuits, bus crosstalk is becoming an increasingly important issue to consider in optimizing performance. The severity of the interference is highly correlated with the particular switching patterns on the bus. For example, if two adjacent wires have simultaneous rising transitions, both transitions speed up and a hold violation is possible. Similarly, a rising transition on one wire can cause a neighboring wire to falsely transition and lead to a logic fault. However, the most detrimental switching pattern on two neighboring wires is opposite transitions; this causes both transitions to slow down and can lead to a setup violation. A number of techniques have been used to mitigate crosstalk interference including specialized routing strategies [4], intentionally skewing signal transition timings on adjacent wires [6], both active and passive shielding [7, 9], and signal encoding to minimize conflicts [14, 2].

Busses can also suffer from other forms of interference such as that due to power grid fluctuations, electromagnetic interference, or alpha particle radiation. Typically, these effects are unpredictable, and difficult to prevent. For example, power grid fluctuations occur when large numbers of gates switch simultaneously. The effect aggravates as the number of gates supplied by the same power grid increases and is not closely linked to the particular switching patterns on the bus. In the nanotechnology context, where circuits are manufactured with a significant proportion of faults, occasional errors may be unavoidable. In these cases using solely preventive techniques may not be effective and active error correction may be necessary.

In the past, a number of proposals have considered error correction for busses to combat both crosstalk [3] and other non-systematic interference [12, 1]. However, these methods only correct errors after they occur; they make no attempts to prevent the interference. Conversely, other bus encoding techniques have been used to prevent crosstalk, but do not correct errors [14, 2]. For example, Victor and Keutzer [14] have proposed encoding the bus to prohibit opposite transitions on adjacent wires. They call such codes self-shielding as they are an alternative to placing shielding wires in between lines. However their method is purely preventive and does not correct errors if they do occur.

In this paper we combine the goals of providing self-shielding and error correction in designing the bus encoding. This is more effective than the conventional approach of treating these separately. Since our method not only reduces the interference due to crosstalk, but also corrects errors, it can be useful in a variety of applications including nanotechnology, low-swing signaling, and radiation-hardened circuits. We provide algorithms for generating optimal bus encodings, and present a general construction method for a practical class of codes. Encoding and decoding circuits are given for specific codes.

2. NOTATION AND TERMINOLOGY

Our basic goal is to design a bus encoding scheme that avoids crosstalk while simultaneously offering error-correction capability. Our problem is closely related to that considered by Victor et al. [14], and we follow much of their notation and terminology. In

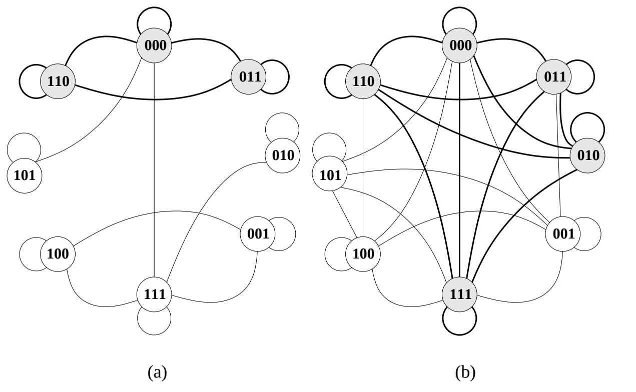

Figure 1: Graphical representation of optimal 3-bit self-shielding codes: (a) single error-detecting, (b) no error control.

addition we use some basic background in classical error correction [10, 11, 13] and self-checking [15, 8] techniques. We model the bus as an n-bit communication channel. During each *signaling interval* the encoding scheme is used to select and transmit an n-bit word, called a *codeword*, from a possibly dynamic set called a *codebook*. The codebook is dynamic in the sense that it can be a function of previously transmitted codewords. We call the overall encoding scheme a *code*.

A code is *memoryless* if it uses a fixed codebook. We define the *rate* of the code as $\log_2 |C_{min}|$, where $|C_{min}|$ is the number of codewords in the smallest codebook. This is the minimum number of bits that can be encoded during every signaling interval.

We say a pair of codewords contains an *invalid transition* if transitioning from one codeword to the other causes adjacent bits to switch in opposite directions. For example, the following codewords contain an invalid transition by bits 3 and 4:

$$\begin{array}{cccccccccc} c_1 \rightarrow & 0 & 1 & \mathbf{1} & \mathbf{0} & 0 & 0 & 1 & 1 \\ c_2 \rightarrow & 1 & 1 & \mathbf{0} & \mathbf{1} & 1 & 0 & 1 & 0 \end{array}$$

Such transitions are undesirable because they increase crosstalk noise. A code is *self-shielding* if it does not allow invalid transitions; this terminology comes from the conventional technique of using shielding wires to prevent crosstalk. Following Victor et al. [14], we assume that neighboring wires are routed in parallel. Therefore, the encoded bus is effectively self-shielding.

In addition to avoiding invalid transitions, we want to be able to differentiate our codewords reliably even in the presence of errors (bit flips). This is possible if the codewords in the codebook have a large enough *Hamming distance* between them, that is, if they differ in enough bit positions. For example, if the codewords have a minimum Hamming distance of three between them, we can correct any single error since the "noisy" codeword must be closer to the original codeword than to any other. In general, if the minimum Hamming distance between any two codewords is d, then we can either correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors, or detect up to $d-1$ errors [10].

We should note that while the codes considered in this paper are simultaneously self-shielding and capable of correcting errors, they do not necessarily remain self-shielding in the presence of errors. In other word, an error occurring on the bus could lead to an invalid transition, though the bus encoding prevents such transitions in error-free operation.

A binary code is *linear* if the bitwise sum (mod 2) of any two codewords is also a codeword. A linear code can be efficiently represented by an independent set of basis codewords [13]. All other codewords can then be formed by a linear combination of these. A standard representation of a linear code is the *generator matrix*, a matrix whose rows are are an independent set of basis codewords. For example, a generator matrix for the code $\{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}$ is

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The generator matrix provides a simple way to map information bits to codewords: multiply the generator matrix by a column vector of information bits. For example, G maps $[1\ 1\ 1]^t$ to the codeword $[1\ 0\ 0\ 1]$. A length n binary linear code encoding k bits with minimum Hamming distance d is called an $[n,k,d]$ code.

3. MEMORYLESS CODES

In a memoryless self-shielding code we use a fixed codebook and therefore the transitions between any two codewords must be valid. Victor et al. [14] use a graphical model to represent these constraints. All n bit words are represented by a vertex in the graph with edges between two vertices if transitions between the corresponding words are valid (see Figure 1b). A memoryless self-shielding code consists of a set of vertices forming a *clique*, that is, a set of vertices with edges between every pair in the set. The maximum rate code is the largest clique in the graph. One such clique is highlighted for the graph in Figure 1b. Properties of the graphs for these codes can be used to obtain a formula for the size of the largest clique for general n [14].

This graphical model easily generalizes for additional constraints. For example, in our case we would like the minimum Hamming distance between any two codewords to be at least d. We can add this constraint to the model by only placing edges between two vertices if the corresponding words satisfy the Hamming distance constraint in addition to the valid transition condition. Figure 1a shows such a graph for a single error-detecting ($d = 2$), self-shielding code on three wires. A clique of maximum size is highlighted.

```
function k_Clique(G, k)
{  V = {vert. of G sorted by ↑ degree}
   if (G is complete & |V|≥k)
      return 1
   for (v ∈ V)
   {  remove vert. w/ degree < k from G & V
      if |V| < k return 0
      G_s = {subgraph w/ vert. incident to v}
      if (k_Clique(G_s, k)==0)
        remove v from G & V
      else
        return 1
   }
}
```

Algorithm 1: Solving Max-Clique

In general the max-clique problem is NP-complete [5, pp. 53-56], however there are a number of heuristic algorithms that perform well for many cases. Here we used a simple pruning algorithm (see Algorithm 1) coded in MATLAB. The algorithm determines if a clique of size k exists in the graph. For the $n = 9$ case, the algorithm took approximately 45 minutes on a Sun-Sparc. The results are shown in Table 1. The maximum rates for memoryless codes without error correction (column two) match those given by Victor et al. [14].

	Without Memory		**With Memory**		
Wires	self-shielding	single error-correcting self-shielding	self-shielding	single error-correcting self-shielding	single error-correcting boundary shift
3	2.32	1.00	2.32	1.00	1.00
4	3.00	1.00	3.17	1.00	-
5	3.70	1.59	3.91	2.00	2.00
6	4.39	2.32	4.75	2.32	-
7	5.09	2.58	5.52	3.17	3.00
8	5.78	3.17	6.34	3.59	-
9	6.48	3.81	7.14	4.25	4.00

Table 1: Maximum rate for self-shielded codes. The first column gives the number of wires used for the encoding. The remaining columns give the maximum number of bits that can be encoded by the self-shielding codes specified by the column headings.

4. CODES WITH MEMORY

For codes with memory, the codebook can be a function of the previously transmitted codeword. For this case we use two graphs to represent the problem. The first graph G_1 has a vertex for each n-bit word, and an edge between two vertices if they form a valid transition. The second graph G_2 contains the same vertices as G_1 but has edges between vertices if the Hamming distance between them is greater than $d-1$. Intuitively, G_1 and G_2 represent the self-shielding and the Hamming distance constraints, respectively.

In a self-shielding minimum distance d code with rate $\log_2 M$ each codeword must be able to transition to a size M subset of codewords that are at least a Hamming distance d apart from each other. In our graphical representation, each vertex in graph G_1 must have edges to at least M vertices that form a clique in graph G_2. We can determine if such a code exists by continuing to eliminate vertices that do not meet this condition from both graphs. If we are left with a non-empty set of vertices all meeting the condition, they form a code with the desired properties, otherwise no such code exists. We list the pseudo code for this algorithm here (see Algorithm 2).

```
function Exist_Code(G1, G2, k)
{  V = {vert. of G1}
   Vin = {}
   while (V not empty)
   {  if (V==Vin)
         return 1
      v = vertex from {V-Vin} of lowest degree
      G_s = {subgraph of G2 w/ vertices
           incident to v in G1}
      if (k-Clique(G_s, k)==0)
         remove v from G1, G2 & V
         Vin = {}
      else
         add v to Vin
   }
   return 0
}
```

Algorithm 2: Search for code with memory

Note that a series of calls to k_Clique is embedded in this algorithm. We found the computation time for specific cases can vary greatly for different implementations of this function. Using Algorithm 1, we were able to calculate the optimal codes for $n \leq 8$, however the computation time became prohibitive for the $n=9$ case and an alternate algorithm was necessary.

For final case we used an integer linear programming (ILP) formulation for the max-clique problem. A binary variable is assigned to each vertex in the graph with a one denoting membership in the clique. For every pair of vertices that do not share an edge, we impose the constraint that both cannot be members of the clique. The size of the max-clique is found by maximizing the sum of the variables, i.e., the number of members in the clique. We used the linear optimization tool CPLEX to perform the optimization. Results for the $n=9$ case took several hours on a Sun-Sparc. We should note that while the ILP implementation was more efficient than Algorithm 1 for this case, it was not as efficient for others. For example, we found that cases in Section 3 required approximately 5 times as much computation time using the ILP formulation compared with Algorithm 1. All of the results are shown in Table 1. Those in columns two and four, for codes without error correction, were previously found by Victor et al. [14].

5. BOUNDARY SHIFT CODES

In the previous section we presented an algorithm for finding maximum rate error-correcting self-shielding codes, however this algorithm does not provide a practical encoder or decoder. In fact decoding these codes may require significant resources. Furthermore, the algorithm becomes computationally infeasible for moderate to large bus sizes. In this section we give a general construction method for practical codes.

We define a *dependent boundary* in a codeword as a position where two adjacent bits differ. We denote the location by the position of the leftmost bit of the boundary. If two codewords do not share any dependent boundaries, they cannot form an invalid transition. For example, consider the following codewords:

$$\begin{array}{cccccccccc} c_1 \rightarrow & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ c_2 \rightarrow & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0. \end{array}$$

Here c_1 and c_2 have dependent boundaries $\{1,3,5\}$ and $\{2,4,7\}$, respectively. Since there is no overlap, the transition must be valid.

Using this property, we note that if a codebook has codewords with only even dependent boundaries, then performing a 1-bit circular right shift yields a new codebook with no even dependent boundaries. Since the two codebooks do not have overlapping dependent boundaries, we can alternate between the two to obtain a self-shielding code. We call this a *boundary shift code*.

For this construction we need an error correction code with no odd dependent boundaries. Let C be an $[n,k,d]$ code, and let C' be formed by duplicating each bit position in C. Then C' is a $[2n,k,2d]$ code with no odd dependent boundaries, since every bit in an odd bit position is followed by a copy. By alternating between C' and a shifted version of it, we obtain a $[2n,k,2d]$ self-shielding code. In addition, *puncturing* C' in the last bit position, i.e., removing the last bit in every codeword, yields a $[2n-1,k,2d-1]$ code.

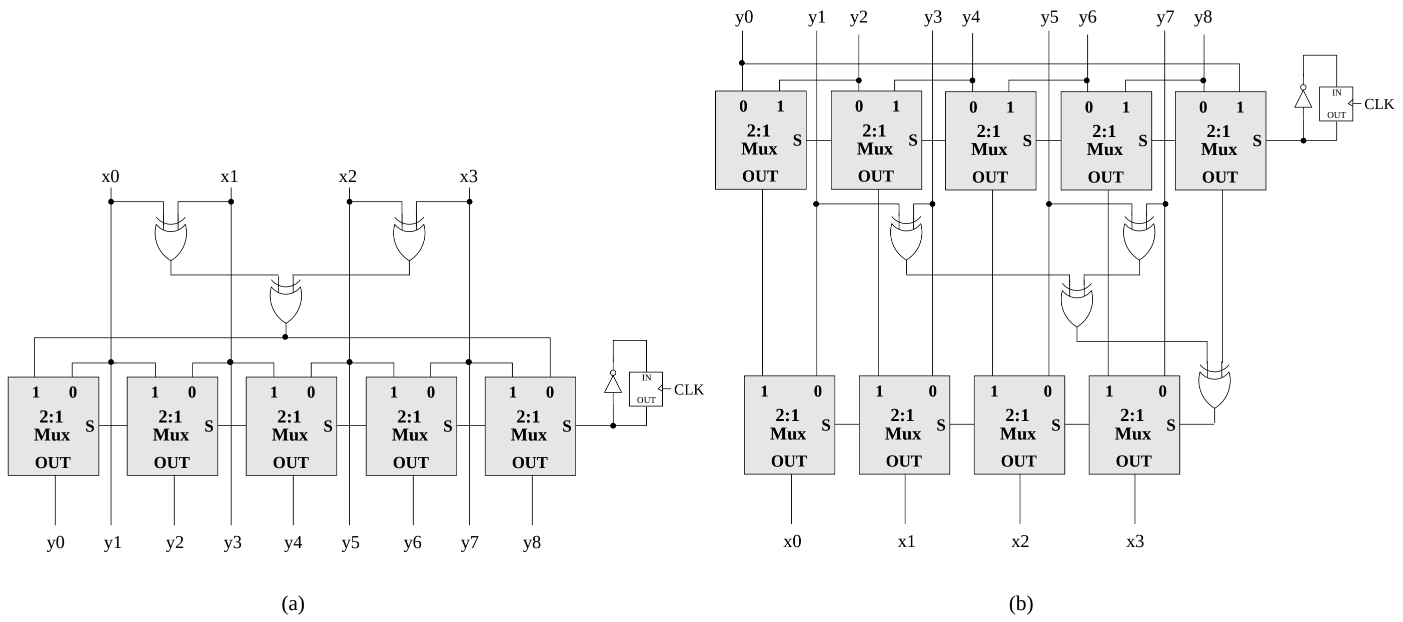

Figure 2: Encoding (a) and decoding (b) circuits for $[9,4,3]$ code.

Using a single parity check code in the above construction gives an infinite class of single error-correcting codes. In a $[k+1,k,2]$ single even parity check code the k data bits are appended with a final bit chosen to make the parity of the codeword even. Applying our construction, we obtain a $[2k+1,k,3]$ single error-correcting self-shielding code.

In Table 1 we compare the rates of these codes to the optimal rates. Their performance is better than that of the optimal memoryless codes, and comparable to that of the optimal codes with memory, particularly when integer rates are of interest.

As an example consider the $[9,4,3]$ boundary shift code with generator matrices:

$$G_0 = \left[\begin{array}{cc|cc|cc|cc|c} \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{array}\right]$$

$$G_1 = \left[\begin{array}{c|cc|cc|cc|cc} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \end{array}\right]$$

Generator matrices G_0 and G_1 are used for encoding during even and odd clock cycles respectively. Equivalently, G_0 can be used for all cycles with the output then right shifted for odd cycles. The following example illustrates how this code would be used to encode a 4-bit bus. The intermediate *pre-shifted* output is shown for clarity.

time	input		pre-shifted output		output
0	1 0 1 0	→	1 1 0 0 1 1 0 0 0	→	1 1 0 0 1 1 0 0 0
1	0 1 1 1	→	0 0 1 1 1 1 1 1 1	→	1 0 0 1 1 1 1 1 1
2	1 0 0 0	→	1 1 0 0 0 0 0 0 1	→	1 1 0 0 0 0 0 0 1
3	0 1 0 0	→	0 0 1 1 0 0 0 0 1	→	1 0 0 1 1 0 0 0 0

We duplicate each input bit and append the parity check at the end yielding the pre-shifted output. If the clock cycle is odd we perform a 1-bit circular right shift before transmitting.

At the receive side, we first undo the right shift if the clock cycle is odd, then decoding can be done by majority vote, where the two "copies" of the desired bit are augmented by a third generated by taking the sum (mod 2) of one copy of each of the other information bits and the parity check. For example, in an even cycle three independent copies of the first information bit are given by y_0, y_1 and $(y_2+y_4+y_6+y_8) \mod 2$. A single error will affect at most one of the three copies and is therefore correctable. The following example shows how noisy versions of the codewords in the previous example would be decoded after unshifting. The highlighted bits correspond to errors.

noisy output		majority vote		data
1 **0** 0 0 1 1 0 0 0	→	(1**0**1) (000) (111) (000)	→	1 0 1 0
0 0 1 1 1 1 1 1 **0**	→	(00**1**) (11**0**) (11**0**) (11**0**)	→	0 1 1 1
0 1 0 0 0 0 0 0 1	→	(**0**11) (00**1**) (00**1**) (00**1**)	→	1 0 0 0
0 **1** 1 1 0 0 0 0 **0**	→	(0**11**) (11**0**) (00**1**) (00**1**)	→	**1** 1 0 0

The last codeword is decoded incorrectly as it contains two errors, and is therefore beyond the code's error correcting capability.

6. PRACTICAL CONSIDERATIONS

Since the codes constructed in the previous section are based on very simple error-correcting codes, they can be encoded and decoded efficiently. Figures 2a and 2b show an encoder and decoder respectively for the $[9,4,3]$ code. These circuits can be generalized in a straightforward manner for larger single error-correcting codes. Gate counts and maximum circuit depths are given in Table 3 for a range of bus sizes and closed form expressions are given for the general case.

For large bus sizes the increased circuit depth may lead to significant delay. This can be reduced by breaking the bus into smaller sub-busses with shielding wires inserted between them. This results in a slight increase in the number of wires and gates needed, but limits the circuit depth. In addition, it also increases the error correction capability, since single errors in each sub-bus can then be corrected independently.

Code	Advantages	Drawbacks
Optimal memoryless	- encoding/decoding by combinational circuits	- relatively low rate - code construction difficult - decoder may be complex
Optimal with memory	- maximum rate	- code construction difficult - encoder/decoder may be very complex - possible error propagation
Boundary Shift Codes	- achieve higher rate than opt. memoryless codes - scalable construction - simple encoder/decoder - integer rates; systematic	- achieve slightly lower rate than optimal codes

Table 2: Comparison of boundary shift codes and optimal self shielding error-correcting codes.

A potentially useful feature of the proposed codes is that they are *systematic*, that is, the information bits are embedded in the encoded codeword, and can therefore be obtained without any decoding logic. For example, the information bits of the $[9,4,3]$ code are given by bits y_1, y_3, y_5 and y_7. Of course, simply using these bits rather than decoding the full codeword sacrifices the error correction capabilities of the code. However one can imagine a scenario where error correction may only be necessary for certain destinations on the bus that are relatively far from the source, while other destinations may opt for error detection or simply picking the information bits off of the encoded codeword.

A possible concern with codes with memory is that since future codebooks may depend on the current transmitted codeword, an uncorrectable error has the potential to propagate and cause additional errors. A clear advantage of boundary shift codes is that they do not suffer from error propagation since the codebook does not depend on the choice of previous codewords transmitted, rather it is only a function of the time index. As long as the source and destination are synchronized, no error propagation will occur.

A summary of advantages and drawbacks of the codes presented in this paper are shown in Table 2. The boundary shift codes have a higher rate (i.e., require fewer additional wires) than the optimal memoryless codes, and only a slightly lower rate than the optimal codes with memory. The primary advantages of boundary shift codes over the optimal codes are a simple scalable construction, and a practical encoder and decoder.

In Table 4 we compare boundary shift codes to other shielding and error correction methods. Compared to the self-shielding codes proposed by Victor et al. [14] our codes are not only able to prevent crosstalk but also tolerate errors; this additional protection comes at the expense of a rate reduction (see Table 1). When compared to placing shielding wires between bus lines, our technique provides error correction in addition to self-shielding without appreciably reducing the rate, though some encoding and decoding logic is required. This is a more effective approach than treating error correction and shielding separately. *For example, to protect a 4-bit bus from single errors requires a total of at least 7 bits [10]. Adding shielding wires to this encoded bus to prevent crosstalk then results in a 13-bit bus. In comparison, the* $[9,4,3]$ *single error-correcting boundary shift code illustrated in Section 5 achieves the same error protection for the 4-bit bus using only 9 bits.* Precharging busses to prevent crosstalk is also an alternative, however it can be costly in terms of power consumption and does not provide error correction.

Bus Size	Wires	Encoder		Decoder	
		Gates	Delay	Gates	Delay
4	9	10	3 gates	15	4 gates
8	17	18	4 gates	27	5 gates
16	33	34	5 gates	51	6 gates
32	65	66	6 gates	99	7 gates
64	129	130	7 gates	195	8 gates
n	$2n+1$	$2n+2$	$\lceil \log_2 n \rceil + 1$	$3n+3$	$\lceil \log_2 (n+1) \rceil + 1$

Table 3: Encoder/decoder gate counts and delay for single error-correcting boundary shift codes.

Method	Advantages	Drawbacks
Shielding Wires	- crosstalk prevention - simple construction - no additional delay	- no error correction - additional wires
Self-Shielding Codes [2, 14]	- crosstalk prevention - relatively high rate	- no error correction - encoding/decoding logic - additional wires - additional delay
Error-Correcting Codes [1, 3, 12]	- error correction	- no crosstalk prevention - encoding/decoding logic - additional wires - additional delay
Bus Precharging	- crosstalk prevention - no additional wires	- no error correction - high power consumption
Proposed Boundary Shift Codes	- crosstalk prevention - error correction	- encoding/decoding logic - additional wires - additional delay

Table 4: Comparison of boundary shift codes and other shielding and error correction methods.

7. CONCLUSIONS

In this paper we have considered the problem of designing bus encoding schemes that provide both crosstalk prevention and active error correction. The former helps reduce the crosstalk interference, while the latter corrects faults after they occur, regardless of their origin. This is an important improvement over previous methods which have been designed to either reduce crosstalk interference or provide error correction, but not both. Our joint approach is particularly applicable to scenarios, such as nanotechnology and radiation hardened circuits, where random errors are a concern in addition to crosstalk interference. We give algorithms for finding optimal codes for various constraints and code parameters.

One of the most significant contributions of this paper is a practical class of error-correcting self-shielding codes called boundary shift codes. These codes are derived from conventional error-correcting codes, which have been studied extensively in the literature. For the specific case of single error-correcting boundary shift codes we give gate level encoding and decoding circuits.

Future work includes case studies of faults in DSM busses and abstractions to more accurate fault models. Simulations of our methods using realistic circuit models would be useful in evaluating the effectiveness of our approach. We also plan to generalize the codes considered here to include additional constraints, such as a limit on the power consumption.

8. REFERENCES

[1] D. Bertozzi, L. Benini, and B. Ricco. Energy-efficient and reliable low-swing signaling for on-chip buses based on redundant coding. *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 93–96, 2002.

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

[3] M. Favalli and C. Metra. Bus crosstalk fault-detection capabilities of error-detecting codes for on-line testing. *IEEE Trans. on VLSI Systems*, pp. 392–396, Sept. 1999.

[4] T. Gao and C. L. Liu. Minimum crosstalk channel routing. *Proc. IEEE/ACM Intl. Conf. on Computer Aided Design*, pp. 692–696, Nov. 1999.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[6] K. Hirose and H. Yasuura. A bus delay reduction technique considering crosstalk. *Proc. Design, Automation and Test in Europe Conf. and Exhibition 2000*, pp. 441–445, 2000.

[7] H. Kaul, D. Sylvester, and D. Blaauw. Active shields: A new approach to shielding global wires. *Proc. IEEE Great Lakes Symp.*, pp. 112–117, Apr. 2002.

[8] P. K. Lala. *Self-Checking and Fault-Tolerant Digital Design*. Academic Press, Inc., 2001.

[9] K. M. Lepak, I. Luwandi, and L. He. Simultaneous shield insertion and net ordering under explicit RLC noise constraint. *Proc. Design Automation Conf.*, pp. 199–202, June 2001.

[10] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1996.

[11] V. Pless. *Introduction to the Theory of Error-Correcting Codes*. Wiley, New York, 1998.

[12] D. Rossi, V. van Dijk, R. Kleihorst, A. Nieuwland, and C. Metra. Coding scheme for low energy consumption fault-tolerant bus. *Proc. IEEE On-line Testing Workshop*, pp. 8–12, 2002.

[13] S. A. Vanstone and P. C. van Oorschot. *An Introduction to Error Correcting Codes with Applications*. Kluwer, Boston, 1989.

[14] B. Victor and K. Keutzer. Bus encoding to prevent crosstalk delay. *Proc. IEEE/ACM Intl. Conf. on Computer Aided Design*, pp. 57–69, Nov. 2001.

[15] J. F. Wakerly. *Error detecting codes, self-checking circuits and applications*. Elsevier North-Holland, Inc., New York, 1978.