Memory-based Cross-talk Canceling CODECs for On-chip Buses

Chunjie Duan* Kanupriya Gulati† Sunil P. Khatri†
* Mitsubishi Electric Research Laboratories, Cambridge, MA 02139
† Department of EE, Texas A&M University, College Station TX 77843

Abstract

In recent times, the ratio of the cross-coupling capacitance between adjacent on-chip wires on the same metal layer to the total capacitance of any wire is becoming quite large. As a consequence, signal wires exhibit a significant delay variation and noise immunity problems. This problem is aggravated for long on-chip buses. In this paper, we develop memory-based crosstalk canceling CODECs for on-chip buses. We describe an Reduced Ordered Binary Decision Diagram (ROBDD) based methodology to accurately compute the bus area overhead of the CODECs. We report the asymptotic overhead for CODECs which cancel three kinds of crosstalk patterns, and demonstrate that the bus size overheads are lower than the corresponding overheads for a memoryless CODEC. This results in a reduced overall area utilization for memory-based cross-talk canceling CODECs, compared to their memoryless counterparts. We also demonstrate that the use of these cross-talk canceling CODECs enables a user to speed up a bus by a factor of over 6×. Further, by using our techniques, a user may trade off the speed gain against the attendant bus size overhead.

1. Introduction

Crosstalk has become a significant problem in deep sub-micron (DSM) VLSI design [8]. The aggressive scaling of processes that lies at the heart of the relentless drive towards smaller and faster ICs results in an increase in wiring delays due to increasing wire sheet resistivities. To reduce this effect, recent processes have scaled wires only in the horizontal dimension, effectively creating 'tall' wires. As a consequence, the cross-coupling capacitance (C_x) between two minimally spaced adjacent wires on the same metal layer is much greater than the substrate capacitance (C_{sub}) of any of the wires.

If the ratio $r = \frac{C_x}{C_{sub}}$ is large, crosstalk between adjacent wires on the same metal layers manifests in ways that make designs unpredictable. In particular, it results in a significant delay variation in a wire, depending on the state of its neighbors. Also, it can result in possible signal integrity problems, since a static wire can suffer a glitch caused by capacitively coupled voltages from its switching neighbors. Crosstalk has therefore become a critical design issue in modern IC design.

Consider three adjacent wires in an on-chip bus, which are driven by signals b_{i-1}, b_i and b_{i+1}. The total effective (switched) capacitance of driver b_i is dependent on the state of b_{i-1} and b_{i+1}. In the best case[1], the total effective capacitance of b_i is $C_{min} = C_{sub}$, and in the worst case[2], the effective capacitance is $C_{max} = 4 \cdot C_x + C_{sub}$. With $r \gg 1$, we observe that $\frac{C_{max}}{C_{min}} \gg 1$, and hence the delay of bus signals strongly depends on the data pattern being transmitted on the bus. As a result of this large delay variation, the worst case delay of a signal in an on-chip bus is also increased, limiting system performance. The problems due to crosstalk are aggravated in long on-chip buses, since bus signals are longer and therefore more capacitive, resulting in larger worst case delays[3]. Therefore, special consideration must be given to crosstalk immunity for such signals. The focus of this paper is to develop memory based encoding techniques to alleviate crosstalk in on-chip buses. Our encoding approaches allow for the selective reduction of the crosstalk effects in on-chip buses. By providing immunity from crosstalk in buses, our encoding based techniques reduce the crosstalk induced delay variation effect in on-chip buses. This has the important benefit of reducing the maximum delay as well as reducing signal integrity problems in the bus signals. The bus size overheads of our techniques are high when a greater crosstalk immunity is desired. This allows the designer to effectively trade off the degree of crosstalk control desired with the bus size overhead. In the sequel, we refer to bus overhead as the additional number of bits required in order to encode a bus in a cross-talk free manner.

The most aggressive of our encoding techniques *actually speeds up a bus by exploiting crosstalk*. In this encoding approach, if a bus signal rises (falls), then our encoding forces one of its neighbors to rise (fall) as well, while the other neighbor is static. As a result, we actually improve the risetime of the wire by utilizing crosstalk to our benefit. This is *not possible with current approaches* to alleviate the cross-talk problem in buses (which stagger bus signals in space and/or time to mitigate the cross-talk problem among bus signals).

In recent times, with wiring delays increasing compared to gate delays [8], it is often the case that the critical delay in a circuit is determined by long buses. In such a case, buses could be encoded with the techniques described in this paper, allowing the design to be operated at a much greater frequency. A designer would gladly tolerate the bus size overhead involved with the use of our approach, in such a scenario.

This paper is organized as follows. Section 2 provides definitions used in the rest of the paper. This section also provides a classification (similar to that of [5]) of bus data patterns based on the maximum amount of crosstalk incurred by such patterns. In Section 3, we discuss previously published approaches for solving this problem. In Section 4, we describe our approach of creating memory-based crosstalk canceling CODECs. In Section 5 we report the results of experiments that we have performed to quantify the tradeoff between the degree of crosstalk immunity achieved by the above techniques, and the bus size overhead incurred. We compare our bus size overheads with those reported in [5, 4], in which memoryless CODECs to eliminate $4 \cdot C$, $3 \cdot C$ and $2 \cdot C$ crosstalk patterns were described. We conclude the paper in Section 6.

2. Preliminaries

In this section, we introduce the classification scheme for bus data transitions which we will utilize in the sequel. Our classification is largely borrowed from that introduced in [5].

Consider an n-bit bus, consisting of signals $b_1, b_2, b_3 \cdots b_{n-1}, b_n$.

DEFINITION 1. *: A* **Vector** *v is an assignment to the signals b_i as follows:*
$b_i = v_i$, (where $1 \leq i \leq n$ and $v_i \in \{0,1\}$).

Consider two successive vectors v^j and v^{j+1}, being transmitted on a bus. For vector v^j, assume $b_i = v_i^j$ (where $1 \leq i \leq n$ and $v_i^j \in \{0,1\}$). Similarly, for vector v^{j+1}, assume $b_i = v_i^{j+1}$ (where $1 \leq i \leq n$ and $v_i^{j+1} \in \{0,1\}$).

Consider a vector sequence $v^1, v^2, \cdots, v^j, v^{j+1}, \cdots v^k$, applied on a bus. This sequence consists of k n-bit vectors. We define five types of crosstalk conditions below. For these definitions, we assume that $0 \leq i \leq n-2$ and $0 \leq j \leq k-1$.

DEFINITION 2. *A sequence of vectors is called a* **4·C sequence** *if $\exists i, j$ s.t.*
$v_i^j = v_{i+1}^{j+1} = v_{i+2}^j = v$ and $v_i^{j+1} = v_{i+1}^j = v_{i+2}^{j+1} = \overline{v}$, where $v \in \{0,1\}$.

DEFINITION 3. *A sequence of vectors is called a* **3·C sequence** *if it is not a 4·C sequence and $\exists i, j$ s.t.*

[1]In the best case, b_{i-1}, b_i, b_{i+1} all simultaneously transition in the same direction.
[2]In the worst case, b_{i-1} and b_{i+1} simultaneously transition in the opposite direction as b_i.
[3]Although current designs attempt to reduce the impact of this worst case delay by staggering bus signals in space and/or time, they are unable to *speed up* the bus by exploiting capacitive cross-talk among wires, which is an important feature of our approach

- $v_i^j = v_{i+1}^{j+1} = v_1$ *and* $v_i^{j+1} = v_{i+1}^j = \overline{v_1}$ *and* $v_{i+2}^j = v_{i+2}^{j+1} = v_2$ *where* $v_1, v_2 \in \{0,1\}$ *OR*
- $v_{i+1}^j = v_{i+2}^{j+1} = v_1$ *and* $v_{i+1}^{j+1} = v_{i+2}^j = \overline{v_1}$ *and* $v_i^j = v_i^{j+1} = v_2$ *where* $v_1, v_2 \in \{0,1\}$.

DEFINITION 4. *A sequence of vectors is called a* **2·C sequence** *if it is not a 4·C or 3·C sequence and* $\exists i, j$ *s.t.*
$v_i^j = v_i^{j+1} = v_1$ *and* $v_{i+1}^j = v_2$ *and* $v_{i+1}^{j+1} = \overline{v_2}$ *and* $v_{i+2}^j = v_{i+2}^{j+1} = v_3$, *where* $v_1, v_2, v_3 \in \{0,1\}$.

DEFINITION 5. *A sequence of vectors is called a* **1·C sequence** *if it is not a 4·C, 3·C or 2·C sequence and* $\exists i, j$ *s.t.*

- $v_i^j = v_i^{j+1} = v_1$ *and* $v_{i+1}^j = v_{i+2}^j = v_2$ *and* $v_{i+1}^{j+1} = v_{i+2}^{j+1} = \overline{v_2}$, *where* $v_1, v_2, v_3 \in \{0,1\}$ *OR*
- $v_{i+2}^j = v_{i+2}^{j+1} = v_1$ *and* $v_i^j = v_{i+1}^j = v_2$ *and* $v_i^{j+1} = v_{i+1}^{j+1} = \overline{v_2}$, *where* $v_1, v_2, v_3 \in \{0,1\}$.

DEFINITION 6. *A sequence of vectors is called a* **0·C sequence** *if it is not a 4·C, 3·C, 2·C, or 1·C sequence.*

DEFINITION 7. *A* $p \cdot C$ **crosstalk canceling CODEC** *(or* $p \cdot C$ *crosstalk free CODEC) transforms an arbitrary m-bit vector sequence into a n-bit vector sequence* ($m < n$) *such that the output vector sequence is a* $(p-1) \cdot C$ *sequence.*

DEFINITION 8. *A set* C_n *of n-bit vectors is said to be a* $p \cdot C$ **crosstalk free clique** *iff any vector sequence* $v_1 \rightarrow v_2$ *made up of vectors* $v_1, v_2 \in C_n$ *is a* $l \cdot C$ *sequence (where* $l < p$*), and there exists* $v_1^*, v_2^* \in C_n$ *such that* $v_1^* \rightarrow v_2^*$ *is a* $(p-1) \cdot C$ *sequence.*

If a sequence of vectors on a bus is a p·C sequence ($0 \leq p \leq 4$), then the physical interpretation of this is that:

- This vector sequence has at least one bit b for which there exists consecutive vectors that require the driver of this bit to charge a capacitance $p \cdot C_x + C_{sub}$. Note that $C_x \gg C_{sub}$.
- For this sequence, there does not exist any bit such that the driver of this bit is required to charge a capacitance greater than $p \cdot C_x + C_{sub}$.

A **memoryless** CODEC simply encodes an m bit vector with a unique n bit vector. A **memory-based** CODEC encodes an m bit vector with an n bit vector. The encoding depends on the k previous n bit vectors that were transmitted on the bus (for a memory depth k).

Note that in the sequel, *if we say that a CODEC is* $kC - free$*, we mean that it results in cross-talk of magnitude* $(k-1)C$ *or less, for any bus transition.*

3. Previous Work

Crosstalk reduction for on-chip buses has been the focus of some recent research. In [15], the main contribution of the authors was to extend the Elmore delay model to account for distributed nature of self and cross-coupling capacitances in on-chip buses. They suggest the possibility of using CODECs to eliminate certain bus transitions. They also suggest that encoding could speed up buses by 2× (this would be achieved by ensuring that bus never exhibits $4 \cdot C$ or $3 \cdot C$ transitions). In [5], the authors classify bus data transitions from a crosstalk viewpoint, and describe memoryless CODECs to eliminate $4 \cdot C$ and $3 \cdot C$ transitions on the bus. They show that the asymptotic overhead when eliminating $3 \cdot C$ transitions is about 44%. In [4], the authors describe $2 \cdot C$ and $1 \cdot C$ cross-talk canceling memoryless CODECs. The CODECs described in [5, 4] are memoryless. The authors of [16] discuss memory based as well as memoryless encoding techniques to eliminate crosstalk. However, area and delay overheads due to CODEC implementation were not quantified. Further, the algorithm to determine the bus overhead required an *explicit enumeration* of all 2^{2n} vector transitions. In contrast, we employ implicit enumeration, resulting in a more compact representation and therefore a more efficient computation. In [7], the authors reduce crosstalk induced delay variation in buses by selectively skewing bus data signals. Finally, [11] proposes a bus repeater sizing methodology which accounts for crosstalk induced delays and controls them by upsizing the drivers. This could result in driver circuits with large power and area requirements.

In [12], the authors describe a technique to simultaneously minimize bus power consumption while eliminating $3 \cdot C$ and $4 \cdot C$ crosstalk. The possibility of eliminating $1 \cdot C$ and $2 \cdot C$ crosstalk is not discussed. Further, the overhead for CODEC implementation is not discussed. The overhead in terms of bus size, of their $3 \cdot C$ crosstalk eliminating CODEC is between 62.5% and 72% (depending on bus size), in contrast to the asymptotic overhead of 44% reported in [5] and [16]. The work of [13] focuses on bus energy as opposed to delay. No CODECs are utilized, rather the approach is to adjust the spacing between bus wires non-uniformly (based on specific bus data statistics), with the ultimate goal of reducing bus energy. However, the worst case bus bit still incurs $4 \cdot C$ crosstalk, so delay is reduced (due to the increase in wire spacing) only minimally.

In [17], [6] and [18], the authors focus on routing techniques which utilize crosstalk information about the wires being routed. The work of [19] aims to reduce crosstalk in datapath circuits. Our paper, on the other hand, focuses on crosstalk in buses (where the problem is significantly more acute since buses tend to be longer, resulting in larger capacitances and therefore more aggravated worst-case delays).

In [15], [5], [7], [11] and [16], the goal was to avoid $4 \cdot C$ and $3 \cdot C$ transitions. In contrast, our techniques can *speed up* a bus even further, by ensuring that the bus never exhibits $2 \cdot C$ transitions as well. In other words, our $2 \cdot C$ free approach can *exploit* cross-talk to speed up a bus further. Additionally, unlike the above papers as well as [4], our techniques utilize memory-based CODECs, resulting in much lower bus size overheads. Our algorithms to find the bus overhead utilizes ROBDDs [3], and thereby represents the vector transitions *implicitly and therefore compactly*. We report the bus size overheads for $4 \cdot C$, $3 \cdot C$ and $2 \cdot C$ free memory-based CODECs.

4. Memory-based Crosstalk Cancellation

For a memory-based code, let v_r be the vector present on the bus at time t_r. Let v_{r+1} be the vector present on the bus at time t_{r+1}. If it is guaranteed that for any r, $v_r \rightarrow v_{r+1}$ is a $p \cdot C$ transition, then the sequence is a $p \cdot C$ sequence (sufficient condition). For an m bit bus, such a sequence satisfies the property that at any given time t_r, there must be at least 2^m distinct $(p+1) \cdot C$ free transitions available. In other words, for any v_r, there must be at least 2^m distinct v_{r+1}'s which are $(p+1) \cdot C$ free with respect to v_r.

To decode the data, the receiving decoder needs to know both the current received symbol *and* the previously received symbol. The encoder generates the next symbol based on the data input and the previously transmitted symbol. As a consequence, memory elements are needed in both the encoder and decoder.

A memory-based code will satisfy the $(p+1) \cdot C$ free condition iff for each vector v in the set, there are at least 2^m vectors (including v itself) that are $(p+1) \cdot C$ free with respect to v. It is *not* required that every pair of vectors in the set is a $(p+1) \cdot C$ free pair.

4.1 Summary of our Approach

Our approach to determine the effective bus of width m that can be encoded in a $k \cdot C$ free manner, using a physical bus of width n consists of two steps:

- First, we construct an ROBDD $G_n^{kC-free}$ which encodes all vector transitions on the n-bit bus that are $k \cdot C$ free.
- From $G_n^{kC-free}$, we find the effective bus width m, such that an m bit bus can be encoded in a $k \cdot C$ free manner using $G_n^{kC-free}$.

These steps are described in the sequel.

4.1.1 Efficient Construction of $G_n^{kC-free}$

We employ an ROBDD [3] based construction of $G_n^{kC-free}$. In particular, we inductively compute $\overline{G_n^{kC-free}}$. Since the ROBDD of a function and its complement contain the same number of nodes (except for a complement pointer), this enables an efficient construction of $G_n^{kC-free}$. Further, the ROBDD allows us to represent the legal (and illegal) $k \cdot C$ free crosstalk transitions on the bus implicitly, sharing nodes maximally and in a canonical manner.

Suppose we want to construct $G_n^{kC-free}$. In that case, we allocate $2n$ ROBDD variables. The first n variables correspond to the vector from which a transition is made (referred to as $v = \{v_1, v_2, \cdots, v_n\}$). The next

n variables correspond to the vector to which a transition is made (referred to as $w = \{w_1, w_2, \cdots, w_n\}$). If a vector sequence $v^* \longrightarrow w^*$ is legal with respect to $k \cdot C$ crosstalk, then $w^* \longrightarrow v^*$ is also legal. In other words, $G_n^{kC-free}(v^*, w^*) = G_n^{kC-free}(w^*, v^*)$.

We construct the ROBDD for $G_n^{kC-free}$ by using ROBDDs for intermediate, partially $k \cdot C$ cross-talk free ROBDDs $\overline{G_i^{kC}}$ ($3 \leq i \leq n$). The construction of the ROBDD of G_n^{kC} proceeds inductively, starting with the base case of G_3^{kC}. The construction of G_3^{4C}, G_3^{3C} and G_3^{2C} are described next.

$$G_3^{4C} = \left[\begin{matrix} \overbrace{\begin{matrix} v_1 & v_2 & v_3 & v_4 & \cdots & v_n \\ 1 & 0 & 1 & - & \cdots & - \\ 0 & 1 & 0 & - & \cdots & - \end{matrix}}^{v} & \overbrace{\begin{matrix} w_1 & w_2 & w_3 & w_4 & \cdots & w_n \\ 0 & 1 & 0 & - & \cdots & - \\ 1 & 0 & 1 & - & \cdots & - \end{matrix}}^{w} \end{matrix}\right]$$

We initialize G_3^{3C} with the transitions of G_3^{4C}, and then append additional $3 \cdot C$ transitions:
$G_3^{3C} = G_3^{4C}$

$$G_3^{3C} += \left[\begin{matrix} \overbrace{\begin{matrix} v_1 & v_2 & v_3 & v_4 & \cdots & v_n \\ 0 & 0 & 1 & - & \cdots & - \\ 0 & 1 & 0 & - & \cdots & - \\ 1 & 0 & 1 & - & \cdots & - \\ 1 & 1 & 0 & - & \cdots & - \\ 0 & 1 & 0 & - & \cdots & - \\ 1 & 0 & 0 & - & \cdots & - \\ 0 & 1 & 1 & - & \cdots & - \\ 1 & 0 & 1 & - & \cdots & - \end{matrix}}^{v} & \overbrace{\begin{matrix} w_1 & w_2 & w_3 & w_4 & \cdots & w_n \\ 0 & 1 & 0 & - & \cdots & - \\ 0 & 0 & 1 & - & \cdots & - \\ 1 & 1 & 0 & - & \cdots & - \\ 1 & 0 & 1 & - & \cdots & - \\ 1 & 0 & 0 & - & \cdots & - \\ 0 & 1 & 0 & - & \cdots & - \\ 1 & 0 & 1 & - & \cdots & - \\ 0 & 1 & 1 & - & \cdots & - \end{matrix}}^{w} \end{matrix}\right]$$

In a similar manner, we initialize G_3^{2C} with the transitions of G_3^{3C}, and then append additional $2 \cdot C$ transitions. Note that the ROBDDs for $\overline{G_3^{4C}}$, $\overline{G_3^{3C}}$ and $\overline{G_3^{2C}}$ are *only partially free of* $4 \cdot C$, $3 \cdot C$ and $2 \cdot C$ transitions. They are immune to $4 \cdot C$, $3 \cdot C$ and $2 \cdot C$ transitions only on the first three bits of the bus. We utilize the complements of these ROBDDs to construct $\overline{G_n^{kC-free}}$ (for $k = 2, 3$ and 4) . The ROBDD $G_n^{kC-free}$ is free of $k \cdot C$ transitions for all n bits of the bus.

Algorithm 1 describes the inductive construction of $\overline{G_n^{kC-free}}$ from G_3^{kC}. In this algorithm, $G_3^{kC}((v_{i+1}, v_{i+2}, v_{i+3}) \leftarrow (v_1, v_2, v_3), (w_{i+1}, w_{i+2}, w_{i+3}) \leftarrow (w_1, w_2, w_3))$ refers to the ROBDD variable substitution of $(v_{i+1}, v_{i+2}, v_{i+3})$ and $(w_{i+1}, w_{i+2}, w_{i+3})$ by (v_1, v_2, v_3) and (w_1, w_2, w_3) respectively.

Algorithm 1 Inductive Construction of $\overline{G_n^{kC-free}}$ from G_3^{kC}

```
  for $i = 1$ $to$ $n-3$ do
    $G_{i+3}^{kC} = G_{i+2}^{kC} + G_3^{kC}((v_{i+1}, v_{i+2}, v_{i+3}) \leftarrow (v_1, v_2, v_3), (w_{i+1}, w_{i+2}, w_{i+3}) \leftarrow (w_1, w_2, w_3))$
  end for
  $\overline{G_n^{kC-free}} \leftarrow G_n^{kC}$
  return $\overline{G_n^{kC-free}}$
```

Note that only the final $\overline{G_n^{kC-free}}$ that is constructed using Algorithm 1 is utilized for CODEC construction, since the intermediate ROBDDs for G_i^{kC} ($i < n$) will possibly have $k \cdot C$ or greater crosstalk transitions, since they are expressed over $2n$ variables.

The final $G_n^{kC-free}$ encodes a family of Finite State Machines (FSMs) containing all legal transitions (in an implicit form using ROBDDs). From $G_n^{kC-free}$, we can find the effective size m of the bus that can be encoded. This procedure is discussed in Section **4.1.2**.

4.1.2 Finding the Effective $k \cdot C$ Free Bus Width from $G_n^{kC-free}$

If an m-bit ($m < n$) bus can be encoded using the legal transitions in $G_n^{kC-free}$, then there must exist a closed set of vertices $V_c \subseteq B^n$ in the v space of $G_n^{kC-free}(v, w)$ such that:

- Each source vertex $v_s \in V_c$ has at least 2^m outgoing edges (v_s, w_d) to destination vertices w_d (including the self edge), such that the destination vertex $w_d \in V_c$.
- The cardinality of V_c is at least 2^m.

The resulting encoder is memory-based.

Given $G_n^{kC-free}$, we find m using Algorithm 2. The input to the algorithm is m and $G_n^{kC-free}$. We first find the out-degrees (self-edges are counted) of each $v_s \in V$ (where V is a hash table). This is done by logically ANDing the ROBDD of the vertex v_s with $G_n^{kC-free}$. We find the cardinality of the resulting ROBDD – it represents the out-degree of v_s. If the number of out-edges of any v_s is greater than 2^m, we add v_s (and its out-degree) into the hash table V.

For each $v_s \in V$, we next check if each of its destination nodes w_d are in V. If $w_d \notin V$, we decrement the out-degree of v_s by 1. If the out-degree of v_s becomes less than 2^m, we remove v_s from V.

These operations are performed until convergence. If at this point, the number of surviving vertices in V is 2^m or more, then an m-bit memoryless CODEC can be constructed from $G_n^{kC-free}$.

We initially call the algorithm with $m = n - 1$ (where n is the physical bus size). If an m bit bus cannot be encoded using $G_n^{kC-free}$, then we decrement m. *We repeat this until we find a value of m such that the m-bit bus can be encoded by* $G_n^{kC-free}$.

Once we know the effective bus size m, we can construct an FSM for the encoder and decoder. There is significant flexibility in constructing the FSMs.

- From the vertices in V, we can select a subset V^{FSM} such that $|V^{FSM} = 2^m|$.
- Once this selection is made, we have further flexibility in assigning the 2^m labels out of each $v \in V^{FSM}$.

In our current implementation, we make both these selections randomly. An interesting avenue for future work is to devise an optimal algorithm to make the selections in each of the above steps, in order to minimize the size of the resulting CODEC implementation.

Algorithm 2 Testing if $G_n^{kC-free}$ can encode an m-bit bus

```
  $test\_encoder(m, G_n^{kC-free})$
  $find\ out-degree(v_s)\ of\ each\ node\ v_s$, insert $(v_s, out-degree(v_s))$ in $V$ if $out-degree(v_s) \geq 2^m$
  $degrees\_changed = 1$
  while $degrees\_changed$ do
    $degrees\_changed = 0$
    for $each\ v_s \in V$ do
      for $each\ w_d$ S.T. $G_n^{kC-free}(v_s, w_d) = 1$ do
        if $w_d \notin V$ then
          $decrement\ out-degree(v_s)$ in $V$
          $degrees\_changed = 1$
        end if
        if $out-degree(v_s) < 2^m$ then
          $V \leftarrow V \setminus v_s$
          $break$
        end if
      end for
    end for
  end while
  if $|V| \geq 2^m$ then
    print($m$ bit bus can be encoded using $G_n^{kC-free}$)
  else
    print($m$ bit bus cannot be encoded using $G_n^{kC-free}$)
  end if
```

5. Experimental Results

We ran our algorithm to construct $G_n^{kC-free}$ and to find the effective bus width m for $4 \cdot C$, $3 \cdot C$ and $2 \cdot C$ free transitions. The ROBDD of $G_n^{kC-free}$ is constructed as described in Section **4.1.1**, and the test for the effective bus width is performed as described in Section **4.1.2**. Based on the values of the real bus width n and the corresponding effective bus width m, we plot the overhead (defined as $\frac{n-m}{m}$) in Figure 1. The irregularity of the curves arises from the fact that sometimes, two buses (of width n and $n+1$) have an identical effective bus width of m. In that case, the overhead as defined above is larger for the bus of real width $n+1$.

Note that this figure indicates that the asymptotic bus size overheads for the memory based CODECs are much lower than the memoryless CODEC overheads reported in [5, 4]. The overhead for a $2 \cdot C$ free memory based CODEC is about 117% compared to 146% for a memoryless CODEC. The overhead for a $3 \cdot C$ free memory based CODEC is about 30% compared

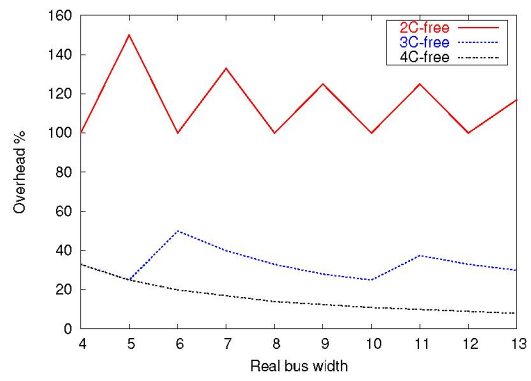

Figure 1: Bus size overheads for Memory-based CODECs

bus trace length	buffer size	1C delay (ps)	2C delay (ps)	3C delay (ps)	4C delay (ps)
5mm	30×	121	241	516	665
5mm	60×	131	213	399	402
5mm	120×	117	136	196	279
10mm	30×	153	437	912	1026
10mm	60×	164	413	722	919
10mm	120×	137	270	379	548

Table 1: Delay comparison for different driver size and trace length (ps)

to 44% for a memoryless CODEC. The overhead for a $4 \cdot C$ free memory based CODEC is about 8% compared to 33% for a memoryless CODEC.

For wider buses, we recommend that the bus be partitioned into smaller bus segments (with inter-segment cross-talk eliminated as outlined in [5]), and each segment be encoded and decoded independently. In such a situation, we could choose a bus width n that yields the lowest overhead, by referring to Figure 1. In particular, the choice of 4 or 6 bit segments is preferable over 5 or 7 bit segments, if we were trying to eliminate $2 \cdot C$ cross-talk.

The standard cell based implementation of the encoder and decoder results in a delay of 280ps, using a 0.1μm *bsim100* process [1] (this is the worst delay among all decoders and decoders required for any of the $2{\cdot}C$-free, $3{\cdot}C$-free and $4{\cdot}C$-free approaches). A Programmable Logic Arrays (PLA) based realization may reduce this delay further.

The implementation of the memory-based CODECs is more complex (compared to memoryless CODECs). Our experiments demonstrate that the encoder/decoder area and delay penalty of our approach (over the memoryless encoders and decoders of [5, 4]) is about 15% and 10% respectively.

However, the *total area of the memory-based approach (including the area of bus wiring) is about 25%, 10% and 20%* lower than the memoryless approach (for buses that are free of $2 \cdot C$, $3 \cdot C$ and $4 \cdot C$ transitions respectively).

In spite of the 10% delay increase of our memory-based CODECs over memoryless CODECs, the bus operates faster compared to an unencoded bus. Further, encoding and decoding delays are *unimportant for heavily pipelined systems*, where these delays can be hidden. In case of heavily pipelined systems, the maximum data-rate is significantly improved by using our encoding schemes, just as in the case of the memoryless approach.

Table 1 reports the worst-case delay among the bus signals under all cross-talk conditions. The results were generated using SPICE [14]. A 0.1μm BPTM *bsim100* [1] process was used, and buses were assumed to be routed on Metal4. Wiring parasitics were obtained from [2] using the interconnect dimensions reported in [10, 9]. Wires were modeled as distributed RC transmission lines. In Table 1, the first column reports the length of the bus wires. Column 2 reports the driver size (in multiples of a minimum-sized driver). Columns 3 through 6 report the worst case delay of the bus in picoseconds, assuming that no greater than 1·C through 4·C cross-talk patterns are allowed respectively.

From the above, we can note that eliminating $2 \cdot C$ transitions on a bus (i.e. the bus has no greater than $1 \cdot C$ transitions) can speed up the bus significantly. For example, in the configuration of 60× drivers, 10mm wires, the delay reduces by about 5.6× compared to the unencoded (i.e. the bus has no greater than $4 \cdot C$ transitions) case.

5.1 Robustness of the Approach

In this section, we discuss the robustness of our approach in terms of its applicability to wide on-chip buses.

For wide buses, we propose that the bus be partitioned into smaller segments (whose size is determined by the results from Figure 1). The resulting segments are quite small (with a real width of 4, 5 and about 6[4] bits for $2{\cdot}C$-free, $3{\cdot}C$-free and $4{\cdot}C$-free buses respectively. Each segment has independent encoders and decoders, with static or dynamic shields between segments for inter-segment cross-talk immunity (as described in [5, 4]). The worst case delay of the resulting standard cell based implementation (the worst case delay is the largest delay of the encoder or decoder, for any of the $2{\cdot}C$-free, $3{\cdot}C$-free and $4{\cdot}C$-free approaches) is about 280ps, for a 0.1μm *bsim100* [1] process. As a result, the overall delays on the bus are reduced for longer on-chip buses with reasonably sized drivers, *even when encoding and decoding delay is accounted for*. However, in case of heavily pipelined systems, the maximum data-rate is significantly improved by using our encoding schemes. In such pipelined systems, the *encoding/decoding delays are unimportant*.

6. Conclusions

We have developed a CODEC based approach to alleviate the cross-talk problem in on-chip buses. Our CODECs are memory-based. The construction of these CODECs is performed using an ROBDD-based approach. By using this approach, we avoid explicitly enumerating all legal bus transitions (and instead enumerate only the illegal transitions). The ROBDD based approach represents the transitions compactly and in a canonical fashion. We have also developed a ROBDD based approach to find the effective $k \cdot C$ free bus bandwidth m of a bus with physical width n.

We demonstrate that the asymptotic bus size overheads for memory-based CODECs that are free of $2 \cdot C$, $3 \cdot C$ and $4 \cdot C$ transitions are respectively around 117%, 30% and 8%. This is in contrast to the memoryless CODEC overheads for the same transitions, which are 146%, 44% and 33% respectively. The overall area reduction (compared to memoryless CODECs) of our approach is about 25%, 10% and 20% for $2 \cdot C$, $3 \cdot C$ and $4 \cdot C$ free encoded buses respectively. The user can trade off the bus size overhead against the cross-talk immunity that is desired. According to our experiments, this tradeoff can result in a bus speed up of above 6×.

References

[1] BPTM Homepage. `www-device.eecs.berkeley.edu/~ptm/`.

[2] Physical Design Modelling and Verification Project (SPACE Project). `http://cas.et.tudelft.nl/research/space/html`.

[3] R. E. Bryant. Graph based algorithms for Boolean function representation. *IEEE Transactions on Computers*, C-35:677–690, August 1986.

[4] C. Duan and S. Khatri. Exploiting crosstalk to speed up on-chip buses. In *Proceedings, Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 778–783, Paris, France, Feb 2004.

[5] C. Duan, A. Tirumala, and S. Khatri. Analysis and avoidance of cross-talk in on-chip buses. In *Hot Interconnects 9*, pages 133–138, Stanford, CA, Aug 2001.

[6] T. Gao and C. Liu. Minimum crosstalk channel routing. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 692–696, Santa Clara, CA, Nov 1993.

[7] K. Hirose and H. Yasuura. A bus delay reduction technique considering crosstalk. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 441–445, Paris, France, Mar 2000.

[8] The International Technology Roadmap for Semiconductors. `http://public.itrs.net`, 2003.

[9] S. Khatri, R. Brayton, and A. Sangiovanni-Vincentelli. *Cross-Talk Noise Immune VLSI design using regular layout Fabrics*. Kluwer Academic Publishers, 2001. ISBN 0-7923-7407-X.

[10] S. Khatri, A. Mehrotra, R. Brayton, A. Sangiovanni-Vincentelli, and R. Otten. A novel VLSI layout fabric for deep sub-micron applications. In *Proceedings of the Design Automation Conference*, New Orleans, June 1999.

[11] D. Li, A. Pua, P. Srivastava, and U. Ko. A repeater optimization methodology for deep sub-micron, high-performance processors. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*, pages 726–731, Austin, TX, Oct 1997.

[12] C.-G. Lyuh and T. Kim. Low power bus encoding with crosstalk delay elimination. In *15th Annual IEEE International ASIC/SOC Conference*, pages 389–393, Sept 2002.

[13] L. Macchiarulo, E. Macii, and M. Poncino. Wire placement for crosstalk energy minimization in address buses. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 158–162, Paris, France, Mar 2002.

[14] L. Nagel. Spice: A computer program to simulate computer circuits. In *University of California, Berkeley UCB/ERL Memo M520*, May 1995.

[15] P. Sotiriadis and A. Chandrakasan. Reducing bus delay in submicron technology using coding. In *Proceedings Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 109–114, Yokohama, Japan, Jan/Feb 2001.

[16] B. Victor and K. Kuetzer. Bus encoding to prevent crosstalk delay. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 57–63, San Jose, CA, Nov 2001.

[17] A. Vittal and M. Marek-Sadowska. Crosstalk reduction for vlsi. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(3):290–298, Mar 1997.

[18] T. Xue, E. Kuh, and D. Wang. Post global routing crosstalk synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(12):1418–1430, Dec 1997.

[19] J.-S. Yim and C.-M. Kyung. Reducing cross-coupling among interconnect wires in deep-submicron datapath design. In *Proceedings. 36th Design Automation Conference (DAC)*, pages 485–490, New Orleans, LA, Jun 1999.

[4] For $4{\cdot}C$-free buses, the bus overhead is quite low, and slowly decreases with increasing real bus width. In order to minimize the CODEC complexity, we select a segment size of 6, although larger segments result in a slightly lower overhead