MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Forbidden Transition Free Crosstalk Avoidance CODEC Design

Chunjie Duan, Chengyu Zhu, Sunil P. Khatri

TR2008-023 June 2008

Abstract

In this work, we present a CODEC design for the forbidden transition free crosstalk avoidance code. Our mapping and coding scheme is based on the Fibonacci numeral system and the mathematical analysis shows that all numbers can be represented by FTF vectors in the Fibonacci numeral system (FNS). The proposed CODEC design is highly efficient, modular and can be easily combined with a bus partitioning technique. We also investigate the implementation issues and our experimental results show that the proposed CODEC complexity is orders of magnitude better compared to the brute force implementation. Compared to the best existing approaches, we achieve a 17% improvement in logic complexity. A high speed design can be achieved through pipelining.

ACM/IEEE Design Automation Conference

Forbidden Transition Free Crosstalk Avoidance CODEC Design

Chunjie Duan
Mitsubishi Electric Research Labs
201 Broadway
Cambridge, MA 02052, USA
duan@merl.com

Chengyu Zhu
Polaris Microelectronics
3000 Long Dong Avenue
Pudong, Shanghai 201203, China
cy1zhu@yahoo.com

Sunil P. Khatri
Department of Electrical Engineering
Texas A&M University
College Station, TX 77843
USA
sunilkhatri@tamu.edu

ABSTRACT

In this work, we present a CODEC design for the forbidden transition free crosstalk avoidance code. Our mapping and coding scheme is based on the Fibonacci numeral system and the mathematical analysis shows that all numbers can be represented by FTF vectors in the Fibonacci numeral system (FNS). The proposed CODEC design is highly efficient, modular and can be easily combined with a bus partitioning technique. We also investigate the implementation issues and our experimental results show that the proposed CODEC complexity is orders of magnitude better compared to the brute force implementation. Compared to the best existing approaches, we achieve a 17% improvement in logic complexity. A high speed design can be achieved through pipelining.

Categories and Subject Descriptors

B.7 [**Hardware**]: Integrated Circuits

Keywords

crosstalk, on-chip bus, Fibonacci number, CODEC

1. INTRODUCTION AND PREVIOUS WORK

The performance of bus-based global interconnects has become a bottleneck to the overall system performance in Deep Sub-Micron (DSM) designs, and the trend is worsening with the advance of fabrication processes. Since the inter-wire coupling capacitance is significantly larger than the wire-to-substrate capacitance and the intrinsic load capacitance of the driver [8], crosstalk induced delay and power consumption have become a major determinant of the system performance [2, 6, 3]. Reducing crosstalk can greatly boost the system performance.

Among the many different crosstalk reduction schemes proposed [11, 6, 7, 5, 14, 3], some focus on reducing the energy consumption, some focus on minimizing the delay and others address both. Different data transition patterns can be classified based on the severity of the crosstalk they impose on the bus [6, 7]. Most of the crosstalk reduction techniques involve removing or lowering the probability of undesired patterns, and inevitably incur area overhead from the additional wires in the bus, additional circuitry or both. The efficiency of a crosstalk reduction scheme should be judged not only by the performance boost it brings about, but also by its area overhead as well. As an example, passive shielding requires a doubling of the number of wires, and hence incurs a 100% area overhead and therefore is not deemed efficient.

Bus encoding schemes can achieve the same amount of bus delay improvement as passive shielding, with a much lower area overhead [6, 7, 10, 5]. These codes are commonly referred to as *Crosstalk Avoidance Codes* (CACs). CACs can be memory-less [7, 6, 5] or memory-based [10]. Memory-based coding approaches generate a codeword based on the previously transmitted code and the current dataword to be transmitted [7, 10]. Although these type of codes need fewer additional bus wires, the CODEC complexity is generally considered too high for these coding schemes to be used in practice. The memory-less coding approaches, on the other hand, use a fixed code book to generate a codeword, solely based on the current input data. The CODECs for memory-less codes are projected to be simpler.

Several different types of memory-less CACs have been proposed [6, 7, 5]. All these codes offer the same degree of delay performance improvement. The area overhead caused by the additional wires ranges from 44% to 68%, which is much better than passive shielding. Two of the most efficient memory-less codes are *forbidden-pattern-free* (FPF) CACs [1] and *forbidden-transition-free* (FTF) CACs. Their overhead performance is near identical, and both methods approach the theoretical lower bound. The FPF is slightly better, but by no more than one wire.

Unfortunately, efficient CODEC designs are not available for either of these codes. Due to the non-linear nature of these codes, researchers have struggled to find a mapping scheme that is mathematically systematic and efficient to implement. Attempts through brute force logic optimization have shown that the CODEC gate count becomes prohibitively large for a bus of reasonable size, since its complexity grows exponentially with the bus size [5, 10].

We have recently proposed several systematic mapping

DAC 2008, June 8–13, 2008, Anaheim, California, USA.

schemes in [1] for FPF-CACs. The proposed mapping schemes enable highly efficient, highly structured CODEC implementations. Our analysis and experimental results show that the CODEC complexity grows quadratically with bus width instead of exponentially. We can implement an encoder for a 32 bit data bus with less than 2700 gates (2-input equivalent) as opposed to over $20K$ gates for a 12-bit bus reported previously [1, 5].

In this work, we focus on the CODEC design for the FTF-CAC. We give the mathematical analysis of the mapping scheme and the coding algorithm based on the mapping scheme. We also investigate implementation issues and compare the complexity of the CODEC with the CODECs for FPF-CACs. The comparison shows that the FTF-CAC CODEC is simpler and faster. Circuit complexity is reduced by more than 17% compared to the FPF-CAC CODECs. The proposed CODEC is also easier to use in conjunction with bus partitioning.

The key contributions of this paper are:

- The presentation of a mathematical formula based on the Fibonacci numeral system (FNS), which introduces an elegant and efficient mapping between datawords and codewords.
- CODEC design for FTF-CACs, based on the FNS frameworks, which scales efficiently with bus size, and has exponentially less complex circuit realization than existing brute force implementations.

The remainder of this paper is organized as follows: In Section 2 we give the delay and power models for an on-chip bus in the presence of crosstalk, show the relation between crosstalk and data pattern, and discuss the properties of FPF-CACs and FTF-CACs. Section 3 provides the mathematical basis for the mapping scheme and the coding algorithm for FTF-CACs. In Section 4, we investigate the circuit implementation details of the proposed CODEC and present our experimental results. Section 5 summarizes the paper.

2. BUS MODELS AND CROSSTALK AVOIDANCE

2.1 Bus Delay Model

Crosstalk in an on-chip bus has been shown to be dependent on the data patterns on the bus and has a significant impact on the signal delay as well as the overall energy consumption of the bus [9, 2]. Assuming a bus with load capacitance C_L and inter-wire capacitance C_I, The delay τ_j of the j^{th} wire in a data bus is given as [2]

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

where k is a constant determined by the driver strength and wire resistance, V_{dd} is the supply voltage, $\delta_j \in \{0, 1\}$ is the normalized voltage change on j^{th} line. $\delta_{j,j\pm1} \in \{0, \pm1, \pm2\}$ 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, while $\delta_{j,j-1}$ and $\delta_{j,j+1}$ correspond to the crosstalk induced delay. $\lambda = C_I/C_L$ and in a DSM process, $\lambda \gg 1$ is satisfied. Therefore the δ_j term has a negligible contribution to the delay [8].

We define the *effective total capacitance* of the driver of j^{th} line, $C_{eff,j}$ as

$$C_{eff,j} = C_L \cdot abs(\delta_j + \lambda \cdot \delta_{j,j-1} + \lambda \cdot \delta_{j,j+1}) \quad (2)$$

and rewrite Equation (1) as

$$\tau_j = k \cdot V_{dd} \cdot C_{eff,j} \quad (3)$$

Table 1 lists the values of C_{eff} for some transition patterns. We can see that the value of C_{eff} varies from C_L to $(1 + 4 \cdot \lambda)C_L$. Crosstalk patterns corresponding to the different $C_{eff,j}$ values are classified as *0C, 1C, 2C, 3C* and *4C* patterns. Since the speed of the data bus is determined by $max\{C_{eff,j}\}$ over *all* the bits in the bus, we observe that by eliminating 4C crosstalk on ALL lines in the bus, we can increase the maximum speed of the bus by $\sim 33\%$. If 3C AND 4C crosstalk can be eliminated on all lines, we can speed up the bus by $\sim 100\%$.

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

Table 1: Classes of crosstalk

2.2 Forbidden pattern free CACs

Forbidden patterns are defined as the two 3-bit patterns "010" and "101". A *forbidden pattern free* code is a set of codewords which do not contain forbidden patterns on any 3 adjacent bus bits. For example, "100110" is FPF while "10111" is not an FPF code. By eliminating the forbidden patterns in the codewords, it is guaranteed that C_{eff} for any bit in the bus does not exceed $(1 + 2\lambda)C_L$ [6] and hence the maximum delay is reduced by 50% compared to an uncoded bus.

The maximum cardinality of FPF codewords is $2f_{m+1}$ [6], where f_m is the m^{th} element in the Fibonacci sequence defined as

$$f_m = \begin{cases} 0 & \text{if m} = 0, \\ 1 & \text{if m} = 1, \\ f_{m-1} + f_{m-2} & \text{if m} \geq 2. \end{cases} \quad (4)$$

The asymptotic area overhead in terms of additional wires is calculated to be $\sim 44\%$ for a binary bus using FPF-CACs.

2.3 Forbidden transition free CACs

The *forbidden transition* is defined as the simultaneous transition, in opposite directions, on any two adjacent wires in a bus. A code is **forbidden transition free** (FTF) if transitions between codewords do not generate forbidden transitions on any adjacent bits of the bus. This type of code was first investigated in [7].

Similar to FPF codes, FTF codes can be generated by eliminating certain patterns. Not to be confused with the 3-bit forbidden patterns, we refer to these patterns as *prohibited pattern*. The prohibited pattern is either a "01" or "10" on two adjacent bus bits.

The possible data patterns on two wires in a bus are "00", "01", "10" and "11". It is easy to see that the elimination of either "01" or "10" on two adjacent bits will cause the pair to be forbidden transition free. It has been proven in [7] that having alternating prohibited patterns on bits d_{2k}, d_{2k-1} and d_{2k+1}, d_{2k} yields a set with the maximum number of codewords (cardinality).

Conversely, by prohibiting "10" on $d_{2k}d_{2k-1}$ and "01" on $d_{2k+1}d_{2k}$, we can produce a different set of FTF-CACs.

The maximum cardinality of FTF-CACs is f_{m+2}, slightly lower than the cardinality of FPF-CACs. When the bus size is large, the area overhead of FTF-CACs over an uncoded bus approaches 44% as well. Table 2 lists the codewords of one set of the 2, 3,4 and 5 bit FTF-CACs.

2-bit	3-bits	4-bits	5 bit	
00	000	0000	00000	10100
01	001	0001	00001	10101
11	100	0100	00100	10111
	101	0101	00101	11100
	111	0111	00111	11101
		1100	10000	11111
		1101	10001	
		1111		

Table 2: FPF-CAC codewords for 2,3,4 and 5 bit busses

Recently, we have showed that there exists a deterministic mathematical mapping for FPF-CACs using FNS, and proposed two different coding algorithms as well as the corresponding CODEC implementations [1]. In this paper, we present the mathematical framework for FTF-CAC design using FNS, along with an algorithm for its CODEC.

3. AN EFFICIENT FTF-CAC CODEC DESIGN

Both the FTF-CACs and FPF-CACs were proposed earlier and algorithms were given to generate codewords for them [7, 6]. However, the CODEC design was not thoroughly addressed in the original papers. Since then, there have been more research results published on designing CODECs for these CACs. Most of the designs were based on bus partitioning techniques, which require additional wires on the bus. More importantly, none of them addressed the fundamental issue of how to map datawords to codewords. This is partially due to the fact that the CACs are non-linear codes, and it is difficult to find a mapping using conventional mathematical expressions. [5] showed that a brute force look-up-table implementation is impractical, as the CODEC size grows exponentially with the bus size. Figure 1 (obtained from [5]) shows this exponential growth with increasing bus size.

The Fibonacci numeral system (FNS) was first linked to the CACs in [4]. However, it was only used for recursive codeword searching. In this section, we discuss the mapping for FTF-CACs, as well as the coding algorithm.

3.1 Fibonacci-based numeral system

A **numeral system** is *"a framework where numbers are represented by numerals in a consistent manner"* [13]. Commonly used numeral systems includes decimal, binary, hexadecimal. These systems differ in their basis. The Fibonacci-based numeral system (FNS) $N(F_m, \{0,1\})$ is the numeral system that uses Fibonacci sequence as the base. In FNS, a number v is represented as the summation of some Fibonacci numbers

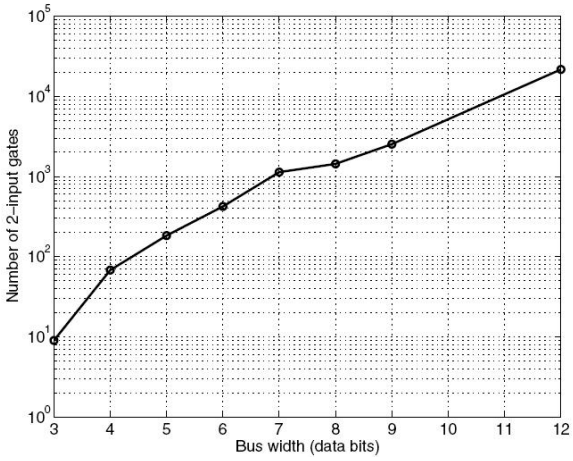

Figure 1: Encoder size with a brute-force implementation

$$v = \sum_{k=1}^{m} d_k \cdot f_k, \tag{5}$$

where $d_k \in \{0,1\}$ and f_k is the k^{th} Fibonacci number defined in Equation (4).

The Fibonacci-based numeral system is *complete* but *ambiguous*, therefore any number has at least one, but possibly more than one representation in FNS. For example, a decimal number 19 can be represented by any of the following *six* 7-digit vectors in FNS: {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* and a vector in the Fibonacci numeral system as a *Fibonacci vector* or *Fibonacci code*. All the Fibonacci vectors that represent the same value are defined as *equivalent vectors*.

We give a very important identity of the Fibonacci sequence here. It is used in the following discussions.

$$f_m = \sum_{k=0}^{m-2} f_k + 1 \tag{6}$$

From Equation (6), we see that the range of a m-bit Fibonacci vector is $[0, f_{m+2})$, where the 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.

3.2 Mapping Scheme

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 $d_m d_{m-1}...d_2 d_1$ is forbidden transition free, $\forall v \in [0, f_{m+2}-1]$.*

Theorem 1 states that for any number $v \in [0, f_{m+2})$, there exists at least one m-bit Fibonacci vector $d_m d_{m-1}...d_2 d_1 =$

v which represents this number and satisfies the *forbidden transition free* property. In other words, a number of any value in the range of $[0, f_{m+2})$ can be mapped to an m bit FTF codeword in the FNS space.

Proof: Let us first define V_{00} as the set of all the k-bit Fibonacci vectors with the two most significant bits (MSBs) of "00", we have $V_{00} \in [0, f_k)$ based on Equation (6). Similarly, we define sets V_{01}, V_{10} and V_{11}. $V_{01} \in [f_{k-1}, f_{k+1})$, $V_{10} \in [f_k, 2f_k)$ and $V_{11} \in [f_{k+1}, f_{k+2})$. This is also shown in Figure 2. We can see from Figure 2 that the ranges of these four sets of vectors overlaps. For any vector $v \in V_{01}$, we can find an equivalent vector in V_{00} or V_{10}. Similarly, any vector $v \in V_{10}$ can be replaced by an equivalent vector in V_{01} or V_{11}.

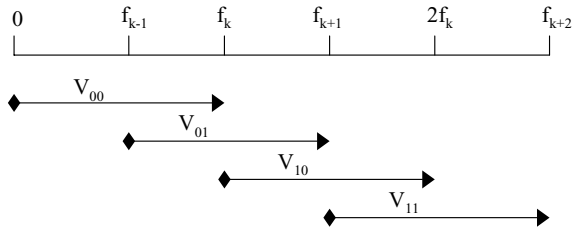

Figure 2: Range of the vectors with MSBs of "00", "01", "10" and "11"

Now for a given number v, we can find an m bit vector $V_{in} = d_m d_{m-1}...d_2 d_1$ in FNS that represents this value (based on the completeness of FNS). Now assuming "01" to be the prohibited pattern at $d_k d_{k-1}$, we can replace $d_k d_{k-1}...d_1$ with an equivalent k bit vector in V_{00} or V_{10} to produce a new m bit vector V_{eq} that is equivalent to V_{in}. V_{eq} has no prohibited pattern at $d_k d_{k-1}$.

Also since "01" is the prohibited pattern at $d_k d_{k-1}$, the prohibited pattern at $d_{k+1} d_k$ and $d_{k-1} d_{k-2}$ is "10". We know that

- If V_{in} does not have the prohibited pattern, V_{eq} does not have the prohibited pattern either since the manipulation cause the k^{th} bit value to either change from '0' to '1' or no change.

- V_{eq} does not have the prohibited pattern at $d_{k-1} d_{k-2}$ since d_{k-1} in V_{eq} is '0'.

The first condition guarantees that the manipulation do not introduce a prohibited pattern into $d_m..d_k$. The second condition guarantees that further manipulation does not destroy the pattern in $d_k d_{k-1}$.

Therefore, by replacing the prohibited pattern recursively from the MSB to the LSB, the final result is an equivalent vector to V_{in} that is prohibited pattern free and therefore it is a FTF codeword. This proves Theorem 1

3.3 Coding algorithm

To design a coding algorithm that maps each input value to an FTF codewords in the FNS, we observe that in Figure 2, if d_k is coded as

$$d_k = \begin{cases} 0 & \text{if } v < f_k, \\ 1 & \text{otherwise} \end{cases}$$

and d_{k-1} is coded as

$$d_{k-1} = \begin{cases} 0 & \text{if } v < f_{k+1}, \\ 1 & \text{otherwise} \end{cases}$$

the "01" is eliminated on $d_k d_{k-1}$. Conversely, if d_k, is coded as

$$d_k = \begin{cases} 0 & \text{if } v < f_{k+1}, \\ 1 & \text{otherwise} \end{cases}$$

and d_{k-1} coded as

$$d_{k-1} = \begin{cases} 0 & \text{if } v < f_{k-1}, \\ 1 & \text{otherwise} \end{cases}$$

"10" is eliminated on $d_k d_{k-1}$.

Based on the above observations, the coding algorithm is straightforward. Algorithm 1 generates FTF codewords with "01" prohibited at $d_{2k+1} d_{2k}$ and "10" prohibited at $d_{2k} d_{2k-1}$.

Algorithm 1 FTF encoding algorithm

```
Input: v;
r_{m+1} = v;
for k = m downto 2 do
  if r_{k+1} < f_{2⌊k/2⌋+1} then
    d_k = 0;
  else
    d_k = 1;
  end if
  r_k = r_{k+1} − f_k · d_k;
end for
d_1 = r_2;
Output: d_m d_{m−1} ... d_1;

⌊·⌋: floor operator.
```

The encoding is carried out sequentially in $m-1$ stages, similar to a division operation. Starting from the MSB, each stage compares the input to a Fibonacci number and produces a coded bit as well as a remainder. The remainder from one stage becomes the input to the next stage.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

A direct implementation of the encoder and decoder based on Algorithm 1 and Equation (5) is illustrated in Figure 3 (assuming m is even). The encoder, which converts a n bit binary vector $v = b_n..b_1$ to an m bit vector $d_m d_{m-1}..d_1$, consists of $m-1$ stages. The decoder converts the bus codewords back to the original n bit binary vector.

For the decoder, even though the block diagram in Figure 3 shows multipliers, the actually implementation does not require multiplication or AND operations. Instead, the operation of one bit input d_j multiplied by a constant f_j can be implemented by simply driving the input bit to the non-zero bit positions of f_j. The m inputs of the summation block have different numbers of bits with the MSB bit having the most number of bits (f_m) and two LSB inputs having 1 bit each. The overall complexity is roughly $O(m^2/2)$.

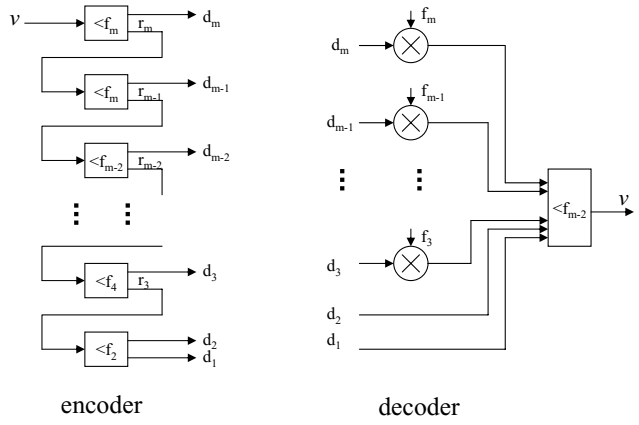

Figure 3: Encoder and decoder block diagram

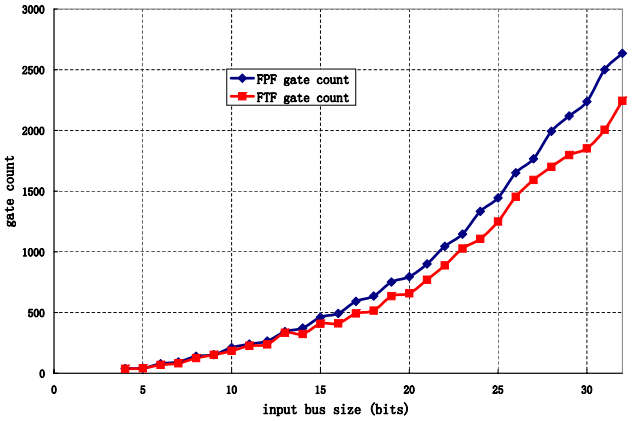

Figure 4: Encoder gate count comparison

$$d_k = \begin{cases} 0 & \text{if } r_{k+1} < f_{2\lfloor \frac{k}{2} \rfloor + 1}, \\ 1 & \text{otherwise} \end{cases}$$
$$r_k = r_{k+1} - f_k \cdot d_k \qquad (7)$$

The encoder stages implement the arithmetic function given in Equation (7). Each stage produces a single bit d_k and a remainder r_k. For the MSB stage, the input r_{k+1} is the input data to be encoded. For other stages, r_{k+1} is the remainder of the preceding stage. If m is even, the implementation requires *one* comparator, *one* adder and *one* 2:1 MUX. If m is odd, the implementation only needs one adder and one MUX since the adder (subtraction) also functions as a comparator. Assuming the resources required for a comparator are same as an adder, the total resources required for an m bit bus encoder is $\frac{3m}{2}$ adders and m MUXes. The number of bits reduces monotonically from the MSB stage to the LSB stage.

If we combine every two stages in the encoder properly, the logic can be further simplified. Let "10" be the prohibited pattern for $d_k d_{k-1}$. A two-bit encoder stage implements the following:

$$d_k d_{k-1} = \begin{cases} 00 & \text{if } r_{k+1} < f_k, \\ 01 & \text{if } r_{k+1} < f_{k+1}, \\ 11 & \text{otherwise} \end{cases}$$
$$r_{k-1} = \begin{cases} r_{k+1} & \text{if } r_{k+1} < f_k, \\ r_{k+1} - f_k & \text{if } r_{k+1} < f_{k+1}, \\ r_{k+1} - f_{k+1} & \text{otherwise} \end{cases} \qquad (8)$$

Equation (8) can be implemented using *two* adders and *two* MUXes. Since the single-bit stage implementation requires *three* adders and *two* MUXes to encode two bits, this two-bit stage implementation is simpler. We should point out that this simplification is only achieved when "10" is the prohibited pattern. We can not reduce the implementation complexity if "01" is the prohibited pattern in the two bit implementation. For comparison, the FPF CODEC proposed in [1] need one adder, one comparator and one MUX for each stage, almost twice the complexity of the FTF CODEC with two-bit implementation. The decoders are identical for both FTF and FPF.

To evaluate the complexity of the CODECs, we synthesized the CODEC in a 90nm process [15]. Figure 4 plots the equivalent gate counts of the encoder for input bus widths from 4 to 32. For 12-bit input data width, the equivalent number of 2-input gates is 245. This is nearly two orders of magnitude lower than the gate count reported in Figure 1. For the input data width of 32-bit, the equivalent gate count is 2247. The growth of the encoder sizes is quadratic with respect to the bus size, as we expected.

For comparison, the gate counts of the FPF-CAC encoders obtained through synthesis [1] are also plotted in Figure 4. The FPF-CAC encoder gate count for a 32-bit bus is 2640, which is 17.6% bigger than the FTF-CAC encoder for the same bus size. On average, the gate count for the FPF encoder is $\sim$ 17% higher that the FTF-CAC encoder. The decoders for both codes are identical.

Without pipelining, the overall delay of the encoder is the summation of all the stage delays. This total delay can be be significant. Fortunately, our design allows pipeline stages to be easily inserted between stages. The speed of the encoder is determined by the slowest stage, the MSB stage. It is reported in [16] that a 64-bit adder has a total delay of less than 250ps using a 65nm process. With additional delay from the MUX, we estimate that the slowest stage of our CODEC, the MSB stage, has a delay of no more than 300ps. Therefore, we expect a pipelined implementation of the CODEC to operated at 3 GHz. It is very important to point out that pipelining cannot be used on a look-up-table based design.

The complexity and speed can be further improved by applying bus partitioning. The maximum input-to-output delay of an non-pipelined m-bit encoder is $\tau(m) \propto O(m^2)$. The total area has the quadratic relation with the number of input bits and therefore partitioning the bus will reduce the total area by close to 50%. Unlike FPF CODECs, which require either two shield wires between the group boundaries or some group complement logic for bus partitioning, the FTF code only needs one grounded wire between two groups.

5. CONCLUSIONS

In this work, we present a CODEC design for the FTF-CAC based on the Fibonacci numeral system. Our analysis show that all numbers can be represented by FTF vectors in

Fibonacci numeral system. We propose encoder and decoder designs which have the following features:

- Efficiency: both the encoder and decoder complexities grow quadratically as opposed to exponentially. Compared to the CODEC for the FPF-CAC [1], the arithmetic operations are less complicated, results in further complexity reduction in implementation.
- Modularity: both the encoder and decoder are constructed in a systematic fashion. The encoder consists of multiple stages and a CODEC design for a larger bus can be extended from a CODEC of a smaller bus.
- Simplicity: bus partitioning becomes trivial and incurs less overhead compared to the FPF-CAC CODECs.

Our CODEC design has been verified through actual implementation. Encoders for bus size from 4 to 32 bit are implemented in a 90 nm CMOS process and the results show a 17% reduction in gate count over the FPF-CAC encoder for the same bus size [1]. The design can achieve high speed through pipelining.

6. REFERENCES

[1] C. Duan, V. Cordero and S. P. Khatri, "Efficient On-Chip Crosstalk Avoidance CODEC Design", IEEE Transactions on VLSI Systems, to appear.

[2] P. Sotiriadis and A. Chandrakasan, "Low power bus coding techniques considering inter-wire capacitance". Proc. of IEEE-CICC 2000, pp 507-510.

[3] K. Kim, K. Baek, N. Shanbhag, C. Liu, and S.-M. Kang, ŞCoupling driven signal encoding scheme for low-power interface design,T̃ Proc. of IEEE/ACM International Conference on Computer-Aided Design, Nov 2000.

[4] Madhu Mutyam, "Preventing Crosstalk Delay using Fibonacci Representation", Intl Conf. on VLSI Design, 2004, pp 685-688.

[5] 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.

[6] C. Duan, A.Tirumala and S.P.Khatri, "Analysis and Avoidance of Cross-talk in On-chip Bus", HotInterconnects, 2001,pp 133-138.

[7] Bret Victor and K. Keutzer,"Bus Encoding to Prevent Crosstalk Delay", ICCAD, 2001, pp 57-63.

[8] Sunil P. Khatri, "Cross-talk Noise Immune VLSI Design using Regular Layout Fabrics", PhD Thesis, UC Berkeley, 1999.

[9] C. Duan and S. P. Khatri, "Exploiting Crosstalk to Speed up On-chip busses", DATE 2004, pp 778-783.

[10] C. Duan, K. Gulati and S. P. Khatri, "Memory-based Cross-talk Canceling CODECs for On-chip busses", ISCAS 2006, pp 4-9.

[11] J. Ma and L. He, "Formulae and applications of interconnect estimation considering shield insertion and net ordering", ICCAD 2001, pp

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

[16] A. Bastani and C. Zukowski, "A Low-Leakage High-Speed Monotonic Static CMOS 64b Adder in a Dual Gate Oxide 65-nm CMOS Technology", Proc. of ISQED 2006, pp312-317.