Enhancing Signal Integrity through a Low-overhead Encoding Scheme on Address Buses

Tiehan Lv, Princeton University
Jörg Henkel, NEC, USA
Haris Lekatsas, NEC, USA
Wayne Wolf, Princeton University

Abstract

Signal integrity is and will continue to be a major concern in deep sub-micron VLSI designs where the proximity of signal carrying lines leads to crosstalk, unpredictable signal delays and other parasitic side effects. Our scheme uses bus encoding that guarantees that at any time any two signal carrying lines will be separated by at least one grounded line and thus providing a high degree of signal integrity. This comes at a small overhead of only one additional bus line (the closest related work needs 14 additional lines for a 32-bit bus) and a small average performance decrease of 0.36%. By means of a large set of real-world applications, we compare our scheme to other state-of-the-art approaches and present comparisons in terms of degree of integrity, overhead (e.g. additional lines required) and a possible performance decrease.

1 Introduction

Increasing signal integrity is an important challenge in deep sub-micron designs since the proximity of signal carrying lines leads to coupling induced by electromagnetic fields. Undesired coupling can lead to various effects: a) a line 'a' may falsely trigger an adjacent line 'b' and thus alter the data word on a bus line, for example; b) due to the electromagnetic coupling between two adjacent lines 'a' and 'b', the signal on line 'a' may be delayed by line 'b', or vice versa, and lead to either a delayed signal or even a logic error due to the late signal arrival. Designers have sought to minimize the probabilities of these effects by various means that can be divided into physical approaches and higher-level approaches. A straightforward physical (geometrical) approach is to simply increase the distance between two signal-carrying lines. The integrity will then approximately increase with the square of the distance between two signal lines. However, this method may be costly in terms of area especially when applied to wide buses (32-bit, 64-bit). Another physical means to reduce coupling is through usage of selected materials: for example, choosing materials with a high dielectric constant applied as a thin layer on signal lines can significantly reduce mutual coupling effects. Other methods are routing-related as they aim to periodically segment the lines of a bus. Then, the interfaces of the segments are transposed and thus preventing two bus lines from being adjacent for the entire length of the bus.

In recent years various approaches for bus encoding have been proposed (see related work) that aim to send more information via bus lines but have less energy consumption. It should be noted that a majority of these approaches increase the informational entropy and thus increasing the bus lines' susceptibility to coupling effects. This is an additional reason for the increasing importance to decrease coupling effects. Our approach addresses the signal integrity problem at a high abstraction level: rather than changing the physics (geometry, materials) of signal carrying lines, we assign logical bus lines to physical bus lines by an encoding scheme. This comes at a small overhead of only one additional bus line (the closest related work needs 14 additional lines for a 32-bit bus) and an average performance decrease of 0.36%.

The rest of the paper is organized as follows: Section 2 gives the problem description and motivation while Section 3 introduces the related work. Our encoding/shielding method is presented in Section 4 whereas experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Problem and Motivation

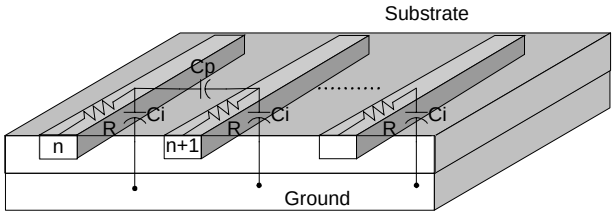

Fig. 1: Simplified physical bus model

Fig. 1 shows a simplified physical bus model as a cross-sectional cut. In a first approximation the major capacitances can be represented as a base (intrinsic) capacitance between bus line and metal layer(s) and a coupling (or 'inter-wire') capacitance between two adjacent bus lines. Modeled as a RC circuit, we obtain the rise time of a signal transmitted via a bus line that shows the dependency on the capacitances and the

1530-1591/03 $17.00 © 2003 IEEE

resistance (with rise time being the time when 90% of the whole voltage swing is reached), we obtain:

$$rise\_time = -RC \ln \frac{0.1}{0.9} \approx 2.2RC \qquad \text{Eq. 1}$$

For the purpose of maximizing the bus clock rate, it is desirable to minimize RC. The coupling capacitances are also responsible for signal coupling (crosstalk) effects.

Unfortunately, the advent of silicon technology with shrinking feature sizes actually *increases* the inter-wire capacitances.

A high-level (i.e. non-physical/geometrical) solution has been proposed by Victor/Keutzer [2] who eliminate crosstalk delay by a scheme that needs additional bus lines (explained in more detail later). They address only the worst-case transitions on two adjacent lines i.e. those transitions where one line switches from '1' to '0' and the adjacent line switches from '0' to '1' and vice versa (according Fig. 2). In this case, the effect of the inter-wire capacitance is effectively doubled due to the charge/discharge processes.

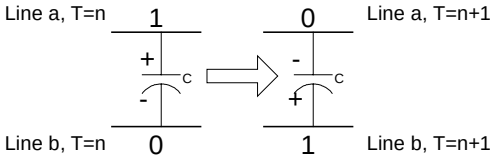

Fig. 2 Worse case transitions

Their (Victor/Keutzer) scheme converts a bus data sequence into a self-shielding sequence in which no two adjacent bus lines will change in opposite directions at the same time. Thus, the before-mentioned worst-case transitions are eliminated. However, this approach does not provide full bus shielding, since it does not solve those cases where a transition on one bus line triggers the transition of another bus line. In short, their approach reduces worst-case cross talk but does not shield in all transition cases. In addition, they need an overhead of 14 bus lines (46 on a 32-bit bus).

Another possible approach to solve the problem is the usage of a pipelined bus as shown in Fig. 3: A bus is separated into segments.

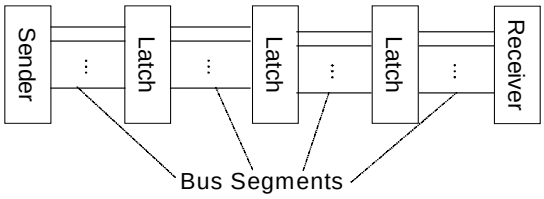

Fig. 3: Discussing the advantages/disadvantages of a pipelined bus in terms of crosstalk effects

Fig. 4 shows the effect on inter-wire capacitances which are effectively halved in each section (assuming a separation in two segments) due to the length reduction (same holds for the intrinsic capacitances).

As a consequence, crosstalk is obviously reduced (since inter-wire capacitances decreased). However, this approach does not provide shielding (falsely triggered signals can still occur), it only reduces the *probability*.

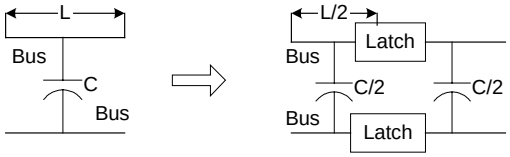

Fig. 4 Inter-wire capacitances in pipelined buses

Our approach, on the other side, provides full shielding (as far as the closest signal carrying bus line is concerned) at an overhead of only one additional line and a small performance decrease of 0.36%. Using our solution, any two adjacent signal carrying bus lines are separated by at least one grounded line at any time.

3 Related Work

Besides the paper from Victor/Keutzer (see previous section) which is the closest work related to our approach, we discuss in the following routing related work as well as approaches to bus encoding (even though the latter may not be aimed at signal integrity).

Zhou/Wong propose a global routing scheme with crosstalk constraints [4] that is based on Steiner Tree and Lagrangian Relaxation techniques. Kirkpatrick and Sangiovanni-Vincentelli show that crosstalk channel routing is an NP-complete problem and they propose a heuristic algorithm for a crosstalk-avoiding router [8]. Chang/Cong introduce a method for crosstalk-controlling routing using a pseudo pin assignment algorithm [5]. Kastner et al. use a model for the coupling free routing (CFR) problem to describe the cross-talk controlled routing [6]. Their model aims to solve the CFR problem and generates a crosstalk-reduced net. Jiang et al. propose an algorithm based on Lagrangian Relaxation to solve an optimization problem related to simultaneous switching [7].

Address buses have drawn increased interest from researchers due to their regularity. Many of the work has focused on low-power encoding, though. Mehta et al. introduce gray code for address bus producing fewer transitions when the memory is accessed sequentially [14] Benini et al. propose a method for low power bus encoding, which uses the fact that a processor accesses the instructions mostly in a sequential way [9]. The receiving end of the address bus calculates the address by adding an offset to the last received address rather than receiving the whole new address. In the case, such a prediction is correct, the bus transition can be reduced and energy associated with the transition is reduced. Hsieh and Pedram propose a method to reduce bus

energy consumption by using a split bus [12]. In this approach, the whole system bus is separated into several segments so that those transactions inside each segments have less associated energy since the capacitance, delay etc. is reduced on each bus segment.

Some recent work that is close to ours, proposed bus-encoding methods to reduce crosstalk (See introduction section on Victor/Keutzer's work). However, the scheme requires a substantial increase of the bus width.

4 Deriving a Shielding Scheme

We will first introduce experiments on data locality on address buses as this is the prerequisite for our approach. We then introduce our shielding scheme and the necessary hardware architecture.

4.1 Data Locality

In a 32-bit processor, the memory space is 4GB. However, few programs actually utilize the whole memory space. Even if an application does need a large memory space, it usually does not evenly spread accesses across the *whole* memory space. Rather, a program tends to spend execution in memory clusters for some time and gradually switch to other clusters [10]. This memory locality is well known and caches have been built to speed up memory access by targeting this locality. The uneven usage of memory space provides an opportunity for us to increase signal integrity. Since memory accesses tend to be clustered, the upper part of the address bus does not change as frequently as the lower part of the address bus does.

Therefore, we may consider separating the address bus in a lower and upper part and then only transfer the lower (more often changing) part. The not-needed bus lines could then somehow be used for shielding.

A critical factor of such an approach is that the frequency the upper part needs to be transferred has to be kept low, such that there is a real gain.

Patterson/Hennessy show that few branches have offsets that need 16 bits to describe [10]. We performed experiments to justify our approach on real-world applications. We use the following formula in the experiment:

$$d(x_i) = x_i - x_{i-1},\ r = \frac{card\{x : |d(x)| < 64kbytes\}}{card(\{x : all\ x\})} \qquad \text{Eq. 2}$$

Here d is displacement, x_i is the current instruction address, and x_{i-1} stands for the previous instruction address. $card(\cdot)$ calculates the number of elements in a set. X stands for address, while r is the percentage of the displacements that fall into a 64kbytes range (16 bits). The range is 32kbytes or 128kbytes for 15 or 17 bits respectively. The simulation results are pictured in Fig. 5: Shown are three address ranges (15, 16 and 17 bit) for each application. It can be observed that in all the cases 16 bits cover more than 98.5% of the displacements whereas the ratio for 15 bits is slightly lower and for 17 bit insignificantly higher. This characteristic will be used by our encoding scheme to provide for shielding as shown in the following.

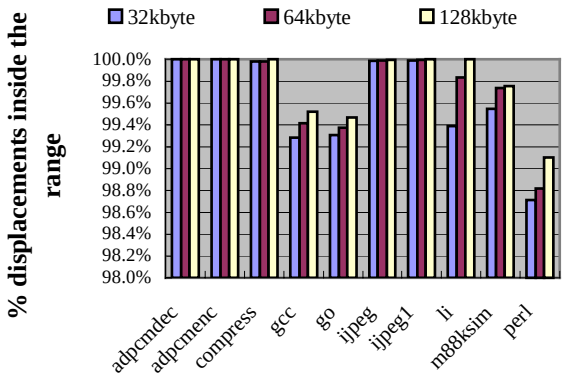

Fig. 5: Percentage of displacement

4.2 Our Shielding Scheme

According to the above experiments, the upper 16 bits (the reason that the upper part is 16 bits will be explained later) are changing infrequently and the lower 16 bits cover almost all the addressing. Looking from another angle we can state that there is a large amount of data redundancy since the informational entropy in the upper 16 bus lines is much lower compared to the lower 16 bus lines.

This leads us to the idea to apply compression techniques to the address bus and reserve superfluous bus lines to act as shields (grounded). As we will show in the following, we apply a dictionary-based compression scheme [13]. Dictionary-based methods are not only providing reasonable compression but they are also low in implementational effort.

Our adaptive dictionary encoding scheme works as follows: Two register cells (i.e. 'dictionaries') are placed on both sides of the buses. Each time after the sender (encoder) processes a word, it saves the word into its register. The receiver (decoder) does the same when it receives a word. When a sender has a word to process, it first identifies the displacement (see Eq. 2) between the current word and the word previously saved. If the displacement is small then the current word is compressible, and only the displacement needs to be transmitted. The receiver can recover the original word by adding the received displacement to the previously saved word. If the displacement is large, the current word is not compressible (this case is explained later). After each bus transaction, the receiver updates its dictionary with the word received. This way, the contents of the two dictionaries are synchronized.

An important factor in this scheme is the way we divide the bus into upper part and lower part. It is clear that when the upper part is not in the dictionary, additional cycles have to be used since the effective bus width is less than the original one (note that the other bus lines will be used for shielding). Suppose the lower part of the bus has the width *n* and the original bus has width *N*, then the number of additional necessary bus cycles $N_{add\text{-}bcyc}$ can be calculated as follows. (Here $\lceil x \rceil$ is the smallest integer that is greater than or equal to x.)

$$N_{add-bcyc} = \left\lceil \frac{N-n}{n} \right\rceil$$

Obviously, we want to keep the bus bandwidth high and therefore we need to keep $N_{add\text{-}bcyc}$ low. However, we also aim to keep the compressed (lower) bus part as narrow as possible. In addition, we aim to not have more than one additional bus cycle for the addresses, which cannot be compressed. This suggests *n=16* and that is also in compliance with the observations made earlier (see Fig. 5).

Our bus encoding and decoding algorithms are shown in Fig. 6 and Fig. 7: when a new input word (address) finds the upper match (i.e. upper part) in the sender's dictionary, then the encoder sends out only the lower part and signals a successful compression on the status line (an additional line). Otherwise, the encoder sends out the input word in two cycles (this case will only rarely happen), first the lower part, and then the upper part. At the same time, the encoder indicates a compression failure on the status line.

Encoder :
($Data_k$ is the current word to be sent, Bus_k is the value sent on the bus.)
$Bus_k\langle 0:15\rangle = Data_k\langle 0:15\rangle$;
if $EncDict = Data_k\langle 16:31\rangle$ **then**
(match : do not change upper part of bus to save energy)
$Bus_k\langle 32\rangle = 1$; (signal a match)
else
$Bus_k\langle 32\rangle = 0$; (signal a miss)
(update dictionary)
$EncDict = Data_k\langle 16:31\rangle$;
(miss : send the original data)
$Bus_k\langle 0:15\rangle = Data_k\langle 0:15\rangle$;

Fig. 6: Encoder scheme

4.3 Implementation Issues

For an efficient (i.e. low area, low latency) implementation, we have taken several means: first, we use bit-wise *xor* operations to accomplish subtraction. The sender (encoder) applies an *xor* to obtain the difference (bit-wise) between current and previous data (address) and the receiver (decoder) does the same to recover the data (address). Secondly, we can just keep the upper part of a word while simply passing the lower part. At the receiver, the lower part can be combined with the upper part from the dictionary to recover the full word.

Decoder :
(Bus_k is the value received on bus, $Data_k$ is the output of the decoder.)
$Data_k\langle 0:15\rangle = Bus_k\langle 0:15\rangle$;
if $Bus_k\langle 32\rangle = 1$ **then**
(match : recover data from dictionary)
$Data_k\langle 16:31\rangle = DecDict$;
else
(miss : use bus data)
$Receive\ in\ the\ \text{sec}\,ond\ cycle\ \ Data_k\langle 16:31\rangle = Bus_k\langle 0:15\rangle$;
(update dictionary)
$DecDict\ = Bus_k\langle 0:15\rangle$;

Fig. 7: Decoder scheme

Fig. 8 shows the simple hardware of the bus encoding/decoding and shielding scheme (shown for a one-entry dictionary). It results an area cost equal to 380 AND gates for an encoding/decoding pair.

Our shielded bus has 33 lines including 17 signal lines (16 bus lines plus one status line) and 16 grounded lines (0V) for shielding.

The ratio of uncompressible input words over all input words is critical. If the ratio is low, then we are able to transmit the data on the address bus in almost one cycle. In the following, we will denote this as a "miss ratio".

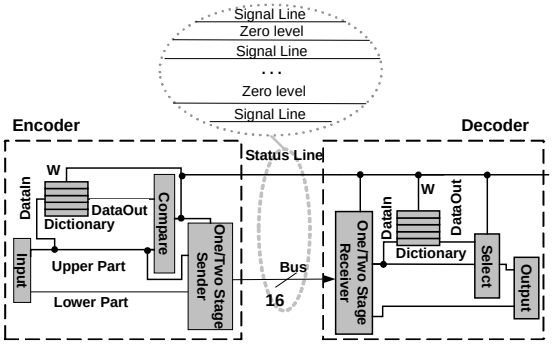

Fig. 8: Implementation

Improving the miss ratio can be controlled by the number of entries and their organization provided in the dictionary. A dictionary can have more than one entry. The index of the matched entry needs to be transferred in the lower part. We call the width of this index **entry_depth**. The entries in the dictionary can also be divided into groups and indexed by a part of the input address. We use term **index_width** for the number of the groups. In the following section, we conduct experiments to explore the best parameterization for our scheme.

5 Experiments and Results

It is our goal to minimize the ***miss ratio*** i.e. the number of times where our scheme cannot compress the bus word and therefore needs an additional cycle to send the word via the fully shielded bus. Fig. 9 and Fig. 10 show these *miss ratios* in dependency of the *index_width* and *entry_depth* for the instruction address bus and data address bus as an average of all applications (our simulations are based on a sub-system comprising a CPU, L1 caches for data and instructions and the address and data buses; see also Fig. 11). As a compromise between effort of implementation and sufficiently low miss ratios, we have chosen *index_width=16, entry_depth=4*. The results for these parameters are comprised in Table 1. Thus, we obtain a low miss ratio of 1.6% in average (we will show later that the all over performance decrease is even lower than this).

Application	Description	Instr. Addr.	Data Addr.
adpcm-enc	ADPCM voice encoder	0.00%	0.00%
adpcm-dec	ADPCM voice decoder	0.00%	0.00%
compress	File compr. (UNIX)	0.00%	0.28%
gcc	GNU C compiler	0.24%	2.55%
go	game from SPEC95	0.04%	8.87%
Ijpeg	JPEG encoder/decoder	0.00%	0.02%
li	Lisp interpreter	0.02%	0.44%
m88ksim	ISS for M88k	0.04%	0.20%
Perl	script lang. interpreter	0.19%	2.01%
	average	*0.06%*	*1.60%*

Table 1: Applications and their encoding miss ratios on instruction and data address buses

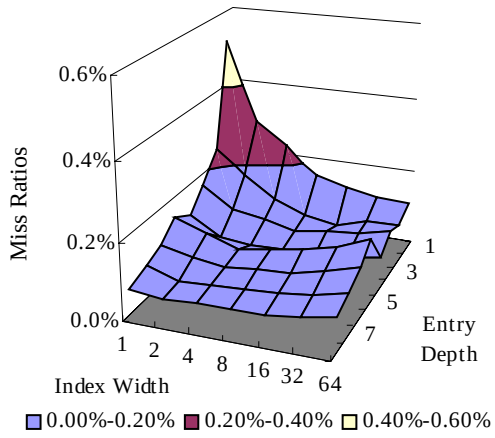

Fig. 9 Average miss ratios on an instruction address bus

In addition to the miss ratios from above, it is necessary to measure the actual performance in terms of used clock cycles and total execution time (the latter accounts also for latency). Our experimental setup is as follows: we used the *sim-outorder* of the *SimpleScalar* tool-suite [11] and modified it according to Fig. 11 i.e., we enhanced it by encoding/decoding subroutines and simulated all applications assuming the shown computer sub-system.

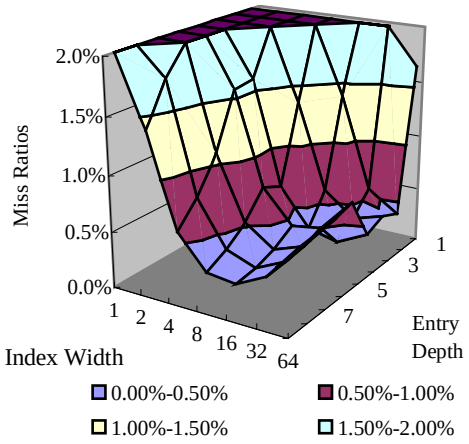

Fig. 10 Average miss ratios on a data address bus

We used system configurations (e.g. cache sizes etc.) that seemed appropriate for every application (note that they vary in size). The ratio between base and coupling capacitance has been chosen to 3.0 (this is in compliance with [3] and also with our own simulations). Furthermore, we assume a $0.10\mu m$ silicon technology that has a global wire delay of 3.4ns [1] (we scaled this number according to our dimension of a 10 mm address bus length). Another assumption is that the wire delay is the bottleneck for the clock cycle.

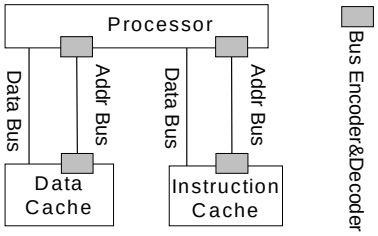

Fig. 11 Modifying SimpleScalar by integrating encoding and decoding modules

Table 2 gives detailed results comparing the "Basic" bus (no shielding), the "Shielding Bus" (adding one grounded line for each signal carrying line), the "Self-shielding Bus" (Victor/Keutzer *[2]*), the *"Pipelined Bus"* (according to Fig. 3), *"Our Scheme"* (see Section 4) and *"Our Scheme + Pipeline"* (our scheme applied on top of the pipelined bus scheme).

"Our Scheme" and *"Our Scheme +Pipeline"* feature the highest shielding (any two signal carrying bus lines are always separated by a grounded line) at 33 bus lines. Except for Victor/Keutzer [2], all other schemes are either unacceptable in terms of shielding or high effort (# bus lines). Compared to our approach, Victor/Keutzer need more bus lines (46) and provide only "medium"

shielding since they address only the worst-case transitions on two adjacent lines i.e. those transitions where one line switches from '1' to '0' and the adjacent line switches from '0' to '1'. Other cases are not covered by their approach. However, Victor/Keutzer do not incur any performance penalty as seen in Table 2 that shows for all approaches and all applications the execution time in terms of cycles and absolute time. Our approaches do show a performance penalty in some cases but the allover penalty is rather small at an average of 0.36% in terms of execution cycles. Therefore, we provide the highest shielding protection at only one bus line overhead (32+1). In lieu to these advantages the slight performance decrease seems very reasonable. Moreover, we predict that our bus can be clocked higher due to the reduced coupling capacitance (see also Eq. 1). That would result in a decreased clock latency and thus, the actual execution time of an application might be even smaller and actually yield a gain in performance. This, however, has not been done and we refer it to our future work.

	"Basic"	***"Shielding Bus"***	***"Self-shielding Bus"[2]***	***"Pipelined Bus"***	**"Our Scheme"**	**"Our Scheme + Pipeline"**
Number of Bus Lines	32	64	46	32	**33**	**33**
Shielding Protection	N/A	High	Medium	Low	**High**	**High**
Benchmark	**Application Execution Time (cycle count)**					
adpcmdec	3.78E+06	3.78E+06	3.78E+06	4.07E+06	**3.78E+06**	**4.07E+06**
adpcmenc	5.64E+06	5.64E+06	5.64E+06	6.11E+06	**5.64E+06**	**6.11E+06**
compress	1.97E+06	1.97E+06	1.97E+06	1.98E+06	**1.97E+06**	**1.98E+06**
gcc	1.31E+09	1.31E+09	1.31E+09	1.35E+09	**1.33E+09**	**1.36E+09**
go	1.83E+08	1.83E+08	1.83E+08	1.88E+08	**1.85E+08**	**1.90E+08**
ijpeg	2.79E+08	2.79E+08	2.79E+08	2.84E+08	**2.79E+08**	**2.84E+08**
li	6.37E+08	6.37E+08	6.37E+08	6.61E+08	**6.37E+08**	**6.61E+08**
m88ksim	1.15E+05	1.15E+05	1.15E+05	1.18E+05	**1.15E+05**	**1.18E+05**
perl	1.98E+09	1.98E+09	1.98E+09	2.02E+09	**2.00E+09**	**2.04E+09**
benchmark	**Application Execution Time (seconds)**					
adpcmdec	1.28E-02	7.34E-03	7.34E-03	6.91E-03	**7.34E-03**	**3.95E-03**
adpcmenc	1.92E-02	1.09E-02	1.09E-02	1.04E-02	**1.09E-02**	**5.94E-03**
compress	6.69E-03	3.83E-03	3.83E-03	3.36E-03	**3.83E-03**	**1.92E-03**
Gcc	4.47E+00	2.55E+00	2.55E+00	2.29E+00	**2.58E+00**	**1.32E+00**
Go	6.22E-01	3.55E-01	3.55E-01	3.20E-01	**3.59E-01**	**1.84E-01**
ijpeg	9.50E-01	5.43E-01	5.43E-01	4.82E-01	**5.43E-01**	**2.76E-01**
Li	2.17E+00	1.24E+00	1.24E+00	1.12E+00	**1.24E+00**	**6.42E-01**
m88ksim	3.92E-04	2.24E-04	2.24E-04	2.00E-04	**2.24E-04**	**1.14E-04**
Perl	6.73E+00	3.85E+00	3.85E+00	3.44E+00	**3.88E+00**	**1.98E+00**

Table 2 Comparison among different methods

6 Conclusion

We have presented an encoding/decoding scheme that capitalizes on the characteristics of data sent via addresses buses. The scheme uses actually dictionary-based compression and effectively needs less physical bus lines. We then take the free bus lines to obtain a high shielding protection such that at any time any two signal carrying adjacent bus lines are separated by a grounded line. We have compared our method to state-of-the art methods in terms of wire overhead, shielding protection and performance. With these benefits, our approach incurs only a slight miss ratio penalty of 1.6% in average and an allover average performance penalty of a mere 0.36%.

References

[1]. M. Horowitz, R. Ho, and K. Mai, "Wires: A user's guide", SRC/Marco workshop at CIS, Stanford, May 1999.

[2]. B.M. Victor and K. Keutzer, "Bus Encoding to Prevent Crosstalk Delay", in ICCAD, pp. 57--63, Nov 2001.

[3]. Z. Huang and M.D. Ercegovac, "Effect of Wire Delay on the Design of Prefix Adders in Deep-Submicron Technology", Proc. 34th Asilomar Conference on Signals, Systems and Computers, 2000.

[4]. H. Zhou and D.F. Wong, "Global routing with crosstalk constraints", In 35th Design Automation Conference, pp. 374--377, 1998.

[5]. C.-C. Chang and J. Cong, "Pseudo pin assignment with crosstalk noise control," in Proc. Int. Symp. on Physical Design, pp. 41--47, 2000.

[6]. R. Kastner, E. Bozorgzadeh and M. Sarrafzadeh. "Coupling Aware Routing", In IEEE International ASIC/SOC Conference, Sept 2000.

[7]. H.-R. Jiang, Y.-W. Chang, and J.-Y. Jou, "Crosstalk-driven interconnect optimization by simultaneous gate and wire sizing", IEEE Trans. on Computer-Aided Design, Vol. 19, No. 9, pp. 999--1100, Sept 2000.

[8]. D.A. Kirkpatrick & A.L. Sangiovanni-Vincentelli, "Techniques for Crosstalk Avoidance in the Physical Design of High-Performance Digital Systems", ICCAD'94, pp 616-620, 1994.

[9]. L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems," Great Lakes VLSI Symposium, pp. 77-82, Urbana IL, Mar 1997.

[10]. J. L. Hennessy, and D. A. Patterson, "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publ., 1996

[11]. D. Burger, and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0", Technical Report 1342, University of Wisconsin Madison, CS Department, Jun 1997.

[12]. C-T. Hsieh and M. Pedram, "Architectural power optimization by bus splitting", IEEE Transactions on Computer Aided Design, pp. 408-414, Apr. 2002.

[13]. K. Sayood, "Introduction to Data Compression", Morgan Kaufmann Publishers, 1996.

[14]. H. Mehta, R. M. Owens, M. J. Irwin, "Some Issues in Gray Code Addressing," GLS-VLSI-96, pp. 178-180, Mar. 1996.