A Bus-Encoding Scheme for Crosstalk Elimination in High-Performance Processor Design

Wen-Wen Hsieh, Po-Yuan Chen, Chun-Yao Wang, and TingTing Hwang

***Abstract*—A crosstalk effect leads to increases in delay and power consumption and, in the worst-case scenario, to inaccurate results. With the scale down of technology to deep-submicrometer level, the crosstalk effect between adjacent wires becomes more and more serious, particularly between long on-chip buses. In this paper, we propose a deassembler/assembler technique to eliminate undesirable crosstalk effects on bus transmission. By taking advantage of the prefetch process, where the instruction/data fetch rate is always higher than the instruction/data commit rate, the proposed method incurs almost no penalty in terms of dynamic instruction count. In addition, when the bus width is 128 b, the required number of extra bus wires is only 7 as compared to the 85 extra bus wires needed in the work of Victor and Keutzer.**

***Index Terms*—Architecture, crosstalk, high-performance, instruction/data bus.**

I. Introduction

In deep-submicrometer technology, a coupling capacitance between interconnects is the dominant factor in the total wire capacitance. Coupling capacitance derives from one signal and its neighboring wire switching in different directions. This effect, the crosstalk, leads to serious timing and signal integrity problems that, in the worst case, result in circuit malfunction. Thus, the elimination of crosstalk has become a very important design issue.

In a bus structure, a number of wires are laid in parallel over a long distance. Hence, the crosstalk problem in a bus structure is particularly salient. One category to address this problem is designed for power consumption, and its objective is to minimize the total crosstalk in all wires [1]–[3]. Another category is designed for performance, and its objective is to minimize the maximum crosstalk effect among all wires [4]–[9]. In this paper, we will focus on the second problem, i.e., the elimination of certain data transmission patterns so that the maximum crosstalk effect is minimized.

Thus, a bus-encoding scheme is proposed for wide bus architecture in high-performance processors. By inserting a deassembler and an assembler at the sending and receiving ends of the bus in a prefetch unit, respectively, certain transmission patterns that cause undesirable crosstalk can be eliminated. Our method takes advantage of the prefetch process, where the instruction/data fetch rate is always higher than the instruction/data commit rate [10]. Therefore, in our approach, there is almost no penalty in terms of dynamic instruction count.

The rest of this paper is organized as follows. Section II reviews a related work on crosstalk elimination and reduction. Section III describes the crosstalk model. Section IV presents our proposed bus architecture. Section V shows the experimental results. Finally, Section VI concludes this paper.

Manuscript received October 11, 2006; revised March 23, 2007. This work was supported in part by the National Science Council under Grants NSC-95-2220-E-007-003 and NSC-95-2220-E-007-019, and in part by the Ministry of Economic Affairs under Grant MOEA-96-EC-17-A-01-S1-038. This paper was recommended by Associate Editor V. Narayanan.

The authors are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan, R.O.C. (e-mail: wwhsieh@cs.nthu.edu.tw; pychen@cs.nthu.edu.tw; wcyao@cs.nthu.edu.tw; tingting@cs.nthu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2007.907260

II. Related Work

Approaches in [4], [5], and [7] use bus-encoding methods to minimize the maximum crosstalk. All the proposed encoding data must be crosstalk-free before they are transmitted on a bus. At the receiving end of the bus, a decoder logic decodes the data into the original data. The goal of these methods is to forbid the signal of adjacent wires from switching in different directions at the same time.

In [4], two kinds of encoding methods with memory and without memory are proposed. The experimental results in [4] show that it takes 40- and 46-b wires to encode a 32-b bus by using the memory and memory-free methods, respectively. In another work using the encoding technique [5], the original code is first divided into several groups, and then each group is encoded to be crosstalk-free through a corresponding encoder. Although there is no crosstalk incurred within each individual group, the crosstalk may occur across the group boundaries. In such a case, inverting one of the encoding outputs at a time until group boundaries are crosstalk-free is proposed. The extra wires for inverse information of each group also need to be encoded crosstalk-free in the same way. According to the experimental results shown in [5], a 32-b bus is encoded into 52-b wires. In these approaches, the core concept of encoding to have a crosstalk-free code sequence is to expand the Boolean space so that the codes that cause the crosstalk will never appear in the encoded sequence. Since all pairwise codes have to be taken into consideration (i.e., one code that causes crosstalk with another is not usable in the resultant code space), length b code is expanded to $b + n$, and n (extra wires) becomes very large as such.

Victor and Keutzer [4] also proved theoretically that the maximum number of wires for encoding n-bit bus is $\lfloor \log F_{n+2} \rfloor$, where F_n is the nth number of Fibonacci sequence. These bus-encoding methods become impractical when the number of bus lines becomes large. For example, a 128-b bus will be encoded with 171 wires in theory but with 213 wires in practice. For a high-performance processor like superscalar and very long instruction word (VLIW) architecture, the width of a bus is usually wide (i.e., more than 64 b). Thus, using the methods described previously is not appropriate. In this paper, we propose to use a novel encoding scheme to produce crosstalk-free sequences. The code (all 0s or all 1s) is used to prevent crosstalk sequences. A single code (all 0s or all 1s) is considered instead of all pairs of codes. Therefore, expanding the Boolean space is not necessary.

III. Preliminary

A. Crosstalk Model

A single wire is associated with two types of capacitance. One is the capacitance C_{ground} between the wire and ground, and the other is the coupling capacitance C_{couple} between the wire and its neighboring wires [15]. The coupling capacitance of a wire can be classified into four types—$1C$, $2C$, $3C$, and $4C$—according to the C_{couple} of two wires [5]. Let the crosstalk effect on a single wire (victim) depend on the signal transition of its neighboring wires (aggressors). We use a trituple (w_{i-1}, w_i, w_{i+1}) to represent the wire signal pattern at a certain time, where w_i represents the victim, and w_{i-1} and w_{i+1} are the aggressors. Table I shows the relationship between the crosstalk and the wire signal transition at time T_{t-1} and time T_t, where $(b, \bar{b}) \in \{0, 1\}$, with $\bar{b}$ being the complement of b.

Note that the transmission of a pattern (b, b, b) followed by any other pattern will never cause signals on adjacent wires to switch into a different direction since the signals in pattern (b, b, b) are the same. Take the pattern (0-0-0) as an example. The signal on each wire either switches from 1 to 0 or stays the same at 0, and hence,

TABLE I
BIT PATTERN OF DIFFERENT CROSSTALK TYPES

crosstalk type	time	bit pattern $(w_{i-1},\ w_i,\ w_{i+1})$
1C	T_{t-1}	$(b,b,b)\ (b,b,b)\ (b,\bar{b},\bar{b})\ (\bar{b},\bar{b},b)$
	T_t	$(b,\bar{b},\bar{b})\ (\bar{b},\bar{b},b)\ (b,b,b)\ (b,b,b)$
2C	T_{t-1}	$(b,b,b)\ (\bar{b},b,b)\ (b,b,\bar{b})\ (\bar{b},b,\bar{b})\ (b,b,\bar{b})\ (\bar{b},b,b)$
	T_t	$(b,\bar{b},b)\ (\bar{b},\bar{b},b)\ (b,\bar{b},\bar{b})\ (\bar{b},\bar{b},\bar{b})\ (\bar{b},\bar{b},b)\ (b,\bar{b},\bar{b})$
3C	T_{t-1}	$(b,\bar{b},b)\ (b,\bar{b},b)\ (\bar{b},\bar{b},b)\ (b,\bar{b},\bar{b})$
	T_t	$(b,b,\bar{b})\ (\bar{b},b,b)\ (\bar{b},b,\bar{b})\ (\bar{b},b,\bar{b})$
4C	T_{t-1}	$(b,\bar{b},b)$
	T_t	$(\bar{b},b,\bar{b})$

TABLE II
PERCENTAGE OF UNDESIRABLE PATTERNS

benchmark	bits of instruction	bits of undesirable	ratio(%)
main1024_reduct	76301824	6524398	8.55
main1024_inpsca	40903424	3732558	9.13
board_test	21319424	1951953	9.16
matrix1	3233792	250935	7.76
matrix2	2939392	235224	8.00
main16_reduct	871808	76715	8.80
fir2dim	788736	58966	7.48
main16_inpsca	647040	59157	9.14
startup	530816	36311	6.84
n_com_updates	340352	24551	7.21

the case where adjacent wires switch from 0 to 1 will never happen. Therefore, transmission patterns with all 0s (or all 1s) followed by any other pattern will never incur undesirable crosstalk.

B. Motivation

According to the bit pattern classification analyzed in the previous section, there are more $2C$-type crosstalk patterns than others. Therefore, it is difficult to eliminate all $2C$-type crosstalk patterns. Instead, we focus on eliminating undesirable patterns, as defined in the following section.

Definition 1: An undesirable pattern is a pair of signals on adjacent wires where one signal is rising while the other is falling and vice versa.

By eliminating the undesirable patterns, all $4C$, all $3C$, and part of $2C$ patterns are eliminated. This undesirable pattern is defined as our target pattern for removal in this paper. In the following sections, the terms no undesirable pattern and crosstalk-free pattern are used interchangeably.

In order to verify that removing undesirable patterns is feasible, the percentage of undesirable patterns incurred on instruction bus transmission is profiled. Experiments were performed by using Simplescalar 3.0 [13], where out-of-order four-issue superscalar architecture is used with a DSPstone benchmark to simulate the speculative fetching. Table II shows the profiling result. The column labeled as bits of instruction gives the total bit number of fetched instructions, and the column labeled as bits of undesirable shows the bit number of undesirable patterns. The column labeled as ratio (%) shows the ratio of the bits of undesirable patterns to the total fetched bits. The results show that the ratio of undesirable patterns is very low.

Since the undesirable patterns take only a small portion of the total transmitted data, but increase the length of transmission cycle period, we propose a deassembler and assembler structure on both ends of a bus to eliminate undesirable patterns.

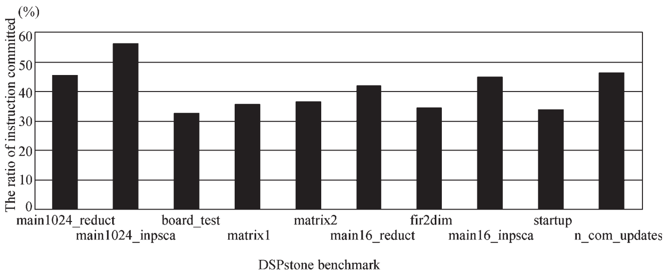

Fig. 1. Ratio of instruction committed.

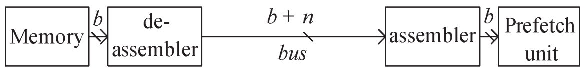

Fig. 2. Basic architecture.

Moreover, we observe that the instruction/data fetch rate is always higher than the instruction/data commit rate in a prefetching process where the instructions/data are fetched into a prefetch unit before they are referenced in order to hide memory-access latency.

In order to verify this observation, the ratio of instruction/data committed to instruction/data fetched is simulated by a superscalar architecture. Fig. 1 shows the ratio of committed instructions to the total fetched instructions for different examples in the benchmark set. The figure shows that the number of committed instructions is only about 30%–50% of the total number of fetched instructions for all examples. In other words, the instruction fetch rate is much higher than the instruction commit rate in bus transmission. We can utilize this characteristic of the prefetching unit to reduce the penalty in terms of dynamic instruction count in our proposed bus architecture.

IV. DEASSEMBLER AND ASSEMBLER TECHNIQUES

A. Basic Scheme

The speed of a processor is always much faster than the speed of accessing data from the memory. To hide memory latencies, a common technique used in high-performance processors is to prefetch instructions or data into buffers before they are used by the processors [10]–[12].

Based on these observations, we develop a bus-encoding scheme for deassembling/assembling data in the memory/prefetch module such that the undesirable patterns are eliminated. Fig. 2 shows our basic architecture. A deassembler is designed to deassemble b-bit data sent by memory into $b+n$-bit crosstalk-free data. The $b+n$-bit crosstalk-free data are then transmitted on the bus. At the receiving end of the bus, an assembler is designed to assemble the $b+n$-bit data into the original b-bit data. The b-bit data are collected in the prefetch unit and sent to the processor on demand. Note that the deassembler is in memory module and that the assembler is in the prefetch module. We are to eliminate the crosstalk on the bus between the memory and prefetch unit in the processor.

The details of our deassembler/assembler bus structure are described as follows. First, a bus structure is partitioned into several channels—$\text{channel}_1, \text{channel}_2, \ldots, \text{channel}_n$—as shown in Fig. 3. The data transmitted on a channel are referred to as a data segment, which are denoted as $\text{data}_{t,i}$, where t is the time stamp, and i is the channel-position index. Each data segment is regarded as a basic data transmission unit. If undesirable patterns occur between two data segments, it is an invalid transition. For example, the transition from

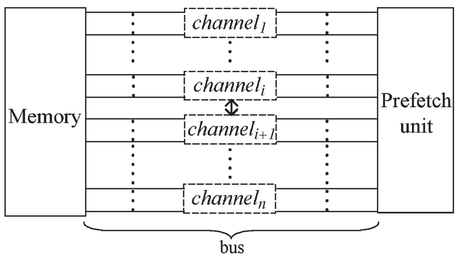

Fig. 3. Partitioned bus structural.

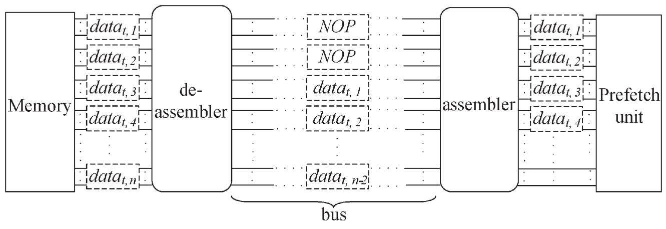

Fig. 4. Sending end and the receiving end.

the data segment data$_{t,i}$ (0-1-0-1) to the data segment data$_{t+1,i}$ (1-0-1-0) is invalid.

In order to remove the undesirable patterns, we propose a deassembling and assembling mechanism shown in Fig. 4. Let data be sent in cycle T_t. In Fig. 4, data$_{t,1}$ represents the data segment to be sent on the first channel position in the current cycle, whereas data$_{t-1,1}$ represents the data segment sent on the first channel position in the previous cycle, which are stored in storage elements in the deassembler. Before sending data, the deassembler checks for undesirable patterns occurring between data$_{t,i}$ and data$_{t-1,i}$. If no undesirable pattern is found, then data$_{t,i}$ are transmitted on the channel$_i$. Otherwise, the data$_{t,i}$ is shifted to the next channel position channel$_{i+1}$, and an all-0 (or all-1) pattern called an NOP segment is inserted onto the channel$_i$ in order to eliminate undesirable patterns. Note that an all-0 (or all-1) pattern will not incur undesirable patterns with any other patterns.

Once data$_{t,i}$ is shifted to channel$_{i+1}$, it is checked with data$_{t-1,i+1}$ to see if any crosstalk occurs between them. The checking continues until data$_{t,i}$ finds a position channel$_j$, where data$_{t,i}$ incurs no crosstalk with data$_{t-1,j}$ or when it reaches the last channel of the bus. Data segments data$_{t,i}$ that are unable to be sent during the current cycle T_t due to the NOP-segment insertion are shifted to the next cycle T_{t+1}. For example, in Fig. 4, assume that data$_{t,1}$ incurs undesirable patterns with data$_{t-1,1}$ and data$_{t-1,2}$. Then, the data$_{t,1}$ is shifted to the two channel positions and will be sent at position channel$_3$. Since the data segments are shifted to the two channel positions, data$_{t,n-1}$ and data$_{t,n}$ would be sent in the next transmission cycle T_{t+1}.

As to the assembler, it removes all the inserted NOP segments and packs the valid data segments, as shown in Fig. 4. After the packing, the assembler informs the processor of the number of completed instructions at the current cycle. Data segments that cannot be packed into a complete instruction will be stored in a buffer queue to wait for the next assembling processing.

Note that the worst-case scenario of transmission time happens when the undesirable patterns occur between data$_{t,1}$ and every data segment transmitted at cycle T_{t-1}. In this case, the bus is filled with all NOP segments at current cycle transmission. However, since the NOP segments do not result in a crosstalk with any other data patterns in the next transmission cycle, all data segments can be sent without incurring any undesirable patterns. Therefore, the worst case is to double the transmission cycles, i.e., one cycle for data segments transmission and one cycle for NOP segments alternately.

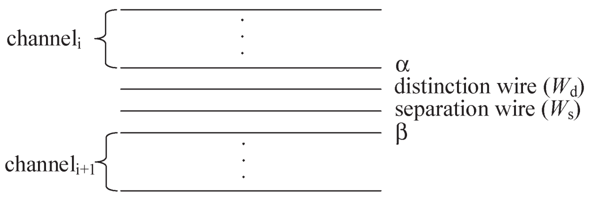

Fig. 5. Distinction wire insertion next to the separation wire.

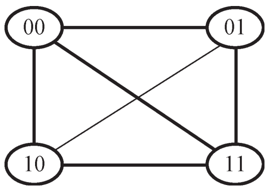

Fig. 6. Two-bit-pattern transition.

B. Insertion of Separation and Distinction Wires

After applying the crosstalk detection and NOP-segment-insertion mechanism described in the previous section, invalid transitions on each channel can be avoided. However, the invalid transition may occur across the boundary of two adjacent channels. Fig. 3 shows an example of the crosstalk occurring between channel$_i$ and channel$_{i+1}$.

In order to prevent the crosstalk from occurring across two adjacent channels, we propose to insert shielding wires called separation wires between every pair of channels, as shown in Fig. 5. A shielding wire set to 0 (or 1) working as a stable ground (or Vdd) wire is sufficient to ensure that a crosstalk never occurs between two adjacent wires.

Next, in order to address the problem of determining whether an all-0 pattern (or all-1 pattern) transmitted on a channel is a data segment or an NOP segment, an extra bit denoted as a distinction wire is required for each channel. For n channels, n extra distinction wires are required. If we place n wires together, a crosstalk may occur on these n wires, and extra mechanism is needed to avoid a crosstalk among these n wires. Consequently, our design is to put each distinction wire next to a separation wire, as shown in Fig. 5. We prove that, for this placement of distinction wires, no extra crosstalk-avoidance mechanism is required.

Before we give the proof, we define the term crosstalk-free cyclic.

Definition 2: A crosstalk-free cyclic is a set of bit patterns where any pair of the patterns in the set does not incur undesirable patterns.

Fig. 6 shows an example of a 2-b pattern graph in which each node represents a 2-b pattern and an edge represents a transition between two nodes. A dark line denotes that the transition between the two nodes is not an undesirable pattern, whereas a thin line denotes that the transition is an undesirable pattern. In this figure, we can see that patterns 00, 01, and 11 form a crosstalk-free cyclic and that patterns 00, 10, and 11 also form a crosstalk-free cyclic.

If we can assure that the data transmitted on wires form a crosstalk-free cyclic, then there will be no undesirable patterns transmitted on the wires. Now, we prove that the placement of distinction wires and separation wires, as shown in Fig. 5, will incur no undesirable patterns.

Theorem: Let two wires W_d and W_s be inserted between two channels channel$_i$ and channel$_{i+1}$, where W_d and W_s are used for distinction and shielding purposes, respectively. The last wire of channel$_i$ denoted as α is adjacent to W_d, and the first wire of channel$_{i+1}$ denoted as β is adjacent to W_s. With this placement of distinction and shielding wires, an encoding for W_d can always be found such that there will be no undesirable patterns transmitted on the four wires.

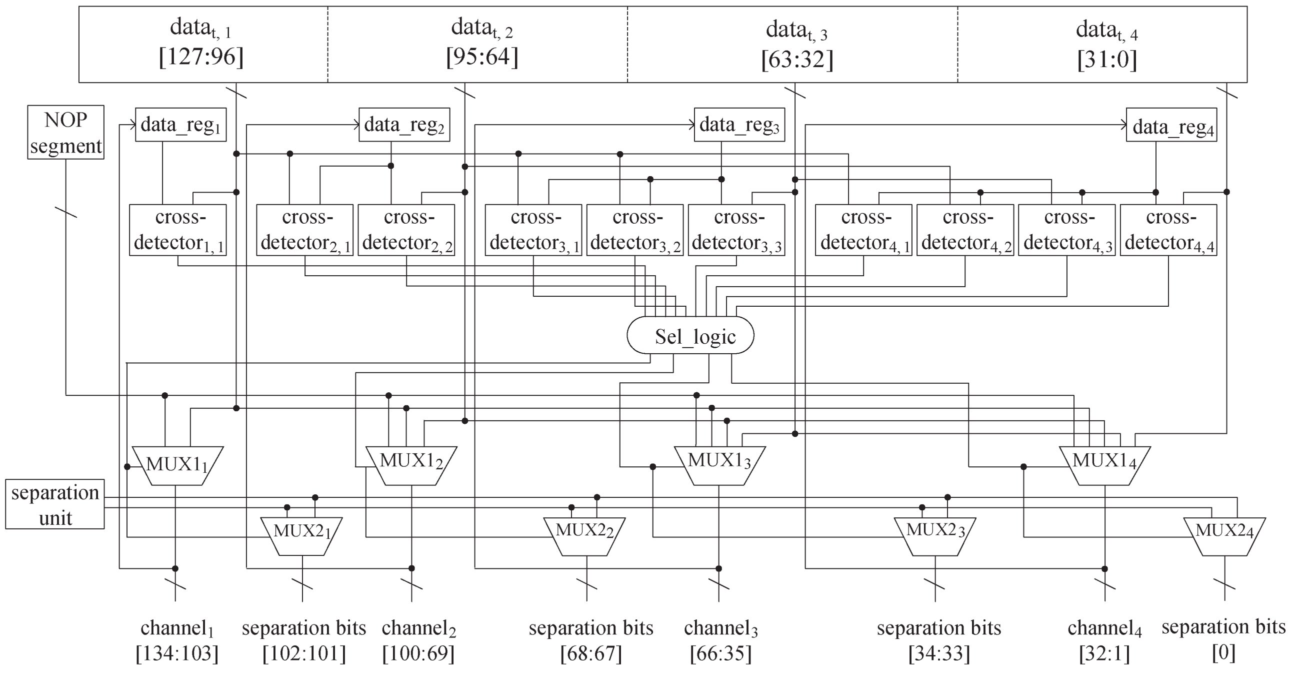

Fig. 7. Deassembler architecture.

Proof: We first consider wires W_d, W_s, and β. Since W_s is a shielding wire, there will be no undesirable patterns transmitted between wires W_d and W_s or between wires W_s and β.

Now, we consider wires α and W_d. Let NOP segment be an all-0 pattern. Then, we set $W_\mathrm{d} = 0$ to denote NOP segment and $W_\mathrm{d} = 1$ to denote data segment. With this assignment, all possible patterns transmitted on α and W_d are 00 (α is the last bit of NOP segment), 01 (α is the last bit of data segment), and 11 (α is the last bit of data segment). Since these three patterns form a crosstalk-free cyclic, there will be no undesirable patterns transmitted on α and W_d. Similarly, if an NOP segment is an all-1 pattern, we can set $W_\mathrm{d} = 1$ to denote NOP segment and $W_\mathrm{d} = 0$ to denote data segment. Then, all possible patterns transmitted on α and W_d are 11, 00, and 10. Since these three patterns also form a crosstalk-free cyclic, there will be no undesirable patterns transmitted on α and W_d. ■

C. Deassembler and Assembler Architectures

In order to check if a crosstalk occurs between a data segment to be sent at current cycle and a data segment already sent at a previous cycle in parallel rather than in sequential, we need a parallel checking architecture. In this section, we describe our deassembler and assembler architectures. The deassembler architecture is shown in Fig. 7. In this example, the width of the whole bus is 128 b, and the width of each channel is set to 32 b. Hence, the bits from 127 to 96 are grouped as channel$_1$, the bits from 95 to 64 are grouped as channel$_2, \ldots$, etc., and the total number of channels is four.

To detect if a crosstalk occurs between the current data segment data$_{t,i}$ and the data sent in channel$_j$ during a previous cycle, two logic elements named data_reg and cross_detector are designed. For each channel$_i$, there is one data_reg$_i$ and $|i|$ cross_detector$_{i,j}$, for j is from 1 to i. The data_reg$_i$ is used to store the data segment sent on channel$_i$ during a previous cycle. The cross_detector$_{i,j}$, where j is from 1 to i, is a combinational logic used to check if data_reg$_i$ and data$_{t,j}$ induce undesirable patterns. In other words, data_reg$_i$ is checked with all data segments data$_{t,j}$ to be sent, for j is from 1 to i, as shown in Fig. 7.

Next, all the output signals of the cross_detector$_{i,j}$ are sent to a logic element named Sel_logic$_i$. With inputs from all cross_detectors, Sel_logic$_i$ will decide which data segment is to be sent on channel$_i$. Then, the output of Sel_logic$_i$ is passed to the first-level multiplexor MUX1$_i$, where the inputs to MUX1$_i$ are data$_{i,j}$, for j is from 1 to i, and NOP segment. This multiplexor is used to select the data segment or NOP segment to be sent. Finally, the outputs of cross_detector$_{i,j}$ are also sent to the second-level multiplexor MUX2$_i$, which is used to select the distinction wires.

At the receiving end of the bus, an assembler is designed to remove the NOP segments. The input of the assembler is a set of data segments interleaved with distinction wires. A logic element is constructed to determine whether the incoming data are data segment or NOP segment. The inputs to this logic element include the distinction wires that record the information to distinguish a data segment from an NOP segment. The output of this logic element is the number of channel positions to be left shifted for each data segment. After the NOP segments are removed, the real data segments are pushed into the prefetch unit.

V. Experimental Results

A. Experiments on Delay

The first experiment studies the performance improvement that can be obtained by eliminating the undesirable patterns. Since two additional logic circuits are inserted, the performance improvement includes the wire delay reduced by eliminating the undesirable patterns and the extra circuit delays caused by deassembler/assembler insertion. Since the crosstalk effect is only noticeable in deep-submicrometer technology, we simulate the wire transition time in 90- and 65-nm technologies by using Spice in [16].

The simulation of the deassembler and assembler delays in 65- and 90-nm technologies is performed as follows. Due to the unavailability of 65- and 90-nm cell libraries, two logic circuits are first designed using Verilog and are synthesized by the Synopsys Design Compiler with the Taiwan Semiconductor Manufacturing Company (TSMC) 0.13-μm cell library. Then, the critical path is extracted and simulated by using Spice in [16] with scale-sized transistors.

The case of 32 b/channel is taken as an example for analysis. Table III shows the simulation results in picoseconds. The columns

TABLE III
TIMING ANALYSIS OF WIRE AND THE DEASSEMBLER/ASSEMBLER

		90nm		65nm	
		3mm	5mm	3mm	5mm
w/o undesirable pattern		217.19	506.45	380.77	685.49
4C		589.72	1321.40	943.36	1992.60
deassembler + assember	ours	363.91		437.82	
	[4]	318.62		355.32	
total	original	589.72	1321.40	943.36	1992.60
	ours	581.10	870.36	818.59	1123.31
	[4]	535.81	825.07	736.09	1040.81
ratio (%)	ours	98.54	65.87	86.77	56.37
	[4]	90.86	62.44	78.03	52.23

labeled as 3 and 5 mm are the wire lengths. The row labeled as w/o undesirable pattern means the wire delay without undesirable patterns. Since eliminating the undesirable patterns can only remove part of the $2C$ crosstalk patterns, the wire delay of the $2C$ crosstalk is reported as such. The row labeled as deassembler + assembler reports the sum of the critical path delays for the deassembler and assembler. The row labeled as total is the total delay of summing the wire delay without undesirable patterns, the deassembler delay, and the assember delay. The last column reports the delay ratio of our design and of the original design ($4C$ wire delay) in [4].

The table shows that the wire delay with $4C$ crosstalk becomes increasingly serious as the process technology is scaled down and as the bus length increases. For example, the wire delay with the $4C$ crosstalk is about twice as that with only the $2C$ crosstalk (e.g., 589.72 ps by $4C$ and 217.79 ps by $2C$ when a bus length is 3 mm in the 90-nm technology). In addition, the extra delay caused by the deassembler and assembler is less significant when the bus length is increased. Summing the delay time for bus transmission, deassembler, and assembler, the improvement rates are about 34% in the 90-nm technology and 42% in the 65-nm technology when the bus length is 5 mm. Compared with the method used in [4], our method performs less efficient when a short bus is used. However, when a long bus is used and technology scales down, our method is comparable with the method used in [4].

The second experiment on delay studies the number of extra cycles required to execute a program. The ten largest examples in DSPstone benchmark set [17] are selected as our test cases. We use the sim-outorder simulator from Simplescalar 3.0 [13] incorporated with our deassembler and assembler architectures to simulate the out-of-order four-issue superscalar architecture without caches. In the simulation, each instruction is 32 b long, and four instructions are issued in parallel (the total bus width is 128 b). Two different channel sizes of 16 and 32 b/channel are simulated. Note that the occurrences of $2C$, $3C$, and $4C$ sequences are determined by the program. In this experiment, the ratio of undesirable patterns is shown in Table II.

Table IV shows the simulation results. The column labeled as inst is the result for instruction bus. The columns labeled as TCC and pen are the total cycle count of the original circuit and the extra number of cycles (penalty) using our architecture, respectively. In the worst case, the cycle-count overhead is only about 0.29% (n_com_updates when channel size is 32 b).

A similar experiment is performed on data bus. In this experiment, the data prefetch is implemented in the dispatch stage. We assume that there are four memory ports—two for read ports and two for write ports. Each data bus is 64 b long, and the total data bus width is 128 b. The experimental result is also shown in Table IV where the column labeled as data is the result for data bus. It is shown in the table that there is no cycle penalty. The main reason is that the utility rate of the data bus is very low. On average, the memory reference instructions (load/store) are only about 40%–50% of the total instructions. Hence, the NOP segments are inserted without overhead.

TABLE IV
CYCLE-COUNT OVERHEAD FOR CHANNEL SIZES 16 AND 32 B

benchmark	TCC	16			32		
		inst		data	inst		data
		pen	(%)	pen	pen	(%)	pen
main1024_reduct	2017283	10	0.0005	0	2117	0.10494	0
main1024_inpsca	1223523	8	0.0007	0	1046	0.08549	0
board_test	441463	87	0.0197	0	574	0.13002	0
matrix1	73822	9	0.0122	0	22	0.02980	0
matrix2	68122	13	0.0191	0	25	0.03670	0
main16_reduct	20610	14	0.0679	0	52	0.25230	0
fir2dim	16472	8	0.0486	0	23	0.13963	0
main16_inpsca	15586	15	0.0962	0	40	0.25664	0
startup	10561	8	0.0758	0	21	0.19884	0
n_com_updates	7956	10	0.1257	0	23	0.28909	0
average			0.04663			0.15235	

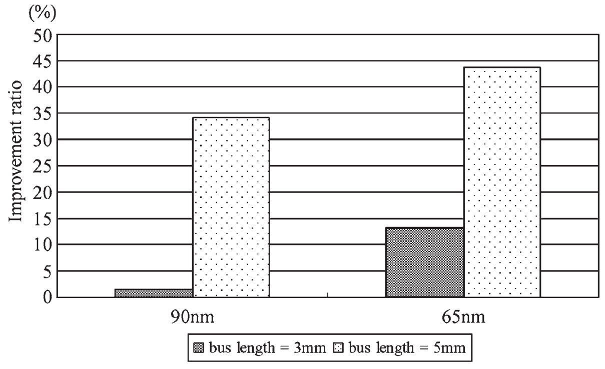

Fig. 8. Improvement ratio for channel size 32 b.

The third experiment on delay studies the improvement rate of the total transmission time for different technologies in the case of 128-b bus width with 32 b/channel. The improvement on the total transmission-time ratio is calculated as improvement_ratio = (new_tcc × rate/orig_tcc) × 100%, where orig_tcc and new_tcc are the total transmission cycle count of the ten programs on the original circuit and the new circuit, respectively, and the rate is the transmission-length reduction rate for different technologies. Fig. 8 shows that the improvement ratios of the total transmission time are about 34% for the 90-nm technology and about 43% for the 65-nm technology when the bus length is 5 mm. Since the cycle penalty is very little, the improvement rate of the total transmission time is close to the cycle-time reduction rate.

B. Experiments on Power

In this experiment, the channel size is set to 32 b, and the power consumption on bus is estimated by a toggle count. Both coupling [18] and transition toggle counts are computed. The coupling toggle count represents the total number of undesirable patterns transmitted on the bus, whereas the transition toggle count represents the total number of bit switching. Table V gives the experimental results. The columns labeled as ctoggle and ttoggle represent the coupling and the transition toggle counts, respectively. The columns labeled as ratio show the ratio of the transition toggle counts to the total toggle counts of the original circuit. Since we removed all the undesirable patterns, there are no coupling toggle counts in our method, as is in [4]. In average, our method reduces the total toggle count to 70% of that in the original circuit, and it is comparable with the total toggle count in [4].

TABLE V
COUPLING TRANSITION TOGGLE COUNTS

benchmark	original		ours	[4]
	ctoggle	ttoggle	ratio(%)	ratio(%)
main1024_reduct	3674752	39934397	58	62
main1024_inpsca	2136696	23162146	51	56
board_test	1077401	9795597	65	68
matrix1	144510	1479133	61	68
matrix2	135050	1311740	89	67
main16_reduct	43450	569921	74	51
fir2dim	33646	345953	92	73
main16_inpsca	43450	326209	58	64
startup	20300	244226	93	74
n_com_updates	13699	157078	58	71
average			70	65

TABLE VI
AREA OVERHEAD FOR 128-B BUS WIDTH

area type		ours		[4]
		channel size		
		16	32	
deassembler/ encoder	gate count	7445	3936	885
	area(μm)	16320.48	8826.45	2359.39
	storage bit	128	128	0
assembler/ decoder	gate count	941	597	1402
	area(μm)	3279.37	1643.01	3381.22
# extra wires (bit)		15	7	85

TABLE VII
NUMBER OF EXTRA WIRES

bus width	ours				[4]	[4]
	channel size					
	4	8	16	32	theoretical	practical
32	15	7	3	1	14	21
64	31	15	7	3	28	45
128	63	31	15	7	59	85

C. Experiment on Area Overhead

Finally, the last set of experiments compares the area overhead of our proposed method and the method in [4] with memory-free approach (practical case) where the minimum number of extra wires inserted is proved. The area overhead includes the area of the deassembler/assembler and the extra wires required for separation flags. Table VI gives the experimental results. The gate count is obtained by synthesizing the circuits using only an NOR gate and an inverter. The result shows that the deassembler in our design takes more area than the encoder in [4]. In addition, storage bits are needed in our approach because the data segments or the NOP segments transmitted during the previous cycle must be stored. As to the required extra wires, the numbers of extra wires used in our method are only 7 and 15 when the channel sizes are 32 and 16 b, respectively, as compared to the 85 extra wires needed for the practical cases proposed in [4].

For the number of extra wires inserted, Table VII shows the comparisons of our results to those in [4] with memory-free approach. Four cases for different channel sizes using our method (4, 8, 16, and 32 b/channel) and two cases presented (theoretical and practical cases) in [4] are shown. The results show that, as the bus width gets wider, the effectiveness of our approach increases. For example, when the bus width is 128 b and the channel size is 32 b, the number of extra wires used in our method is only seven as compared to 59 and 85 extra wires needed for the theoretical and practical cases proposed in [4], respectively. It can be seen that our method is particularly suitable for a high-performance processor where the wide bus structure is used.

VI. CONCLUSION

In this paper, we have proposed a new bus structure to eliminate undesirable patterns which cause a crosstalk effect during data transmission. By inserting a deassembler and assembler at the sending and receiving ends of the bus, respectively, certain transmission patterns that cause undesirable crosstalk can be eliminated. By taking advantage of the prefetch process where the instruction/data fetch rate is always higher than the instruction/data commit rate in a high-performance processor, the proposed method incurs almost no penalty in terms of dynamic instruction count. According to the experimental results, our method achieved about 43% performance improvement rate at the expense of a small number of wire increase as compared with the original design in 65-nm technology.

REFERENCES

[1] S. P. Khatri, A. Mehrotra, R. K. Brayton, A. S. Vincentelli, and R. H. J. M. Otten, "A novel VLSI layout fabric for deep sub-micron application," in *Proc. Des. Autom. Conf.*, Jun. 1999, pp. 491–496.

[2] R. Arunachalam, E. Acar, and S. R. Nassif, "Optimal shielding/spacing metrics for low power design," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Feb. 2003, pp. 167–172.

[3] S. K. Wong and C. Y. Tsui, "Re-configurable bus encoding scheme for reducing power consumption of the cross coupling capacitance for deep sub-micron instruction bus," in *Proc. DATE*, Nov. 2004, vol. 1, pp. 130–135.

[4] B. Victor and K. Keutzer, "Bus encoding to prevent crosstalk delay," in *Proc. ICCAD*, Nov. 2001, pp. 57–63.

[5] C. Duan, A. Tirumala, and S. P. Khatri, "Analysis and avoidance of crosstalk in on-chip buses," in *Proc. Hot Interconnects*, Aug. 2001, pp. 133–138.

[6] L. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "A crosstalk aware interconnect with variable cycle transmission," in *Proc. DATE*, Feb. 2004, vol. 1, pp. 102–107.

[7] C. Duan and S.P. Khatri, "Exploiting crosstalk to speed up on-chip buses," in *Proc. DATE*, Feb. 2004, pp. 778–783.

[8] W. A. Kuo, Y. L. Chiang, T. Hwang, and A. C. H. Wu, "Performance-driven crosstalk elimination at post-compiler level," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 3, pp. 564–573, Mar. 2007.

[9] T. K. Tien, S. C. Chang, and T. K. Tsai, "Crosstalk alleviation for dynamic PLAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1416–1424, Dec. 2002.

[10] S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 174–199, Jun. 2000.

[11] V. Srinivasan, E. S. Davidson, and G. S. Tyson, "A prefetch taxonomy," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 126–140, Feb. 2004.

[12] X. Zhuang and H. H. S. Lee, "Reducing cache pollution via dynamic data prefetch filtering," *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 18–31, Jan. 2007.

[13] [Online]. Available: http://www.simplescalar.com/

[14] [Online]. Available: http://www-device.eecs.berkeley.edu/ ptm

[15] P. P. Sotiriadis and A. Chandrakasan, "Reducing bus delay in sub-micron technology using coding," in *Proc. ASPDAC*, Jan./Feb. 2001, pp. 109–114.

[16] L. Nagel, "Spice: A computer program to simulate computer circuits," *UC Berkeley UCBERL Memo M520*, May 1995.

[17] V. Zivojnovic, J. M. Velarde, C. Schlager, and H. Meyr, "DSPstone: A DSP-oriented benchmarking methodology," in *Proc. Int. Conf. Signal Process. Appl. Technol.*, Oct. 1994, pp. 715–720.

[18] J. Liu, K. Sundaresan, and N. R. Mahapatra, "Efficient encoding for address buses with temporal redundancy for simultaneous area and energy reduction," in *Proc. 16th ACM Great Lakes Symp. VLSI*, Apr.–May 2006, pp. 111–114.