Re-configurable Bus Encoding Scheme for Reducing Power Consumption of the Cross Coupling Capacitance for Deep Sub-micron Instruction Bus

Siu-Kei Wong, Chi-Ying Tsui
Department of Electrical and Electronic Engineering
The Hong Kong University of Science and Technology

Abstract

In very deep sub-micron designs, cross coupling capacitances become the dominant factor of the total bus loading and have a significant impact on the power consumption. In this paper, we propose two re-configurable bus encoding schemes, which are based on the correlation among the bit lines, to reduce the power consumption at the cross coupling capacitances of the instruction buses. The instruction is encoded by flipping and reordering the bit lines during compilation time to reduce the total switching capacitances. A crossbar is used to map back the data to the original instruction code before sending to the instruction decoder. The reordering can be re-configured during run-time by using different configurations in the crossbar. We propose two types of re-configuration, static and dynamic. Static coding uses a fix flipping and re-configuring pattern after the corresponding program is compiled. Dynamic coding allows different re-configuring patterns during program execution. Experimental results show that by using the proposed schemes, significant energy reduction, 17-23%, can be achieved. Comparisons with existing bit lines reordering encoding scheme have also been made and on average more than 15% reduction can be obtained using our method.

1. Introduction

In very deep sub-micron era, energy consumption at the cross coupling and stand-alone capacitances of the long buses become the dominant factors on the overall power consumption of the system-on-chip designs. Previous works were done to minimize the stand-alone switching capacitances. The Bus-Invert code [1] and its modification [2] have been proposed, where a great reduction in power consumption can be achieved. The Gray code [3] and T0 code [4] have been designed to reduce the power consumptions at the address buses. Recently research works [5] [6] have been done on reducing the energy consumption at the cross coupling capacitances. Switching statistics of the buses, which are obtained by tracing the data sequences of a number of real-world applications, are exploited for the encoding. The major advantage of statistic based encoding scheme is that further minimization of power can always be achieved. Macchiarulo *et al.* [7] have proposed the idea of rearranging the physical ordering of the bit lines based on the statistic of the data sequences to reduce the cross coupling switching activity of the address bus. However, these methods do not work efficiently on the instruction bus because the correlations of the instruction bits are usually lower than that of the address bus. Another disadvantage is that the reordering pattern is fixed for all different program applications because it is implemented directly during the physical design. In this work, we propose a novel method to reduce the power consumption of the instruction bus by dynamically re-configuring the order of the bit of the bus. It enhances the approach proposed in [7] in the following ways:

- Select a set of bit lines to flip before rearranging the bit lines order
- For each program application, there is a dedicate set of flipping and reordering pattern which is determined during compilation time. Re-configurable crossbar is used as the hardware decoder to get back the original instruction code during runtime.
- Instead of having a fixed reordering pattern for an application, we also develop a dynamic method in which multiple reordering patterns are allowed during the program execution.

Inverting a set of bit lines before rearranging the order increases the flexibility and enhances the efficiency of the bit lines reordering. Therefore, we propose a two-phase algorithm to generate the optimal flipping and reordering pattern. Phase one is responsible for flipping the bit lines. An algorithm is proposed to find the optimum set of bit lines to be inverted. Phase two rearranges the physical

*This work was supported in part by Hong Kong RGC CERG under Grant HKUST6214/03E and HKUST HIA02/03.EG03

order of the bit lines so that the cross coupling switching is minimized. A heuristic algorithm is proposed to find an optimal order of the bit lines. These two optimization steps are then carried out iteratively until no improvement is obtained.

The static encoding we proposed includes the whole two-phase algorithm and the encoding is done during the compilation time. The dynamic encoding we proposed includes only the reordering phase since we want to reduce the overhead required for storing the flipping pattern. By doing so, a smaller look-up-table is needed for the decoding during execution.

In next section, we will first describe the bus model and the architecture of our targeted system. Then, the overview of the basic encoding scheme will be given in section 3. The static and dynamic encoding scheme will be described in section 4 and 5, respectively. In section 6, experimental results will be presented. Finally, conclusions will be given in section 7.

2. Bus model and embedded system model

In the bus model, Cc is the cross coupling capacitances between two adjacent wires and Cs is the stand-alone capacitances between the wire and the substrate. The total bus capacitances is equal to the sum of the cross coupling capacitances and the stand-alone capacitances. The total energy consumption of the bus is thus equal to:

$$Energy = (Y \cdot Cc + X \cdot Cs) \cdot Vdd^2, \quad (1)$$

where Y is the total number of cross coupling switching for the data transfer and X is the corresponding total number of bit lines switching. X can be calculated by summing up X_{ij}, which represents the transition of bit line i from cycle j to cycle j+1. It will be equal to one if there is a 0 to 1 transition. Otherwise, X_{ij} will be equal to zero. Thus, X is given by $X = \sum_{i=0}^{m-1} \sum_{j=1}^{n-1} X_{ij}$, where n is the total number of clock cycles needed for the data transfer and m is the bit-width of the bus.

Y can be calculated by the summation of $Y_{(i,i+1),j}$, which represents the cross coupling transition from cycle j to cycle j+1 between bit line i and its adjacent line i+1. Table 1 shows the values of $Y_{(i,i+1)}$ in different cases, which is the same bus model used in [7]. Therefore, Y is given by $Y = \sum_{i=0}^{m-2} \sum_{j=1}^{n-1} Y_{(i,i+1),j}$.

Table 1. The value of $Y_{(i,i+1)}$

Y(i,i+1)	bit line i at time j-> j+1				
bit line i+1 at time j-> j+1		0->0	0->1	1->0	1->1
	0->0	0	1	0	0
	0->1	1	0	2	0
	1->0	0	2	0	1
	1->1	0	0	1	0

The architecture of the embedded system that we target is shown in Figure 1. It includes three main components, the main program memory, the instruction cache and the CPU core.

We assume that the main program memory is organized into n blocks and there are m blocks inside the instruction cache. The block sizes of the main memory and the cache are the same. Direct mapping is used as the block allocation scheme from the memory to the cache although other mapping schemes can also be used.

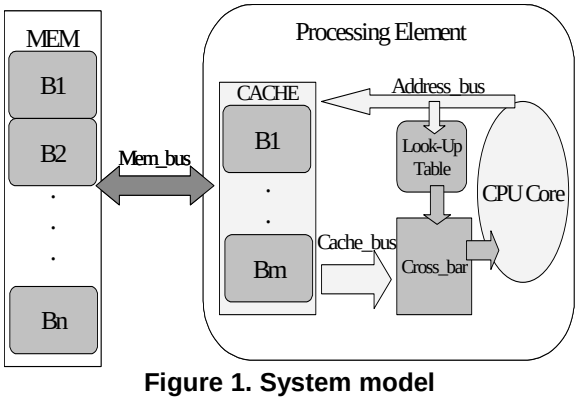

Figure 1. System model

3. Overview of the encoding scheme

The main idea of the proposed encoding schemes is to reduce the instruction bus energy during program execution, by encoding the instructions during compilation time, which is done off-line. The following gives an overview on how the static and dynamic encoding schemes work:

- First the compiler generates the statistical information of the program during compilation
- The instructions will be encoded during compilation by using one of the following schemes:
  a) For static encoding scheme, the instructions are encoded by flipping and reordering the bit lines. Only one flipping and reordering configuration is generated for each program.
  b) For dynamic encoding scheme, the instructions are encoded by only reordering the bit lines. Multiple reordering configurations are generated for each program.
- The decoding information for each configuration are attached as the header of the program.
- The program is then loaded into the main program memory.
- When a program is called, the decoding information stored in the header are loaded from the main memory to the CPU core and stored inside a look-up table before execution.

- During execution, instructions will be decoded by getting the configuration information from the look-up table and re-configuring the crossbar to get back the original order before sending to the instruction decoder. This is illustrated in Figure 1.

Using the proposed schemes, we can reduce the cross coupling switching in both the main program memory and the instruction cache buses during program execution.

4. Static bus encoding scheme

In static encoding, only one configuration will be generated for each program. The configuration will not change during the program execution. A two-phase algorithm is used to generate the configuration based on the switching statistics obtained from the statistical trace of the program. In the following, we will describe the two-phase algorithm in more details. The energy overhead required for the static encoding scheme will also be described.

4.1 Phase one encoding

The main idea of the phase one encoding is to invert a set of bit lines such that the total cross coupling switching is reduced. If the adjacent bit lines are always switching in opposite direction, the cross coupling switching is very large. We can achieve a great reduction by simply inverting either one of the bit lines. However, additional cross coupling switching will be created if both bit lines are switching in a similar fashion and one of them is flipped. Therefore, we need to find the optimal set of bit lines to flip in order to achieve the optimal result.

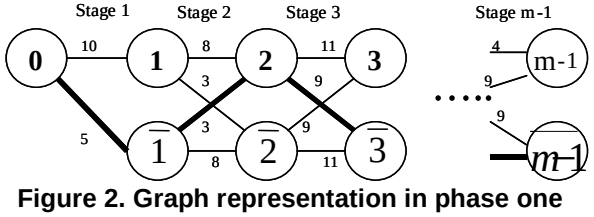

Figure 2. Graph representation in phase one

Problem formulation This problem can be formulated as a graph optimization problem. The graph representation for m bit lines is shown on Figure 2. Each vertex represents a bit line. The upper vertices represent the original bit lines. The lower vertices represent the inverted state of the bit lines. The edges represent the number of cross coupling switching between two bit lines.

Bit lines 0 and m-1, are the boundaries of the bus and we can treat them as the start and end points of the graph. We need to traverse from the start point to the end point and only one of the vertices in each stage is visited in the traversal path. Thus, the optimization problem is to find the shortest path (the highlighted path) from the start point to the end point.

One of the important characteristic of cross coupling capacitances is that the cross coupling switching between two inverted bit lines is equal to that between two non-inverted bit lines as shown in Figure 2. The top and bottom edges are always the same in each stage. If only one bit line among the two is inverted, the coupling switching is the same no matter which one is inverted. This is shown by the fact that the values of cross edges are always the same in each stage. Based on this characteristic, we solve this shortest path problem by using a greedy algorithm. The algorithm starts at vertex 0 and the edge with the lowest cost is always selected.

4.2 Phase two encoding

Phase two reduces the cross coupling switching by rearranging the order of the bit lines. We formulate this reordering problem as a graph optimization problem. The graph is a completely-connected undirected graph. Figure 3 shows a simple example of the graph. The vertices represent the bit lines. The weight on the edges represents the cross coupling switching between two bit lines. Our objective is to find a path such that every vertex is visited once and the total weight is minimized (the highlighted path). This problem is a variant of the well-known Traveling Salesman Problem (TSP) [8]. Here we employ a similar algorithm used in [7] to solve this problem.

The optimization problems in phase one and phase two are actually dependent. By doing phase one and phase two iteratively, further optimization in energy consumption can be achieved. The iteration stops until no improvement can be obtained.

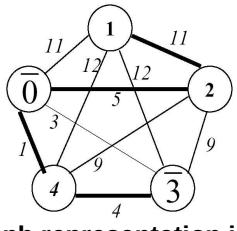

Figure 3. Graph representation in phase two

4.3 Required overhead

To implement the static encoding, two types of overhead are needed. The first one is the extra bus transition activity for sending the decoding information from the main memory to the CPU. If the bus width is 32 bits, the decoding information contains $32\log_2 32+32$ bits and 6 cycles are needed to transfer this information. The first $32\log_2 32$ bits are the control bits for the 32-bits mux-based crossbar. The remaining 32 bits are used for inverting back the flipped bit lines. In static encoding, only one configuration is generated so that only one set of 6 cycles is sent before the execution of the program. Thus, the extra bus power and impact on the execution

performance are small. The other overhead is the additional hardware for decoding the instructions before entering the CPU. A crossbar for rearranging back to the original order, a set of inverters for inverting back the bit lines and a set of registers for storing the decoding information are needed. The power consumption of these overheads will be analyzed and presented in the experimental results in section 6.

5. Dynamic bus encoding scheme

5.1 Encoding strategy

In the dynamic encoding scheme, multiple permutations are generated for a single program. However, overhead will become significant on the overall power consumption. To reduce the overhead, two strategies are proposed. The first one is that only the phase two of the two-phase algorithm, which is the rearranging of bit lines order, will be adopted so that the decoding information can be reduced. The second strategy is that the number of permutations generated is based on the number of blocks in the cache. As shown in Figure 1, only m permutations will be generated. The encoding scheme will first locate all the blocks in the main memory, which will be mapped to the same cache block by using direct mapping method. For example, memory blocks B_1, B_{m+1} and B_{2m+1} will be mapped to cache block B_1 by using direct mapping. The encoding scheme will analyze these memory blocks and generate a permutation by running the phase-two algorithm mentioned in section 4.2. These memory blocks will have the same permutation. The instructions inside these memory blocks are rearranged with the bit lines order according to this permutation and then loaded into the memory.

To further enhance the performance of the encoding scheme, the permutation will bias to the memory blocks with higher miss rate and frequent execution, i.e. we put more weight on the switching statistics on these memory blocks. For example, assume B_1, B_{m+1} and B_{2m+1} share the same permutation and we know B_1 will always be loaded to the cache, then the permutation will bias to B_1. In other words, the permutation is suited better for B_1. Miss rate and execution frequency statistics of the memory block are obtained by analyzing the dynamic trace during the benchmarking of the program.

5.2 Decoding strategy

Before execution, all the decoding information (attached in the header file) will be sent from the main program memory to the CPU and stored in a look-up table. Because there are m permutations according to the number of blocks in cache, m set of different decoding information will be stored.

During program execution, when the CPU fetches instructions from a particular cache block, the same instruction cache address will also be sent to the look-up table to get the corresponding decoding information. This information is sent to reconfigure the crossbar for rearranging the instruction bits before sending to the instruction decoder (Figure 1).

5.3 Overhead required

The overhead in the dynamic encoding scheme is higher than that in the static encoding scheme. There are two main types of overhead, the extra bus power for sending m sets of decoding information and additional hardware for decoding. Here, one set of decoding information contains $32\log_2 32$ bits. Thus, we need 5m cycles for sending the decoding information before the program execution. In addition, a look-up table is needed for storing this decoding information. The number of entries in the look-up table is equal to the number of blocks in the cache. A crossbar is also needed for rearranging the instruction bit lines back to the original order. No extra inverters are needed, as phase one algorithm is not used. The power consumption and timing overhead of these extra hardware will be analyzed and discussed in the next section.

The decoding overhead has a significant impact on the energy consumptions of the dynamic encoding scheme. To reduce the required overhead, we introduce a limited reordering scheme. The limited reordering divides the 32 bits into n m-bits groups, where m is equal to 32/n and the reordering is done within the m-bits group only. The reduction in cross-coupling switching is decreased with the size of the group, but the size of the look-up table and the crossbar also become smaller and the energy consumptions of the overhead are reduced. So there is a trade-off between the reduction in cross-coupling switching and the decoding overhead. For a specific bus length, there are optimal values for n and m.

6. Experimental results and comparison

In order to examine the effectiveness and the efficiency of the proposed static and dynamic encoding schemes, we carried out experiments using 10 real world benchmarks. We used the ARM processor architecture in our experiments. Armulator [9] is used to analyze the execution of the benchmark programs. The bit width of the instruction is equal to 32 bits. The lengths of the memory bus and the cache bus are assumed to be 20mm and 15mm, respectively, in the analysis. The impact of the bus length on the overall energy reduction scheme will be analyzed at section 6.5. We used a 0.07μm technology and the corresponding values of the cross coupling capacitance and stand-alone capacitance are obtained

Table 2. Results of static encoding scheme

1Kb (eng. In nJ)	Energy Overhead		Block size = 16 instructions			Block size = 32 instructions			Block size = 64 instructions		
Sheme	LUT	Crossbar	org_eng	enc_eng	% red.	org_eng	enc_eng	% red.	org_eng	enc_eng	% red.
arrayinc	4	163	2846	2519	11.51	4574	3909	14.54	6854	5833	14.90
bitfieldeasy	3	120	1978	1785	9.77	2718	2428	10.69	4701	4198	10.70
bytedemo	8	329	5003	4444	11.17	7683	6712	12.65	15074	12831	14.88
dhry	116	4897	60533	59319	2.00	92959	87277	6.11	153508	138815	9.57
dowhilelv	1	34	525	470	10.39	691	607	12.15	1367	1186	13.26
globalvar	0.2	10	161	146	9.69	299	262	12.33	456	397	12.83
init	10	404	5676	5116	9.86	7506	6693	10.84	16173	13983	13.54
qsort	5	218	3119	2809	9.93	5530	4828	12.70	8686	7481	13.88
randtest	1	41	603	541	10.35	1012	890	12.12	1515	1305	13.91
regPromote	7	279	5299	4666	11.94	7810	6841	12.40	12971	11415	12.00
			Avg. % red.		9.66	Avg. % red.		11.65	Avg. % red.		12.95

Table 3. Results of dynamic encoding scheme

1Kb (eng. In nJ)	32 bits reordering						Limited reordering
Block Size	LUT energy	Crossbar Energy	org_eng	enc_eng	avg. % red.	max % red. among 10 benchmarks	avg. % red.
16	289	659	8574	7035	18.10	23.09	15.27
32	135	657	13078	10741	18.83	20.44	15.28
64	64	655	22130	18482	17.13	18.54	13.68

from [10]. 1V was used as the supply voltage. The instruction cache size is assumed to be 1K byte and the embedded system is assumed to run at 500MHz. Three different cache block sizes were simulated in our experiments. They are 16 instructions per block (64bytes), 32 instructions per block (128bytes) and 64 instructions per block (256bytes). All the additional hardware, including the look-up table and the crossbar are designed using Cadence and simulated using Hspice. SRAM is used for the look-up table deign and mux-based design is used for the crossbar.

6.1 Results of the static encoding scheme

Table 2 shows the simulation results of 10 benchmarks with different cache block sizes. The second and third columns show the energy consumption of the look-up table and the crossbar when executing the benchmarks. Org_eng and enc_eng are the total energy consumption at the memory bus and cache bus for the original scheme (i.e. no encoding) and the proposed static encoding scheme, respectively. The energy consumptions of the overhead are the same for different block sizes since one configuration is used for the whole program. The extra bus energy, energy consumption of the look-up table and the crossbar are included in the enc_eng. It can be seen that on average about 10% energy reduction is achieved and in some benchmarks, more than 14% energy reduction is achieved. It can also be seen that the energy reduction is increased with the cache block size. It is because the energy per miss is increased with the cache block size.

6.2 Results of the dynamic encoding scheme

Table 3 summarizes the simulation results of the dynamic encoding scheme. The same set of benchmarks is simulated. The values in table 3 are the average energy reduction of the 10 benchmarks. It can be seen that the energy consumption of the look-up table is much larger than in static one. The seventh column shows the result of the benchmark with maximum energy reduction among 10 benchmarks. It can be seen that on average 17 to 18% reduction are achieved for different cache block size and the maximum reduction can be as high as 23%. The last column shows the average energy reduction of the limited reordering in dynamic encoding scheme. Here we divided the 32-bits bus into 4 8-bits groups.

6.3 Comparison with previous work

Table 4. Comparison between different schemes

1KB **Table 4.**	Avg. % red by [7]	Comparison (% in further reduction)	
		static vs [7]	dynamic vs [7]
16	5.60	4.30	13.24
32	3.67	8.28	15.73
64	2.12	11.06	15.33

Comparisons with previous work [7] have also been made. This scheme rearranges the physical bit lines order based on the statistic of the dynamic trace of instructions sequences obtained by running a larger set of benchmarks. Then the permutation is implemented directly during physical design so that no additional hardware is needed. Table 4 shows the comparison among the static, dynamic schemes and the method proposed in

[7] for different block sizes. For dynamic encoding scheme, an extra 15% improvement can be obtained over that of [7].

6.4 Power, timing and area overhead

We simulated the delay and power overhead of the extra hardware. The average energy per access of the 320 bytes (for block size 16) look-up table, 160 bytes (for block size 32) look-up table and 80 bytes (for block size 64) look-up table is about 1.49pJ, 0.74pJ and 0.37pJ respectively. The average energy per access of the 32-bits crossbar is about 1.76pJ.

The delay of the cache is greater than that of the look-up table, and hence the look-up table is not on the critical path. However, the delay of 32-bits crossbar needs to be included in the critical path. The delay of the mux-based crossbar is about 0.4ns from the simulation results. Note that a straight simple implementation was used and no special delay and energy optimization technique has been used. If timing is a critical constraint in the embedded system, the limited reordering can be used. 4 sets of 8-bits crossbar are used instead of a large 32-bits crossbar. The delay of the crossbar is reduced to 45.2ps. The simulations on using 4 group of 8-bits configuration have been done for the dynamic encoding scheme by using the same set of benchmarks. The results are summarized in the last column of Table 3. It can be shown that on average the energy reduction is about 3-4% less than that of using 32-bits reordering scheme.

The area overhead is also considered. The layout area of 320 bytes (the largest one) look-up table is about 25000μm^2 and the area of the 32-bits crossbar is about 15360μm^2, which are relatively small compared to the area of the overall embedded system.

6.5 Impact of the bus length

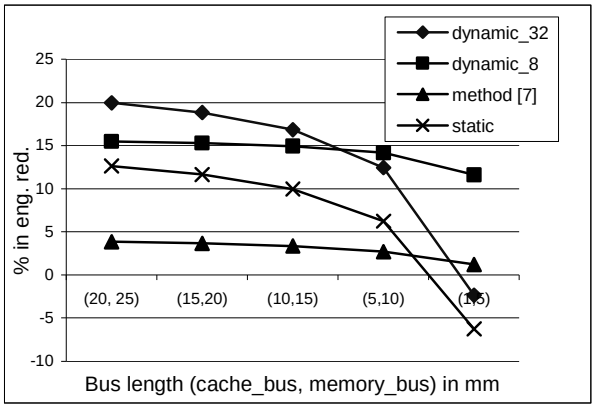

Figure 4. Energy reduction vs. buses length

The impact of the memory and cache bus length is also investigated. Because of the overhead, if the bus length is too short, the gain in the cross coupling switching capacitances reduction may not able to offset the overhead. So there is a minimum bus length requirement for this scheme. Figure 4 summarizes the simulation results for different encoding schemes for different bus lengths. For the dynamic encoding, we simulate both the 32-bits reordering (dynamic_32) and the limited reordering (dynamic_8). As expected it is shown that the energy reduction decreases with the bus length. If the bus length is too short, the static encoding and dynamic_32 encoding will not give good result since the energy consumptions of the overhead are relatively larger. Thus for shorter bus, it is better to use limited reordering instead of full 32-bits reordering.

7. Conclusions

In this paper, we propose both static and dynamic encoding schemes to reduce the cross coupling switching capacitances of the instruction buses. Both schemes are software-encoding schemes and encoding is carried out during compilation time. Experimental results show that, by using dynamic encoding scheme, 17-23% reduction can be obtained comparing with un-encoded instructions and on average 15% further reduction can be obtained compared to other existing work.

References

[1] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Trans. On VLSI Systems*, vol. 3, pp. 49-58, Mar. 1995.

[2] Y. Shin, S. Chae and K. Choi, "Partial bus-invert coding for power optimizations of system level bus," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp.127-129, 1998.

[3] C. L. Su, C. Y. Tsui, " Saving Power in the Control Path of Embedded Processors," *IEEE Design & Test Magazine*, Vol. 11, No. 4, pp.24-31, Winter 1994

[4] L. Benini, *et al.*, "Asymptotic zero-transition activity encoding for address buses in low-power microprocessor-based systems," in *Proc. the Great Lakes Symp. VLSI*, pp. 77-82, 1997

[5] T. Lv, *et al.*, "An Adaptive Dictionary Encoding Scheme for SOC Data Buses," *Proc. of the 2002 Design, Automation and Test in Europe Conference and Exhibition*, pp. 1059-1064, 2002

[6] J. Henkel, H. Lekatas, "A2BC: adaptive address bus coding for low power deep sub-micron designs," *Proc. of Design Automation Conference*, pp. 744-749, 2001

[7] L. Macchiarulo, *et al.*, "Low-Energy Encoding for Deep-Submicron Address Buses," *ISLPED'01*, pp. 176-181, Aug. 2001.

[8] Thomas H.Cormen, *et al.*, "Introduction to Algorithms," Second Edition: Cambridge, 2001.

[9] "Armulator,", http://www.arm.com

[10] "Predictive Technology Model," http://www-device.eecs.berkeley.edu/~ptm