Efficient On-Chip Crosstalk Avoidance CODEC Design

Chunjie Duan, Victor Cordero, Sunil P. Khatri

Abstract

Interconnect delay has become a limiting factor for circuit performance in Deep Sub-Micron designs. As the crosstalk in an on-chip bus is highly dependent on the data patterns transmitted on the bus, different crosstalk avoidance coding schemes have been proposed to boost the bus speed and/or reduce the overall energy consumption. Despite the availability of the codes, no systematic mapping of datawords to codewords has been proposed for CODEC design. This is mainly due to the non-linear nature of the crosstalk avoidance codes (CAC). The lack of practical CODEC construction schemes has hampered the use of such codes in practical designs. This work presents guidelines for the CODEC design of the forbidden pattern free crosstalk avoidance code(FPF-CAC). We analyze the properties of the FPF-CAC and show that mathematically, a mapping scheme exists based on the representation of numbers in the Fibonacci numeral system. Our first proposed CODEC design offers a near-optimal area overhead performance. An improved version of the CODEC is then presented, which achieves theoretical optimal performance. We also investigate the implementation details of the CODECs, including design complexity and the speed. Optimization schemes are provided to reduce the size of the CODEC and improve its speed.

Very Large Scale Integration (VLSI) Systems, IEEE Transactions

Efficient On-Chip Crosstalk Avoidance CODEC Design

Chunjie Duan, *Member, IEEE,* Victor Cordero and Sunil P. Khatri, *Member, IEEE*

Abstract—Interconnect delay has become a limiting factor for circuit performance in Deep Sub-Micron designs. As the crosstalk in an on-chip bus is highly dependent on the data patterns transmitted on the bus, different crosstalk avoidance coding schemes have been proposed to boost the bus speed and/or reduce the overall energy consumption. Despite the availability of the codes, no systematic mapping of datawords to codewords has been proposed for CODEC design. This is mainly due to the non-linear nature of the crosstalk avoidance codes (CAC). The lack of practical CODEC construction schemes has hampered the use of such codes in practical designs. This work presents guidelines for the CODEC design of the "forbidden pattern free crosstalk avoidance code" (FPF-CAC). We analyze the properties of the FPF-CAC and show that mathematically, a mapping scheme exists based on the representation of numbers in the Fibonacci numeral system. Our first proposed CODEC design offers a near-optimal area overhead performance. An improved version of the CODEC is then presented, which achieves theoretical optimal performance. We also investigate the implementation details of the CODECs, including design complexity and the speed. Optimization schemes are provided to reduce the size of the CODEC and improve its speed.

Index Terms—crosstalk, on-chip bus, Fibonacci number, CODEC

I. Introduction

As VLSI technology has marched into the *Deep Sub-Micron* (DSM) regime, new challenges are presented to circuit designers. As one of the key challenges, the performance of bus based interconnects has become a bottleneck to the overall system performance. In large designs (e.g. SOCs) where long and wide global busses are used, interconnect delays often dominate logic delays.

Once negligible, crosstalk has become a major determinant of the total power consumption and delay of on-chip busses. The impact of crosstalk in on-chip busses has been studied as part of the effort to improve the power and speed character-istics of the on-chip bus interconnects. Figure 1 illustrates a simplified on-chip bus model with crosstalk. C_L denotes the *load capacitance* seen by the driver, which includes the re-ceiver gate capacitance and also the parasitic wire-to-substrate parasitic capacitance. C_I is the inter-wire coupling capacitance between adjacent signal lines of the bus. In practice, this bus structure is electrically modeled using a distributed RC network, after including the parasitic resistance of the wire as well (not shown in Figure 1). For DSM processes, C_I is

Chunjie Duan is with Mitsubishi Electric Research Labs, 201 Broadway, Cambridge, MA 02139 USA

Victor Cordero and Sunil P. Khatri are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA
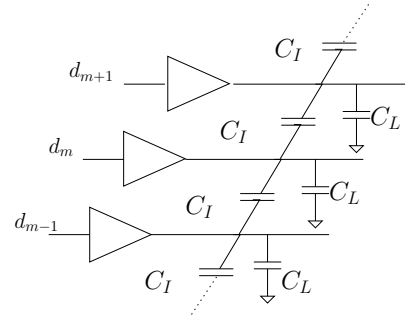

Fig. 1. On-chip Bus model with Crosstalk

much greater than C_L [7]. Based on the energy consumption and delay models given in [1], the energy consumption is a function of the total crosstalk over the entire bus. The delay, which determines the maximum speed of the bus, is limited by the maximum crosstalk that any wire in the bus incurs. It has been shown that reducing the crosstalk can boost the bus performance significantly [1][5].

Different approaches have been proposed for reducing crosstalk by eliminating specific data transition patterns. Some schemes focus on reducing the energy consumption, while others focus on minimizing the delay. Certain schemes offer improvements in both. In this paper, we focus on crosstalk avoidance for delay reduction.

As the crosstalk is dependent on the data transition patterns on the bus, patterns can be classified based on the severity of the crosstalk they impose on the bus. A more detailed explanation of pattern classification is given in Section II-A. The general idea behind techniques that improve on-chip bus speed is to remove undesirable patterns that are associated with certain classes of crosstalk. Among the proposed schemes, some are more aggressive than others (they remove more patterns and achieve higher speed improvements). Different schemes incur different area overheads since they requires additional wires, spacing between wires, or both.

As one of the simplest techniques to eliminate the crosstalk induced delay penalty, passive shielding inserts passive (e.g. grounded) shield wires between adjacent active data lines [10]. This technique can reduce the bus delay by nearly 50%. However, it requires doubling the number of wires and hence incurs a 100% area overhead. **Crosstalk can also be exploited to speed up the bus. Techniques such as active shielding can reduce the bus delay by up to 75% [8][11] at the price of 200% or more area overhead.**

It has been discovered relatively recently that encoding the bus can eliminate some classes of data patterns with

much lower area overhead compared to the shielding techniques [5][6]. These codes are commonly referred to as *Crosstalk Avoidance Codes* (CACs). CACs can be further divided into two categories: memory-less and memory-based. The memory-based coding approaches generate a codeword based on the previously transmitted code and the current dataword to be transmitted [6][9]. On the receiver side, the data is recovered based on the received codewords from the current and previous cycles. The memory-less coding approaches use a fixed code book to generate a codeword to transmit, solely based on the input data. The corresponding receiver decoder uses the current received codeword as the only input to recover the data.

The theoretical lower bound of the area overhead for memory-based codes is lower compared to memory-less codes. However, the memory-based CODECs are much more complex and the only known codeword generation method is an exhaustive search and pruning based method.

Several different types of memory-less CACs have been proposed. **The code designs are discussed in [5][4][3][6].** These codes offer the same degree of delay reduction as the passive shielding technique, with much less area overhead (ranging from 44% to 68%). Unfortunately, none of the referred papers addresses the mapping between datawords and codewords for the CODECs. So far, all the CODEC design approaches are based on *bus partitioning* (which breaks a big bus into a number of small groups (lanes) and applies CAC coding on each group independently). Such an approach has to deal with the crosstalk across the group boundaries. Several different schemes are proposed to handle this inter-group crosstalk, such as group inversion and bit overlapping [5][4]. In all cases, more wires are needed and therefore the overall area overhead is higher than the theoretical lower bound.

In this work, we offer a systematic CODEC construction solution for the *forbidden-pattern-free crosstalk avoidance code* (FPF-CAC). The mapping scheme we propose is based on the representation of numbers in the Fibonacci numeral system. We show that all datawords can be represented in the Fibonacci-based numeral system with FPF vectors. We propose several different coding schemes that allow the CODECs to be constructed *for any arbitrary bus size*. With such a systematic mapping, the CODEC for a wider bus is constructed by a simple extension of the CODEC for a smaller bus. The first CODEC proposed in the paper is proven to have near-optimal area overhead performance. We further offer an improved coding scheme that achieves optimal overhead performance. We also propose modifications to our near-optimal CODEC that will reduce the complexity and improve the delay performance of the CODEC.

The key contributions of this paper include:

- We define a deterministic mapping scheme for the FPF-CAC based on the *Fibonacci-based binary numeral system*.
- Based on the mapping scheme, we propose coding algorithms that allow systematic CODEC constructions so that the CODEC for a wider bus is obtained as an extension of the CODEC for smaller bus.
- We show that the CODEC gate count grows quadratically

with bus size as opposed to the exponential growth for the existing approaches.

The remainder of the paper is organized as follows: Section II first provides some background on delay and power analysis of the bus in the presence of crosstalk. The classification of crosstalk is given in Section II-A. In Section II-C, the forbidden-pattern-free cross avoidance code is defined and its performance is discussed, including codeword generation and overhead computation. A lower bound of the area overhead is established. Section III focuses on the construction of the CODEC for FPF-CAC. We give the mathematical basis for the CODEC construction and discuss the overhead performance of different CODECs. In Section IV, we investigate the circuit implementation details of the proposed CODECs. Experimental results are also presented. Conclusions are drawn in Section V.

Notation: For clarity, throughout this paper, unless specified otherwise, an n bit bus is represented by a vector $b_n b_{n-1}...b_2 b_1$, with b_n being the most significant bit and b_1 the least significant bit.

II. FORBIDDEN PATTERN BASED CAC

A. Crosstalk classification

As stated in the previous section, the degree of crosstalk in an on-chip bus is dependent on data transition patterns on the bus. Based on the model shown in Figure 1, the delay τ_j of the jth wire in a data bus is given as [1]:

$$\tau_j = abs(k \cdot C_L \cdot \Delta V_j + k \cdot C_I \cdot \Delta V_{j,j-1} + k \cdot C_I \cdot \Delta V_{j,j+1}) \quad (1)$$

where k is a constant determined by the driver strength and wire resistance, ΔV_j is the voltage change on the jth line and $\Delta V_{j,k} = \Delta V_j - \Delta V_k$ is the relative voltage change between the jth and k^{th} line. Since on-chip busses are generally full-swing binary busses, we can assume that the two output voltage levels are V_{dd} and 0V and hence $\Delta V_j \in \{0, \pm V_{dd}\}$ and $\Delta V_{j,k} \in \{0, \pm V_{dd}, \pm 2 \cdot V_{dd}\}$. If we let $\lambda = \frac{C_I}{C_L}$, Equation 1 can be rewritten as:

$$\tau_j = k \cdot C_L \cdot V_{dd} \cdot abs(\delta_j + \lambda \cdot \delta_{j,j-1} + \lambda \cdot \delta_{j,j+1}) \quad (2)$$

Here $\delta_j \in \{0, 1\}$ is the normalized voltage change on j^{th} line. $\delta_{j,j\pm1} \in \{0, \pm 1, \pm 2\}$ is the normalized relative voltage change on j^{th} line (relative to the $j+1^{th}$ or $j-1^{th}$ line). The δ_j term corresponds to the intrinsic delay and the remaining two terms correspond to the crosstalk induced delay. Since $\lambda \gg 1$, the first term has negligible contribution to the delay.

If we define $C_{eff,j}$ as the *effective total capacitance* of the driver of jth line, we have:

$$C_{eff,j} = C_L \cdot abs(\delta_j + \lambda \cdot \delta_{j,j-1} + \lambda \cdot \delta_{j,j+1}) \quad (3)$$

and

$$\tau_j = k \cdot V_{dd} \cdot C_{eff,j} \quad (4)$$

From Eq. 4, we get $min(C_{eff,j}) = C_L$ and $max(C_{eff,j}) = (1 + 4 \cdot \lambda)C_L$ depending on the transition pattern on the wire of interest as well as its immediate neighbors on either side. Crosstalk patterns are classified as *0C, 1C, 2C, 3C* and *4C* patterns respectively as shown in Table I. The last column in Table I gives example transition patterns on three adjacent bits of the bus $b_{j+1}b_jb_{j-1}$.

Class	C_{eff}	Transition patterns
0C	C_L	$000 \rightarrow 111$
1C	$C_L(1+\lambda)$	$011 \rightarrow 000$
2C	$C_L(1+2\lambda)$	$010 \rightarrow 000$
3C	$C_L(1+3\lambda)$	$010 \rightarrow 100$
4C	$C_L(1+4\lambda)$	$010 \rightarrow 101$

TABLE I

CLASSES OF CROSSTALK

The speed of the data bus is determined by $max\{C_{eff,j}\}$ over all bits in the bus. An uncoded bus that transmits random data experiences $max\{C_{eff,j}\} = (1+4\lambda)C_L$ and therefore the speed of such bus must be designed to accommodate the $4C$ crosstalk delay. Based on Table I and Equation 4, when $\lambda \gg 1$, by eliminating $4C$ crosstalk on ALL lines in the bus, we can increase the maximum speed of the bus by $\sim 33\%$. If we can eliminate the $3C$ AND $4C$ crosstalk on all lines, the bus can be sped up by $\sim 100\%$. This has been verified by experiments in [5].

B. Energy consumption of busses with crosstalk

The focus of this paper is on crosstalk avoidance code that speeds up the busses, we shall, however, point out that the crosstalk also impacts the average bus power consumption. A detailed discussion is given in [1] and the overall energy consumption for a given bus transition is

$$E_{total} = \sum_{j=1}^{n} E_j^L + \lambda \sum_{j=1}^{n} E_j^I$$
$$= \sum_{j=1}^{n}(1 + C_{eff,j} \cdot \lambda)C_L \cdot \Delta V_j \cdot V_j \quad (5)$$

where $C_{eff,j}$ is the effective capacitance on the jth wire defined earlier.

Equation 5 shows that the bus energy is the summation of the energy consumption of each given bit and that the crosstalk also has effect on energy consumption. Therefore avoiding crosstalk could result in reduction of the overall energy consumption of a bus as we show later.

C. Forbidden pattern based Crosstalk Avoidance

The *forbidden pattern* based crosstalk avoidance code was first proposed in [5]. The *forbidden patterns* are defined as 3-bit patterns "101" and "010". A code is **forbidden pattern free** (FPF) if there is no forbidden pattern in any three consecutive bits. As examples, 1101110 is not forbidden

pattern free; 1100110 is FPF. It has been shown in [5] that for a code that contains only FPF codewords, the bus that transmits only these codewords will experience maximum crosstalk of no greater than 2·C. Therefore, by encoding the datawords to FPF codewords, we can speed up the bus by $\sim 100\%$. This type of code is referred herein as *forbidden pattern free crosstalk avoidance code* (FPF-CAC).

The FPF-CAC can be generated using an inductive procedure [5]. Let S_m be the set of m-bit FPF-CAC codewords, an m-bit vector $V_m^i = b_m b_{m-1}..b_1$ is a codeword. Any codeword $V_m^i \in S_m$ can be considered as concatenating $V_{m-1}^i = b_{m-1}b_{m-2}..b_1$ with bit b_m, where $V_{m-1}^i \in S_{m-1}$.

The following is the inductive procedure that generates FPF codewords, where "·" is the concatenation operator:

Algorithm 1 FPF codeword generation

$S_2 = \{00, 01, 10, 11\}$
for $m \geq 3$ **do**
   $S_m = \{\}$;
   for $\forall V_{m-1}^i \in S_{m-1}$ **do**
      if $b_{m-1}^i b_{m-2}^i = 00$ or 11 **then**
         add $0 \cdot V_{m-1}^i$ and $1 \cdot V_{m-1}^i$ to S_m;
      else if $b_{m-1}^i b_{m-2}^i = 01$ or 10 **then**
         add $b_{m-1}^i \cdot V_{m-1}^i$ to S_m;
      end if
   end for
end for

Table II lists the codewords of the 3,4 and 5 bit FPF-CACs generated by Algorithm I.

2-bit	3-bits	4-bits	5 bit	
00	000	0000	00000	10000
01	001	0001	00001	10001
10	011	0011	00011	10011
11	100	0110	00110	11000
	110	0111	00111	11001
	111	1000	01100	11100
		1001	01110	11110
		1100	01111	11111
		1110		
		1111		

TABLE II

FPF-CAC CODEWORDS FOR 2,3,4 AND 5 BIT BUSSES

From Algorithm I, we can see that for each $m - 1$ bit codeword $V_{m-1}^i \in S_{m-1}$ with last two digits $b_{m-1}^i = b_{m-2}^i$, two m-bit codewords can be generated. For V_{m-1}^i with the last two digits $b_{m-1}^i \neq b_{m-2}^i$, only one m-bit codeword can be generated. The total number of FPF-CAC codewords can be computed based on the following equations:

Definition 1: For an m-bit vector $b_m b_{m-1}...b_2 b_1$, we define the following quantities:

- $T(m)$ is the total number of distinct m-bit vectors.
- $T_g(m)$ is the total number of FPF vectors.

- $T_b(m)$ is the total number of non FPF vectors.
- $T_{gg}(m)$ is the number of FPF vectors satisfy $b_m^i = b_{m-1}^i$.
- $T_{gb}(m)$ is the number of FPF vectors satisfy $b_m^i \neq b_{m-1}^i$

For the base case, a 3-bit bus: $(m = 3)$

- $T_g(3) = 6$;
- $T_b(3) = 2$;
- $T_{gg}(3) = 4$;
- $T_{gb}(3) = 2$;

For busses with more than 3-bits: $(m > 3)$

$$T_g(m) = 2 \times T_{gg}(m-1) + T_{gb}(m-1) \tag{6}$$
$$T_{gg}(m) = T_{gg}(m-1) + T_{gb}(m-1) \tag{7}$$
$$T_{gb}(m) = T_{gg}(m-1) \tag{8}$$

Based on Algorithm I and the definitions of T_g, T_{gg} and T_{gb}, we get:

$$T_g(m) = T_{gg}(m) + T_{gb}(m) \tag{9}$$
$$T_{gg}(m) = T_g(m-1) \tag{10}$$

Equation 6 can be re-written as:

$$
\begin{aligned}
T_g(m) &= 2 \times T_{gg}(m-1) + T_{gb}(m-1) \\
&= (T_{gg}(m-1) + T_{gb}(m-1)) + T_{gg}(m-1) \\
&= T_g(m-1) + T_g(m-2)
\end{aligned}
\tag{11}
$$

The relationship shown in Equation 11 is the same as the relationship of elements in the Fibonacci sequence. (A more detailed discussion about Fibonacci sequence will be given in Section III.) With the initial conditions $T_g(2) = 2 \cdot f_3$ and $T_g(3) = 2 \cdot f_4$, we have:

$$T_g(m) = 2 \cdot f_{m+1} \tag{12}$$

where f_m is the mth element in the Fibonacci sequence.

$T_g(m)$ gives the maximum *cardinality* of the m-bit FPF-CAC code. To encode an n-bit binary bus into FPF-CAC code, the minimum number of bits needed m_{opt} is the smallest integer m that satisfies Equation 13. We can also compute the lower bound of the area overhead $OH(n)$, which is defined as the ratio between the additional area required for the coded bus and the area of uncoded bus.

$$n \leq \lfloor log_2(2 \cdot f_{m+1}) \rfloor \tag{13}$$
$$OH(n) = \frac{m-n}{n} \tag{14}$$

For the Fibonacci sequence, $\varphi = \lim_{k \to \infty} \frac{f_{k+1}}{f_k} = 1.618$, also known as the *golden ratio*, is the asymptotic ratio of two consecutive elements of the sequence [12]. Hence $\lim_{k \to \infty} f_k = c\varphi^k$ where c is a constant. Therefore, for large busses, the lower bound of the overhead is:

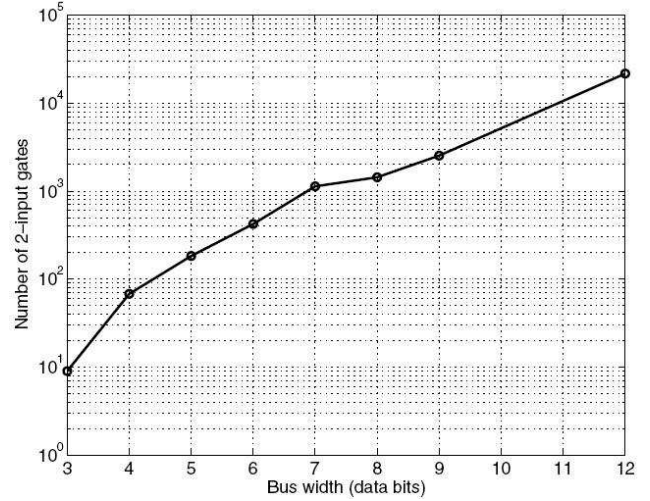$$OH_{min} \geq \frac{1}{log_2\varphi} - 1 \approx 44\% \tag{15}$$

Fig. 2. CODEC gate count, reprinted from [4] with permission from the authors. Copyright®IEEE

III. FPF-CAC CODEC DESIGN

As discussed in the previous section, the 3C and 4C crosstalk classes can be avoided if the bus is encoded using the FPF code. We provided the recursive procedure for generating the codewords and showed how to compute the total number of codewords and the lower bound for the area overhead. However, the mapping scheme between the input datawords and the output codewords was not discussed, nor was it shown how a CODEC for the FPF-CAC can be constructed.

Conceptually, the mapping between the datawords and the codewords is flexible, provided it can be reversed by the decoder. In the case when the size of the code book is not a power of two, a 1-to-1 mapping is not required. A 1-to-many mapping for certain datawords may reduce the CODEC complexity further.

When the data bus width is small, the CODEC can be implemented and the mapping flexibility can be exploited to optimize the speed and/or the area of the CODEC. However, as the data bus width increases, the CODEC size grows exponentially. Figure 2 shows the number of 2-input gates required for CODECs of data bus widths varying from 3 to 12^1. The total number of mapping permutations also grows rapidly. For example, to encode the 8-bit data to an 11-bit CAC bus, there are over 6×10^6 possible mapping permutations. In addition, the CAC codes are non-linear and therefore it is difficult to extend a mapping scheme for smaller busses to larger busses.

Several different schemes have been proposed for CODEC construction for FPF-CAC or other memory-less CACs [5][4][6]. These schemes are all based on *bus partitioning*, which breaks up a wide bus into smaller *groups* or *lanes* (typically 3 to 5 bits) and exhaustively searches for the optimal mapping that yields the most efficient CODEC for the groups. Unfortunately, in order to handle crosstalk across the group boundaries, these schemes all inevitably suffer from additional area overhead.

---

[1]Even though the CODEC is for a slightly different crosstalk avoidance code, we feel that the results can be used as a benchmark.

In this section, we propose two coding schemes that allow us to encode data to the FPF-CAC without partitioning the bus. These coding schemes allow us to systematically construct the FPF-CAC CODECs for busses of arbitrary size. By "systematically", we mean that the CODEC for a larger size bus is extended from the CODEC of a smaller bus. The gate counts of the proposed CODEC implementation roughly grow quadratically with respect to the bus size, instead of exponentially for previous approaches [4]. Both our schemes are based on the Fibonacci numeral system.

A. Fibonacci-based binary numeral system

The Fibonacci binary numeral system was first used in CAC designs in [3] for crosstalk avoidance coding. The paper proposed an inductive codeword generations algorithm for a type of CAC called *self-shielding* code[2]. The inductive algorithm is similar to the ones proposed in [5][6]. However, none of these papers address the mapping scheme and CODEC designs as we do in the rest this section.

A numeral system is *"a framework where numbers are represented by numerals in a consistent manner"* [13]. The most commonly used numeral system in digital design is the binary numeral system, which uses powers of two as the *base*. A number's binary representation is defined in Equation 16. The binary numeral system is *complete* and *unambiguous*, which means that each number has one and only one representation in the binary numeral system.

Definition 2:

$$v = \sum_{k=1}^{n} b_k \cdot 2^{k-1} \qquad b_k \in \{0,1\} \qquad (16)$$

$$= \sum_{k=1}^{m} d_k \cdot f_k \qquad d_k \in \{0,1\} \qquad (17)$$

The Fibonacci-based numeral system $N(F_m, \{0,1\})$ is the numeral system that uses Fibonacci sequence as the base. The definition of the Fibonacci sequence is given in Equation 18. A number v is represented as the summation of some Fibonacci numbers and no Fibonacci number is in the summation more that once as indicated in Equation 17.

Definition 3:

$$f_m = \begin{cases} 0 & \text{if m = 0,} \\ 1 & \text{if m = 1,} \\ f_{m-1} + f_{m-2} & \text{if m} \geq 2. \end{cases} \qquad (18)$$

Similar to the binary numeral system, the Fibonacci-based numeral system is complete and therefore any number can be represented. However, the Fibonacci-based numeral system is *ambiguous*. As an example, there are *six* 7-digit vectors in the Fibonacci numeral system for the decimal number 19: {0111101, 0111110, 1001101, 1001110, 1010001, 1010010}. For clarity, we refer to a vector in the binary numeral system as a *binary vector* or *binary code*; a vector in the Fibonacci numeral system as a *Fibonacci vector* or *Fibonacci code*. All

[2]In some literatures, this type of code is also called *forbidden transition free* code

the Fibonacci vectors that represent the same value are defined as *equivalent vectors*.

A very important property of the Fibonacci sequence that is used in the following discussions is given in Equation 19.

$$f_m = \sum_{k=0}^{m-2} f_k + 1 \qquad (19)$$

The n-bit binary vector has the range of $[0, 2^n - 1]$ and a total of 2^n values can be represented by n-bit binary vectors. From Equation 19, we know that the range of a m-bit Fibonacci vector is $[0, f_{m+2} - 1]$, where minimum value 0 corresponds to all the bits d_k being 0 and the maximum value corresponds to all d_k being 1. Therefore a total of f_{m+2} distinct values can be represented by m-bit Fibonacci vectors.

B. Near-Optimal CODEC

We first propose a coding scheme that converts the input data to a forbidden pattern free Fibonacci vector. The code is near-optimal since the required overhead is no more than 1 bit more than the theoretical lower bound given in Equation 15.

Theorem 1: $\exists d_m d_{m-1}...d_2 d_1 = v$, $d_m d_{m-1}...d_2 d_1 \in N(F_m, \{0,1\})$ and is FPF, $\forall v \in [0, f_{m+2} - 1]$

Theorem 1 states that for any number $v \in [0, f_{m+2} - 1]$, there exists at least one m-bit Fibonacci vector $d_m d_{m-1}...d_2 d_1 = v$ that represents this number and is *forbidden pattern free*.

In order to prove Theorem 1, we first derive the following corollaries:

Corollary 1: The following two m-bit Fibonacci vectors are equivalent: $d_m d_{m-1}...d_3 01$ and $d_m d_{m-1}...d_3 10$. In other words, $d_m d_{m-1}...d_3 01 \equiv d_m d_{m-1}...d_3 10$.

Proof: Since $f_2 = f_1 = 1$, it is obvious that the last two digits are interchangeable.

∎

Corollary 2: For an m-bit Fibonacci vector $d_m d_{m-1}..d_2 d_1$, if three consecutive bits $d_k d_{k-1} d_{k-2}$ have a value 100, replacing them with 011 produces an equivalent vector.

Proof: This can be proven based on the definition of Fibonacci sequence given in Equation 18.

∎

Corollary 3: For an m-bit Fibonacci vector $d_m d_{m-1}..d_2 d_1$, if a number of consecutive bits (slice) in the vector $d_k d_{k-1}...d_{k-n}$ have a pattern of 0101...0100 (alternating 0 and 1 except the last two bits), replacing the slice with the pattern 0011...1111 (all bits are 1 except the first two bits) produces an equivalent vector.

Proof: From the right to left, we can recursively replace 100 with 011 (Corollary 2) until $d_{k-1}d_{k-2}d_{k-3}$ has been replaced.

∎

Corollary 4: For an m-bit Fibonacci vector $d_m d_{m-1}..d_1$, if a slice in the vector $d_k d_{k-1}...d_{k-n}$ has a pattern of 1010...1011 (alternating 1 and 0 except the last two bits), replacing the slice with the pattern 1100...0000 (first two bits are 1s and the other bits are 0s) produces an equivalent vector.

Proof: From the right to left, we can recursively replace 011 with 100 (Corollary 2) until $d_{k-1}d_{k-2}d_{k-3}$ has been replaced. ∎

The proof of Theorem 1 is given as follows:

Proof:

- $\exists d_m d_{m-1}..d_1 = v, \quad d_m d_{m-1}..d_1 \in N(F_m, \{0,1\}),$
  $\forall v \in [0, f_{m+2}-1].$
  In other words, for any number $v \in [0, f_{m+2}-1]$, there exists at least one m-bit Fibonacci vector $d_m d_{m-1}..d_1$. This follows from the completeness of the Fibonacci number system.
- If the vector $d_m d_{m-1}..d_1$ is *not* FPF, we can produce at least one equivalent vector that is FPF by performing some or all of the following procedures:
  - If the vector ends with a forbidden pattern (101 or 010), there exists an equivalent vector that ends with 110 or 001 (Corollary 1).
  - If any slice in the vector contains forbidden pattern, they can be replaced with a pattern that is forbidden pattern free using Corollary 3, 4. ∎

By proving Theorem 1, we show the existence of an FPF Fibonacci vector for any number v in the range $[0, f_{m+2}-1]$. The coding algorithm that encodes a given number v to an FPF Fibonacci vector is given in Algorithm 2:

Algorithm 2 Near-Optimal FPF-CAC Encoder

FPF-CAC(v)
\\ MSB stage:
if $v \geq f_{m+1}$ **then**
  $d_m = 1;$
  $r_m = v - f_m;$
else
  $d_m = 0;$
  $r_m = v;$
end if
\\ other stages:
for k = m-1 to 2 **do**
  if $r_{k+1} \geq f_{k+1}$ **then**
    $d_k = 1;$
  else if $r_{k+1} < f_k$ **then**
    $d_k = 0;$
  else
    $d_k = d_{k+1};$
  end if
  $r_k = r_{k+1} - f_k \cdot d_k;$
end for
\\LSB
$d_1 = r_2;$
return $(d_m d_{m-1}...d_1);$

Algorithm 2 shows that an m-bit FPF vector is generated in m stages. Each stage outputs one bit of the output vector (d_k) and the remainder (r_k) that is the input to the following stage. In the kth stage, the input r_{k+1} is compared to two Fibonacci numbers f_{k+1} and f_k. If $r_{k+1} \geq f_{k+1}$, d_k is coded as 1; If $r_{k+1} < f_k$, d_k is coded as 0; If the value r_{k+1} is in

between, d_k is coded to the same value as d_{k+1}. The remainder is computed accordingly based on the value of d_k. We shall refer the ranges $[f_{k+1}, f_{k+2})$, (f_k, f_{k+1}) and $[0, f_k)$ as the *force-1 zone*, *gray zone* and *force-0 zone* of the kth stage respectively. The most significant bit (MSB) stage is slightly different from other stages since no bit proceeds it. It encodes d_m by comparing the input v with only one Fibonacci number f_{m+1}.

The decoder is a straightforward implementation of Equation 17, which converts the Fibonacci vector back to the binary vector. Figure 3 shows the encoder and decoder structures based on Algorithm 2.
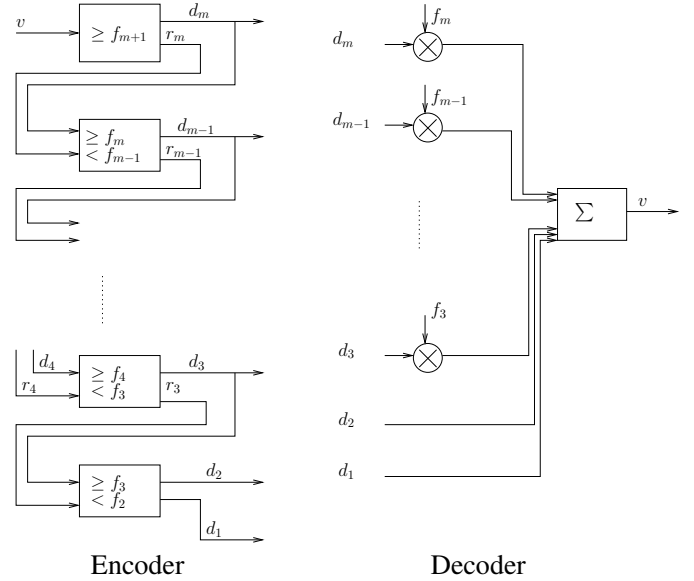

Fig. 3. CODEC structure (Based on Algrithm-2)

The correctness of Algorithm 2 can be proven by showing that if after the kth stage, the partially generated output vector $d_m...d_{k+1}d_k$ is FPF, adding the output of the $(k-1)^{th}$ stage, d_{k-1} will not introduce a forbidden pattern.

We first recognize that if $d_k = d_{k+1}$, no forbidden pattern will be produced regardless of the value of d_{k-1}. Therefore we only need to show that when $d_k \neq d_{k+1}$, d_{k-1} will satisfy $d_{k-1} = d_k$. Based on Algorithm 2, $d_k \neq d_{k+1}$ occurs only when d_k is *forced* to be 0 or 1. The proof is reduced to proving that when d_k is forced to one particular value, d_{k-1} will be coded the same value as d_k. The proof is given as follows:

Proof:

- If $d_k = 1$ and $d_k \neq d_{k+1}$ [d_k **in force-1 zone**]
  $\Rightarrow r_{k+1} \geq f_{k+1}$ and $r_k = r_{k+1} - f_k$
  $\Rightarrow r_k \geq f_{k+1} - f_k$
  $\Rightarrow r_k \geq f_{k-1}$ [d_{k-1} **not in force-0 zone**]
  $\Rightarrow d_{k-1} = 1$
- If $d_k = 0$ and $d_k \neq d_{k+1}$ [d_k **in force-0 zone**]
  $\Rightarrow r_{k+1} < f_k$ and $r_k = r_{k+1}$

$$\Rightarrow r_k < f_k \quad [d_{k-1} \textbf{ not in force-1 zone}]$$
$$\Rightarrow d_{k-1} = 0$$

∎

In Table III the complete 6-bit codewords generated using Algorithm 2 are listed as CODE-1. As stated earlier, the MSB stage is different from other stages since there is no preceding bit and, for the values in the gray zone, d_{m-1} can be coded to be either value. In Algorithm 2, we arbitrarily choose to code the MSB (d_m) to be 0 when the input value is in the gray zone. If we code d_m to be 1 for all values in the gray zone, we end up with a different set of codewords as listed in CODE-2 in the table. All codewords in both CODE-1 and CODE-2 are FPF. For clarity, we only list codewords in CODE-2 that are different from codewords in CODE-1.

Input	CODE-1	CODE-2
Decimal value	f_6 f_5 f_4 f_3 f_2 f_1 8 5 3 2 1 1	f_6 f_5 f_4 f_3 f_2 f_1 8 5 3 2 1 1
20*	1 1 1 1 1 1	
19*	1 1 1 1 1 0	
18*	1 1 1 1 0 0	
17*	1 1 1 0 0 1	
16*	1 1 1 0 0 0	
15	1 1 0 0 1 1	
14	1 1 0 0 0 1	
13	1 1 0 0 0 0	
12	0 1 1 1 1 1	1 0 0 1 1 1
11	0 1 1 1 1 0	1 0 0 1 1 0
10	0 1 1 1 0 0	1 0 0 0 1 1
9	0 1 1 0 0 1	1 0 0 0 0 1
8	0 1 1 0 0 0	1 0 0 0 0 0
7	0 0 1 1 1 1	
6	0 0 1 1 1 0	
5	0 0 1 1 0 0	
4	0 0 0 1 1 1	
3	0 0 0 1 1 0	
2	0 0 0 0 1 1	
1	0 0 0 0 0 1	
0	0 0 0 0 0 0	

TABLE III

7 BIT CODE BOOK

Based on Equation 19, we can easily see that the total numbers of codewords in both CODE 1 and CODE-2 are f_{m+2}, slightly smaller than the maximum cardinality of $2 \cdot f_{m+1}$ given in Equation 14. Since $f_{m+2} < 2 \cdot f_{m+1} < f_{m+3}$, we know for a given size input data vector n, the number of bits needed for the proposed CODEC is no more than 1 bit more than the minimum number of bits required, m_{opt}. Table IV lists the number of bits needed to encode the binary data from 3 to 32 bits: n_{in} denotes the number of bits for the input binary bus; m_{opt} the number of bits required for the optimal code; m_{no} the number of bits needed for the proposed CODEC and $\Delta(m)$ the difference between the two.

n_{in}	m_{opt}	m_{no}	$\Delta(m)$	n_{in}	m_{opt}	m_{no}	$\Delta(m)$
3	4	4	0	18	26	26	0
4	5	6	1	19	27	28	1
5	7	7	0	20	29	29	0
6	8	9	1	21	30	30	0
7	10	10	0	22	31	32	1
8	11	12	1	23	33	33	0
9	13	13	0	24	34	35	1
10	14	15	1	25	36	36	0
11	16	16	0	26	37	38	1
12	17	17	0	27	39	39	0
13	18	19	1	28	40	41	1
14	20	20	0	29	42	42	0
15	21	22	1	30	43	43	0
16	23	23	0	31	44	45	1
17	23	23	1	32	46	46	0

TABLE IV

OVERHEAD COMPARISON

C. Optimal CODEC

A quick examination shows that ALL the valid FPF-CAC codewords are actually listed in Table III: there are a total of $(f_{m+1} - f_m)$ codewords in CODE-2 that are not included in CODE-1. The total number of codewords in CODE-1 is f_{m+2}. Therefore the total number of distinct codewords in both CODE-1 and CODE-2 is:

$$\begin{aligned} T_g(m) &= f_{m+2} + f_{m+1} - f_m \\ &= f_{m+1} + f_m + f_{m+1} - f_m \\ &= 2 \cdot f_{m+1} \end{aligned} \quad (20)$$

We can see that the reason that the near-optimal codes do not reach the maximum cardinality is due to the redundant FPF Fibonacci vectors for the values in the *gray zone*. For a coding scheme to reach the optimal overhead performance, we need to remove this redundancy.

Table V shows how such modification is done. We simply move the codewords in the CODE-2 gray zone to the top of CODE-1. In doing so, the values these codewords represent are shifted by $f_{m+2} - f_m = f_{m+1}$. The MSB stage of the CODEC is modified to reflect this value shift in the new mapping scheme:

$$\begin{cases} d_m = 1, r_m = v - f_{m+2} & \text{if } v \geq f_{m+2}, \\ d_m = 1, r_m = v - f_m & \text{if } f_{m+2} > v \geq f_{m+1}, \\ d_m = 0, r_m = v & \text{if } v < f_{m+1} \end{cases} \quad (21)$$

The decoder is modified accordingly:

$$v = \begin{cases} \sum_{k=0}^{m-1} d_k \cdot f_k + f_{m+1} & \text{if } b_m b_{m-1}{=}10, \\ \sum_{k=0}^{m-1} d_k \cdot f_k & \text{otherwise} \end{cases} \quad (22)$$

Table V gives the codewords based on the optimal CODEC and the value each codeword represents. We can consider the codewords as having an extra bit as shown in the second column (XB) in the table. This bit is not transmitted on the bus since its value can be recovered by the decoder based on

the values of d_m and d_{m-1}.

Input	XB	OPT ENC						Encoder 2					
Decimal	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_6	f_5	f_4	f_3	f_2	f_1
value	13	8	5	3	2	1	1	8	5	3	2	1	1
25*	1	1	0	0	1	1	1						
24*	1	1	0	0	1	1	0						
23*	1	1	0	0	0	1	1						
22*	1	1	0	0	0	0	1						
21*	1	1	0	0	0	0	0						
20*	0	1	1	1	1	1	1						
19*	0	1	1	1	1	1	0						
18*	0	1	1	1	1	0	0						
17*	0	1	1	1	0	0	1						
16*	0	1	1	1	0	0	0						
15	0	1	1	0	0	1	1						
14	0	1	1	0	0	0	1						
13	0	1	1	0	0	0	0						
12	0	0	1	1	1	1	1	1	0	0	1	1	1
11	0	0	1	1	1	1	0	1	0	0	1	1	0
10	0	0	1	1	1	0	0	1	0	0	0	1	1
9	0	0	1	1	0	0	1	1	0	0	0	0	1
8	0	0	1	1	0	0	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1						
6	0	0	0	1	1	1	0						
5	0	0	0	1	1	0	0						
4	0	0	0	0	1	1	1						
3	0	0	0	0	1	1	0						
2	0	0	0	0	0	1	1						
1	0	0	0	0	0	0	1						
0	0	0	0	0	0	0	0						

TABLE V

CODE BOOK FOR OPTIMAL CODEC

The overhead performance of the optimal coding scheme reaches the theoretical lower bound given in Equation 15. Compared to the CODEC with near-optimal overhead performance, the optimal CODEC has added complexity. Using the optimal CODEC does not offer additional gain in overhead in the example here as the total number of distinct codewords increases from 21 to 26. As shown in Table IV, only when $f_m + 2 < 2^n < 2 \cdot f_{m+1}$ does the optimal coding offer saving of one bit.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULT

The encoder and decoder based on Algorithm 2 can be implemented using the structure illustrated in Figure 3. The encoder converts a n-bit binary vector $v = b_n..b_1$ to an m-bit vector $d_m d_{m-1}..d_1$.

The encoder consists of m stages. Figure 4 depicts the details inside the kth stage, where $k < m$. The inputs of the stage are outputs from the previous stage: d_{k+1} and r_{k+1}. and the outputs are d_k and r_k. There are subtractors and a 2-to-1 MUX.

For the near optimal encoder, the MSB stage is simpler than the other stages. For CODE-2, the MSB stage requires only
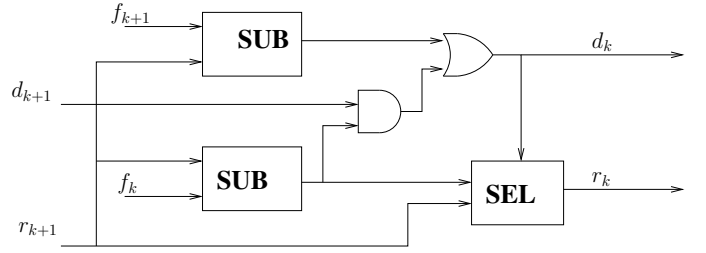

Fig. 4. Internal logic of the kth block

one subtractor and one selector. The MSB stage of the near-optimal CODEC can be modified to further simplify the logic. Let $b_n b_{n-1}...b_1$ be the binary input vector, if we let $d_m = b_n$ and r_m be $b_{n-1}b_{n-2}...b_1$. The mathematical expression of this mapping is:

$$v = b_n \cdot 2^{n-1} + \sum_{k=1}^{m-1} d_k \cdot f_k \qquad (23)$$

This modification is to simply code the output MSB bit to the input MSB bit. The outputs are still FPF codes because to code a n-bit binary code to an m-bit Fibonacci code, n and m satisfy: $2^n < f_{m+2}$ and we have:

$$\begin{aligned} 2^n &\le f_{m+2} < 2 \cdot f_{m+1} \\ \Longrightarrow 2^{n-1} &< f_{m+1} \end{aligned} \qquad (24)$$

Therefore the n bit input binary data can be broken into the MSB bit and a $(n-1)$ bit vector. We simply construct an encoder for the $n-1$ bit vector. The MSB bit controls the output bit value when the $(n-1)$ bit input value is in the gray zone.

Figure 5 shows the modified CODEC with the simplified MSB stage. On the encoder side, the MSB of the input b_n is mapped directly to the MSB of the output d_m. The rest of the input vector $b_{n-1}b_{n-2}...b_1$ becomes the input of the $(m-1)^{th}$ stage. On the decoder side, the first input of the summation stage is $d_m \cdot 2^{n-1}$, instead of $d_m \cdot f_m$ as in Figure 3.
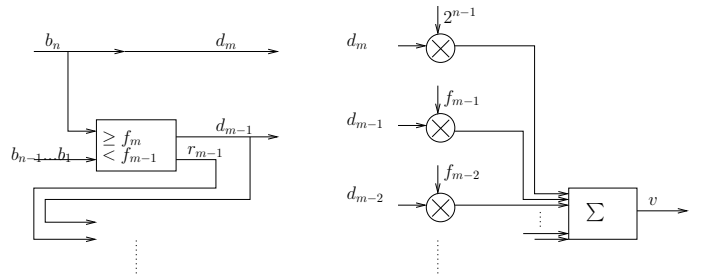

Fig. 5. CODEC structure with MSB optimization

To evaluate the complexity of the CODECs, we implemented the near-optimal CODEC in both an FPGA (Xilinx XC4VLX15-12 [15]) and ASIC (both a TSMC 90nm process [16] and a TSMC 130nm process). Figure 6 plots the resource and delay of the FPGA implementation and Figure 7 plots the equivalent gate counts of the encoder for input bus widths from 8 to 32 implemented using a TSMC 90 nm process [16]. For a input data width of 12 bits, the equivalent number of 2-input gates is 369. This is roughly two orders of magnitude lower than the gate count reported in Figure 2 [4]. For the input data

width of 32-bit, the total area is 17865 μm^2 and the equivalent gate count is 2,762. The sizes grow quadratically with the bus size, as we expected. The gate count for the 130nm are very close to the 90nm process.

Figure 8 compares the gate count for three different encoder implementations: a single level Look-up Table (LUT) implementation using random mapping, a single level LUT with Fibonacci numeral system mapping and the staged design proposed in this paper. All the designs are implemented using the same TSMC 90nm process. We can see clearly that the size of the LUT-based designs grow exponentially. The figure shows that the mapping schemes affect the encoder complexity: on average, a LUT-based encoder using the Fibonacci mapping is 50% smaller than a randomly mapped encoder. It also shows that for small busses, the proposed design is not advantageous compared with LUT based design. However, for busses with 8 or more bits, the proposed staged designs offer significant savings in terms of gate count.

Figure 6 and Figure 7 also show the delays of the encoder. Understandably, the delay increases as the input bus size goes up since the total delay is the accumulated delay of all stages. Unlike the single level implementation, our design allows pipeline stages to be inserted between stages to mitigate this problem.

Bus partitioning can also be used to reduce the total size and improve the speed of the decoder. Our experiment confirmed that the maximum input-to-output delay of an non-pipelined m-bit encoder is $\tau(m) \propto O(m^2)$. Reducing the bus in half can improve the bus speed by approximately a factor of 4. Similarly, the total area has the quadratic relation with the number of input bits and therefore partitioning the bus will reduce the total area by $\sim 50\%$.

The decoder structure is simpler than the encoder and has no ripple delay. However, as the bus width grows, the summation stage size goes up and more delay will be incurred. Note that there is no multiplication or AND operation in the actual implementation. Since f_k is a constant, it is a simple case of connecting d_k to the non-zero bit positions of f_k.
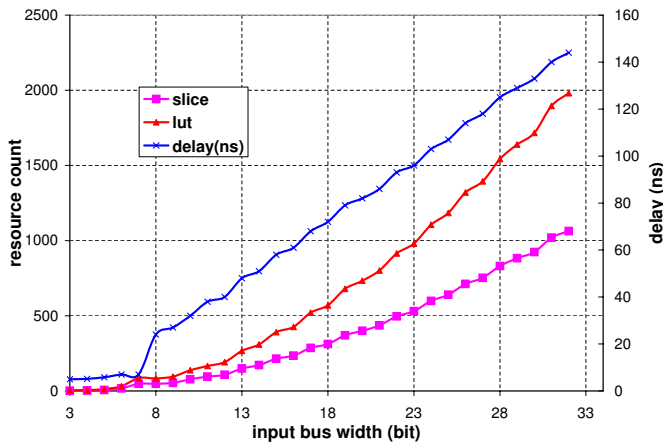
Figure IV illustrates a structure where an n-bit input bus is split into two $\frac{n}{2}$ -bit groups. Each group is encoded and decoded independently. The maximum delay of the encoder and decoder stages are $\tau_{enc}(n/2)$ and $\tau_{dec}(n/2)$, instead of $\tau_{enc}(n)$ and $\tau_{dec}(n)$.

The crosstalk that occurs across the boundary must be dealt with when the bus partitioning technique is employed. In Figure IV, we simply duplicate the boundary lines. This requires two extra wires for each added partition. Other approaches that can be applied to minimize the number of additional wires, such as group inversion as proposed in [5]. One additional wire has to be used as an inversion indication. The trade-off between area/speed and overhead can be balanced to achieve the required speed while minimizing the additional area overhead.

The FPF code is originally proposed to improve the bus speed. However, our simulations show that by applying the FPF encoding, the bus energy consumption can be reduced as well. We randomly generated 10,000 input vectors for 8, 12, 16 and 32 bit data and transmitted these randomly generated data on an uncoded bus and a coded bus respectively. For each bus width, a normalized total energy consumption, E_{norm} is computed based on Equation 5 with V_{dd} and C_L both set to 1. Table VI gives the normalized energy consumption for coded and uncoded busses. The comparison is technology independent and is also independent of bus configurations, i.e., bus length, wire sizing, provided that $\lambda \gg 1$ is guaranteed. The results indicate that even with $\sim 44\%$ more wires, coded busses have lower total energy consumption. It is important to point out that such saving is achieved using random sequences. For busses that transmit data with regularity, the results may vary. We also would like to point out that in our simulation, we do not include the power consumption of the CODEC.

V. CONCLUSIONS

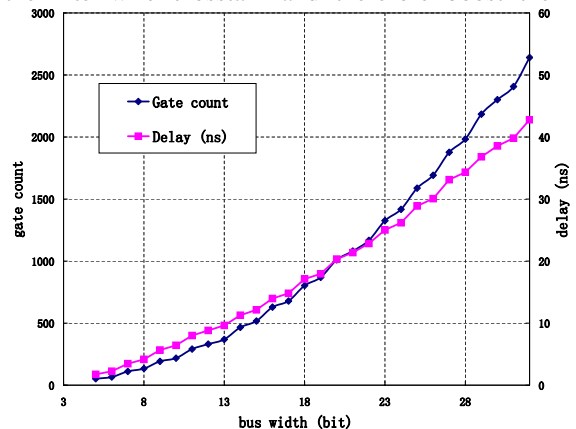Crosstalk avoidance codes are shown to be able to reduce the inter-wire crosstalk and therefore boost the maximum

Fig. 6.   Resource count and delay of the proposed encoder for different input bus sizes (FPGA implementation)

Fig. 7.   Gate count and delay of the proposed encoder for different input bus sizes (TSMC 90nm implementation)

bus size	E_{norm} **uncoded**	E_{norm} **coded**	saving
8	35075	27825	20.7 %
12	55004	44222	19.6 %
16	74999	61456	18.6 %
32	154191	124148	19.5 %

TABLE VI

POWER CONSUMPTION COMPARISON BETWEEN CODED AND UNCODED BUSSES

Fig. 9. Bus partition for delay/area improvement

speed on the data bus. They have the advantage of consuming less area overhead than shielding techniques. Even though several different types of codes have been proposed in the past few years, no mapping scheme was given which facilitates the CODEC implementation. Compounded by the non-linear nature of the CAC, the lack of a solution to the systematic construction of the CODEC has hampered the wide use of CAC in practice.

In this work, we give what we believe is the first solution to this problem. We showed that data can be coded to a forbidden pattern free vector in the Fibonacci numeral system. We first give a straightforward mapping algorithm that produces a set of FPF codes with near-optimal cardinality. The area overhead of this coding scheme is near the theoretical lower bound. The CODEC based on this coding scheme is systematic and has very low complexity. The size of the CODEC grows quadratically with the data bus size as opposed to exponentially in a brute forced implementation. Our systemic coding scheme allows the code design of arbitrarily large busses without having to resort to bus partitioning.

We further proposed an improved coding scheme which yield a set of FPF codes with maximum cardinality. The area overhead of this optimal coding scheme matches the theoretical lower bound. We gave the corresponding modification in the CODEC design as well.

This paper also discusses issues associated with CODEC implementations. We proposed a modified coding scheme that
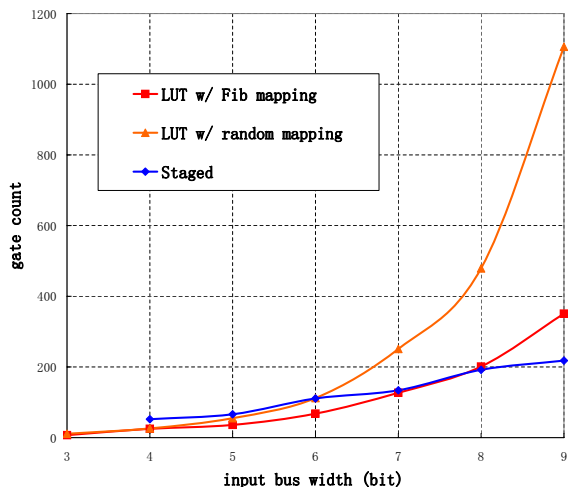

Fig. 8. Gate count comparison for different encoder implementations in a TSMC 90nm process: a single level Look-Up-Table (LUT) based implementation using Fibonacci mapping, a single level LUT using random mapping and the proposed staged encoder
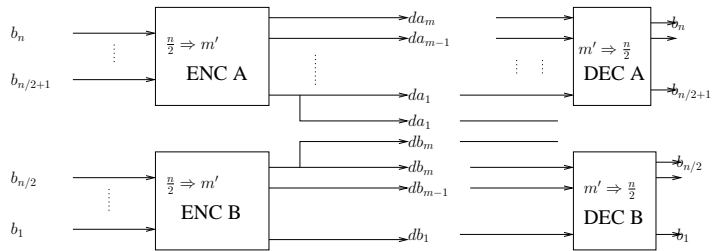
eliminates the MSB stage in the encoder and simplifies the decoder side as well. The modification reduces the total gate count and improves the CODEC speed.

We implemented the near-optimal CODEC in both an FPGA and a 90nm ASIC process. The reported results show that the size of our CODEC is several orders of magnitude lower than a previously reported design for a 12-bit bus. We also investigated the possibility of combining our approach with bus partitioning in very large busses to address the propagation delay issue as well as to reduce the total size of the CODEC.

We compared the average bus energy consumption of uncoded and FPF coded busses in simulation. Our experimental results show that FPF coding offers on average $\sim 20\%$ power saving.

Even though this work is strictly limited to one class of crosstalk avoidance code (the FPF-CAC), we believe that the approach can be easily adapted to other varieties of crosstalk avoidance codes as well.

REFERENCES

[1] P. Sotiriadis and A. Chandrakasan, "Low power bus coding techniques considering inter-wire capacitance". Proc. Of IEEE-CICC 2000, pp 507-510
[2] K. Kim, K. Baek, N. Shanbhag, C. Liu, and S.-M. Kang, Coupling driven signal encoding scheme for low-power interface design, Proc. of IEEE/ACM International Conference on Computer-Aided Design, Nov 2000.
[3] Madhu Mutyam, "Preventing Crosstalk Delay using Fibonacci Representation", Intl Conf. on VLSI Design, 2004, pp 685-688.
[4] S.R. Sridhara, A. Ahmed, and N. R. Shanbhag, "Area and Energy-Efficient Crosstalk Avoidance Codes for On-Chip busses", Proc. of ICCD, 2004, pp 12-17
[5] C. Duan, A. Tirumala and S. P. Khatri, "Analysis and Avoidance of Crosstalk in On-chip Bus", HotInterconnects, 2001,pp 133-138.
[6] Bret Victor and K. Keutzer,"Bus Encoding to Prevent Crosstalk Delay", ICCAD, 2001, pp 57-63.
[7] Sunil P. Khatri, "Cross-talk Noise Immune VLSI Design using Regular Layout Fabrics", PhD Thesis, UC Berkeley, 1999.
[8] C. Duan and S. P. Khatri, "Exploiting Crosstalk to Speed up On-chip busses", DATE 2004
[9] C. Duan, K. Gulati and S. P. Khatri, "Memory-based Cross-talk Canceling CODECs for On-chip busses", ISCAS 2006
[10] J. Ma and L. He, "Formulae and applications of interconnect estimation considering shield insertion and net ordering", ICCAD 2001, pp
[11] H. Kaul, D. Sylvester and D. Blauuw, "Active shielding of RLC global interconnects", Proc. of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems, 2002, pp 98-104
[12] Wikipedia,"Fibonacci number", http://en.wikipedia.org/wiki/Fibonacci_number
[13] Wikipedia, "Numeral System", http://en.wikipedia.org/wiki/Numeral_system
[14] Saraswat, Haghani and Bernard, "A low power design of Gray and T0 codecs for the address bus encoding for system level power optimization". www.studentimaster.usilu.net/ saraswap/prabhat/projects/

[15] Xilinx Virtex4 Family datasheet, "http://www.xilinx.com/ products/silicon_solutions/fpgas/virtex/virtex4/index.htm"

[16] TSMC 90 nm process, "http://www.tsmc.com/english/b_technology /b01_platform/b010101_90nm.htm"