Resource-Constrained Low-Power Bus Encoding with Crosstalk Delay Elimination

Meeyoung Cha, Chun-Gi Lyuh, and Taewhan Kim

Department of Electrical Engineering & Computer Science
and Advanced Information Technology Research Center (AITrc) *
Korea Advanced Institute of Science and Technology, KOREA

Abstract— In deep-submicron (DSM) technology, minimizing power consumption of a bus is one of the most important design objectives in embedded system-on-chip (SoC) design. In this paper, we address the problem of design space exploration of low-energy software bus encoding in embedded SoC design. Traditionally, finding a bus encoding that leads to a minimum energy consumption of bus has been an important research issue, but relatively little attention has been paid to the cost of software encoding implementation. In embedded system design, the memory space for storing the encoding information is strictly limited. Consequently, exploring the bus encoding implementation alternatives under such constraint becomes very necessary and/or useful. In this paper, we propose a systematic design space exploration algorithm for low-power bus encoding which completely eliminates the crosstalk delay. From experiments on a set of benchmark designs, the proposed algorithm was shown to consume 48% less power consumption on average over existing techniques with relatively little memory overhead.

I. Introduction

The power consumption and wire delay are the two most important design factors in SoC bus design. The loading capacitance and the coupling capacitance are the major sources of dynamic power consumption of a bus. When switching activity occurs, transition delay along the wire becomes twice or more than that of the wire with steady signal. (This delay penalty is called the *crosstalk delay* [1].)

It is known that lowering down the transition switching activity on the bit lines of a bus leads to a significant reduction of (dynamic) power consumption on bus [2]. Consequently, extensive research has been done for reducing the switching activity on the bus. The Gray code [3], the T0 code [4], and the Beach code [5] are designed for instruction address buses which are likely to be in consecutive values. The bus-invert code [6] is designed for data buses which are generally assumed to be random in values. The work in [7] partitioned the bus lines into two parts by analyzing the transition probabilities on the bit lines, and applied the bus-invert method partially to one of the parts. The authors in [8] further extended the work in [6] by grouping the bus lines into any arbitrary numbers, so that each group can be selectively applied for bus-invert coding. Meanwhile, those approaches only consider the self-transition activity on the bus. Recently, the authors in [9] considered the minimization of coupling activity on the bus to reduce power consumption. Further, the authors in [10] proposed a method to determine a relative placement order of the bit lines of a bus to reduce the coupling capacitance.

In high-speed designs where the crosstalk delay limits the clock speed, a simplistic scheme to prevent the crosstalk delay is to place a grounding wire between every two wires. Even though this shielding completely removes the crosstalk delay, the overhead on such scheme is not cost effective. To use less bus wires, recently Victor and Keutzer [1] (theoretically) introduced the concept of *self-shield encoding*. However, they did not propose any guideline or method of generating a self-shield bus encoding, also no comments were made for power minimization. Lyuh and Kim [11] complemented the work in [1] by proposing a technique which finds a self-shield bus encoding with minimum power consumption. One common factor among the approaches in [1, 3, 4, 5, 6, 7, 8, 9, 10, 11] is that their encoders are assumed to be implemented with hardware logic. However, in embedded system design, there are two ways in implementing encoders: hardware (i.e., ASIC or FPGA) and software (i.e., memory). In this paper, we address a new problem of software implemented bus encoding.

II. Preliminaries and Problem Definition

A. Interconnect Power Model

Dynamic power consumption on the interconnect can be calculated using the following formula:

$$P_{dyn} = (X \cdot (C_s + C_l) + Y \cdot C_c) \cdot V_{dd}^2 \cdot f_c \quad (1)$$

where C_s and C_l are the loading capacitances, C_c is the coupling capacitance, V_{dd} is the supply voltage, f_c is the clock frequency [9], and X and Y are formulated in following ways.

Self transition activity: X denotes the self transition activity for C_s and C_l. Let $p^i_{r,s}$ be the transition probability that the signal line i of bus changes from state $r \in \{0,1\}$ to $s \in \{0,1\}$. Then, the quantity of X_i for signal line i can be expressed as $X_i = p^i_{0,1}$.

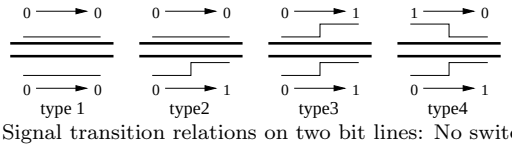


Fig. 1. Signal transition relations on two bit lines: No switching (*type 1*), single line switching (*type 2*), both lines switching to the same states (*type 3*), and both lines switching to the opposite states (*type 4*).

Coupling transition activity: Y denotes the coupling transition activity for C_c. It depends on the switching activity relation between two adjacent bit lines which can be classified

*Acknowledgment : This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

into four types as in Figure 1. In case of *type 1*, no switching activity occurs and dynamic power consumption is unaffected by C_c. *Type 2* is the case when single line switching occurs and the resulting state is different from the other. In this case, dynamic power consumption is affected by $\alpha \cdot C_c$ where α is a constant factor. In *type 3*, line switching occurs in both of the adjacent wires but to the same resulting states. In this case, dynamic power consumption is not affected by C_c like *type 1*. Finally, both of the adjacent wires change their states to different resulting states in *type 4*. In this case, dynamic power consumption is affected by $\beta \cdot C_c$ where β is a constant factor. The effective capacitance by *type 4* is larger than that by *type 2*, since β value is usually two times of α value.

Let $p^{i,j}_{pq,rs}$ denote the coupling transition probability that the ith bit line of a bus changes its state from $p \in \{0,1\}$ to $r \in \{0,1\}$, and at the same time the adjacent jth bit line changes its state from $q \in \{0,1\}$ to $s \in \{0,1\}$. Then, the quantity of Y between lines i and j can be calculated as:

$$Y_{i,j} = \alpha \cdot \sum_{s=0,1} (p^{i,j}_{ss,01} + p^{i,j}_{ss,10}) + \beta \cdot (p^{i,j}_{01,10} + p^{i,j}_{10,01}).$$

Finally, X and Y for dynamic power consumption in *Eq.1* are calculated as: $X = \sum_{i=1}^{W} X_i$, $Y = \sum_{i=1}^{W-1} Y_{i,i+1}$ where W is the number of bit lines of bus.

B. Self-Shield Encoding

Victor and Keutzer [1] introduced a self-shield encoding scheme in a way to reduce wire penalty while removing *type 4* switching activities. The idea is that the original code (i.e., *dataword*) whose bit-width is n can be encoded into a code whose minimal bit-width is m such that $m < 2n$ and there is no *type 4* switching in the encoded code sequence. The encoded codes are referred to as *codewords*.

We define capacitance ratio $\gamma = \frac{C_c}{C_s + C_l}$. The value of γ increases as the aspect ratio of the interconnect increases. It is easily shown that dynamic power consumption in *Eq.1* is proportional to the value of

$$Z = X + \gamma \cdot Y. \quad (2)$$

Therefore, we want to encode a set of datawords in a sequence so that the value of Z is as small as possible while completely eliminating the crosstalk delay.

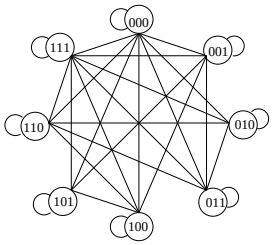
C. Static and Dynamic Encodings

Static encoding is the simplest encoding method that requires the least hardware resource for implementation. On the other hand, *dynamic* encoding can be used to enhance the flexibility of encoding. For example, one dataword can be encoded into several codewords according to the current codeword, current dataword, and the next dataword to transfer. Such encoding information is usually maintained in a table form using memory (i.e., software implementation). The degree of flexibility in dynamic encoding is proportional to the memory size required for storing the encoding information. (The memory size for decoding information can be measured accordingly from the corresponding dynamic encoding instance. For this reason, we focus on the problem of developing an effective (low-power) dynamic encoding technique in this paper.)

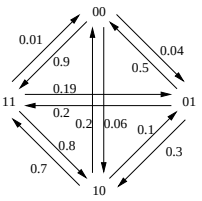
Let M_{enc} denote the memory size required for storing the encoding information for dynamic mapping. Further, suppose we have a memory constraint of M for storing the encoding information in the embedded system. Then, we want to have a dynamic mapping with M_{enc} such that $M_{enc} \leq M$ and the amount of power consumption (i.e., the quantity of Z in *Eq.2*) minimized while completely eliminating the crosstalk delays.

III. Low Power Bus Encoding

This section consists of two parts: (1) A low-power encoding technique with unlimited memory resource; (2) A low-power encoding technique with limited memory resource. Both encoding techniques ensure complete prevention of crosstalk delays. The key part of our consideration is an exploitation of data transition characteristics (based on a simulation profile) of the application's data sequences. Specifically, our low-power encoding techniques accept a set of data transition probabilities and bit width of a bus (i.e., n) as input, and produce a self-shielding dynamic encoding instance that minimizes the quantity of Z in *Eq.2* with minimal bit-width m.



(a) Codeword graph



(c) Transition probability graph

f	00	01	10	11
000 (00)	110	111	000	001
000 (01)	110	100	001	000
000 (10)	111	101	100	110
000 (11)	011	010	000	111
001 (00)	111	001	011	101
001 (01)	100	011	101	000
⋮			⋮	
111 (00)	100	110	000	111
111 (01)	000	101	001	010
111 (10)	011	101	100	001
111 (11)	111	011	001	010

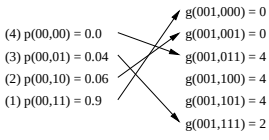
(b) A random encoding

transfer sequence	random	proposed	proposed with refinement
10	000	000	000
01	101	001	001
00	001	000	001
10	011	111	000
11	111	000	000
01	011	111	111
11	110	001	011
10	110	000	011
00	000	111	000
11	001	000	000
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
	6 transitions Z = 24	10 transitions Z = 16	4 transitions Z = 10

(h) Encoding example

f	00	01	10	11
000 (00)	100	001	111	000
000 (01)	000	100	111	001
000 (10)	111	001	100	000
000 (11)	001	111	000	100
001 (00)	011	111	001	000
001 (01)	000	011	001	111
⋮			⋮	
111 (00)	011	001	111	000
111 (01)	000	011	111	001
111 (10)	111	001	011	000
111 (11)	001	111	000	011

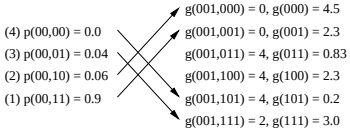
(d) The proposed encoding



(e) The proposed codeword selection

f	00	01	10	11
000 (00)	100	001	111	000
000 (01)	000	100	111	001
000 (10)	111	001	100	000
000 (11)	001	111	000	100
001 (00)	101	111	000	001
001 (01)	001	101	000	111
⋮			⋮	
111 (00)	110	011	000	111
111 (01)	111	110	000	011
111 (10)	000	011	110	111
111 (11)	011	000	111	110

(f) The proposed encoding with refinement



(g) Refined codeword selection

Fig. 2. Illustrative examples of the proposed encoding scheme.

A. Encoding without Resource Constraint

Suppose we want to encode the 2-bit datawords with 3-bit codewords. We use notation A to indicate a set of possible datawords and B to indicate a set of possible codewords. The graph, called *codeword graph*, in Figure 2(a) shows a crosstalk delay immune bit-transition relation among codewords where nodes represent distinct codewords and there is an edge between two nodes if there is no crosstalk delay between the bit-transition of corresponding codewords. We denote B_{c_i} to be the set of codewords that can immediately follow without causing crosstalk delay after the transfer of codeword c_i.

The table in Figure 2(b) shows an instance of random bus encoding where each entry in the left column of the table indi-

cates the codeword that is currently being transferred for the dataword in the parenthesis next to the codeword, and each entry in the right column of the table shows the codeword to be transferred next according to the dataword in the first row of the same column of the table. We can represent such encoding information by using a one-to-one mapping function f from A to B_{c_i} for every combination of codewords c_i in B and datawords in A. Further, let $g(c_i, c_j)$ be the weighted sum of self and coupling transitions incurred when codewords c_i and c_j are transferred successively.

However, note that a random encoding does not take into account the characteristics of application design which usually exposes a non-uniform distribution of the data values. To reflect this characteristic, we obtain the transition probability among datawords from a simulation profile, and utilize it in generating the encoding instance. Figure 2(c) shows an example of the *transition probability graph* for a specific application. Let $p(d_i, d_j)$ be the transition probability from dataword d_i to d_j. We estimate the total power consumption, Z, for an encoding with mapping function f by the quantity of $\tilde{Z}$ as follows:

$$\tilde{Z} = \sum_{\forall d_i \in A} \sum_{\forall d_j \in A} \sum_{\forall c_k \in B} p(d_i) \cdot p(d_i, d_j) \cdot g(c_k, f_{c_k(d_i)}(d_j))$$
$$\text{where,} \quad p(d_i) = \frac{\sum_{\forall d_k \in A} p(d_k, d_i)}{|A|}.$$

Then, we propose a greedy method for selecting the next codeword among a set of valid codewords for currently given codeword, dataword, and the next dataword to transfer.

GMap: (A, G_{prob}, B)
Low-power self-shield encoding w/o resource constraint
foreach (codeword c_i in B) {
  foreach (dataword d_j in A) {
    • Arrange all datawords in A by decreasing order of $p(d_j, \cdot)$ values;
    • Let $d_{k_1}, d_{k_2}, \cdots, d_{k_{|A|}}$ be the sorted list of datawords;
    • Arrange all codewords in B by nonincreasing order of $g(c_i, \cdot)$ values with tie-breaking by the value of $g(\cdot)$;
    • Let $c_{l_1}, c_{l_2}, \cdots, c_{l_{|B_{c_i}|}}$ be the sorted list of codewords;
    • Set $f_{c_i(d_j)}(d_{k_p}) = c_{l_p}$, $p = 1, 2, \cdots, |A|$;
  } **endfor**
} **endfor**
return (mapping_table f);

Fig. 3. A summary of GMap algorithm.

GMap: Let c_i be the current codeword encoded to transfer the current dataword d_j. To determine the next codeword to be transferred for any possible next datawords in A, GMap first arranges the datawords in A according to the decreasing order of $p(d_j, \cdot)$ values. Let d_{k_1} be the first dataword in the sorted list. Then, we map $f_{c_i(d_j)}(d_{k_1})$ to a codeword in B_{c_i} with the least value of $g(c_i, \cdot)$. Suppose, $f_{c_i(d_j)}(d_{k_1}) = c_{l_1}$. We then update the sorted dataword list as well as B_{c_i} by deleting d_{k_1} and c_{l_1}, and repeat the process until the list becomes empty. The intuitive idea of such mapping comes from the fact that assigning datawords with high (low) transition probability (i.e., $p(d_j \cdot)$) to codewords with low (high) bit transition cost (i.e., $g(c_i, \cdot)$) would be likely to reduce the total bit-transitions.

Figure 2(d) shows the mapping table obtained by GMap. In particular, Figure 2(e) shows how the next codewords to transfer are determined for possible next datawords. For example, GMap first sets $f_{001(00)}(11)$ to 000 because $p(00, 11)$ is the highest transition probability and $g(001, 000)$ is the least value. Next GMap set $f_{001(00)}(10)$ to 001 because $p(00, 10)$ is the next highest transition probability and $g(001, 001)$ is another least value. Figures 2(h) shows an example of encoding sequence for a sequence of datawords that follows the transition probability in Figures 2(c). We enhance the quality of GMap from the observation that a relatively large number of the $g(c_i, \cdot)$ values are identical. This implies that a good tie-breaking rule may reduce the quantity of Z considerably. We refine the procedure of GMap by integrating a simple but effective tie-breaking rule as follows.

Tie-breaking: Suppose we have two possible mappings c_{j_1} and c_{j_2} for codeword c_i with the same $g(c_i, \cdot)$ values. We estimate the average amount of g value for a transition from codeword c_j to the next possible candidate codewords in B_{c_j} as following:

$$g(c_j) = \frac{\sum_{\forall c_l \in B_{c_j}} g(c_j, c_l)}{|B_{c_j}|}.$$

Note that $g(c_j)$ indicates the expected value of self and coupling transitions incurred when a codeword in B_{c_j} is transferred after the transfer of c_j. Finally, we resolve the ties by selecting the mapping with the least value of $g(\cdot)$ among the mapping candidates.

Figure 2(f) shows the mapping table obtained by GMap with the proposed tie-breaking rule, and Figure 2(g) shows how the tie-breaking is applied to the mapping in Figure 2(e). Note that in Figure 2(h), the value of Z is reduced further in the encoding that applied the proposed tie-breaking rule to GMap. Figure 3 summarizes the overall procedure of GMap with tie-breaking, whose time complexity is polynomial-time bounded in terms of $|A|$ and $|B|$.

B. Bus Encoding with Resource Constraint

If we have a memory resource constraint to store the mapping table, i.e., $M_{enc} \leq M$, we need to produce a mapping table that is fitted into the given memory size. We accomplish this by gradually reducing the full mapping table obtained by GMap while retaining the encoding information in the original table as much as possible as follows.

GMap-resource: (A, G_{prob}, B)
Low-power self-shield encoding with resource constraint
repeat {
  • Arrange all codewords in B by increasing order of $|B_{(\cdot)}|$ values with tie-breaking by the value of $g(\cdot)$;
  • Let c_i be the codeword with the least value of $|B_{(\cdot)}|$;
  • Remove c_i from B;
  • Apply GMap(A,G_{prob},B) to get M_{enc} and f;
} **until** ($|B| \leq |A|$ or $M_{enc} \leq M$)

Fig. 4. A summary of GMap-resource algorithm.

GMap-resource: We reduce the size of codeword graph by removing one of the nodes at each iteration. Selection of a candidate codeword is based on the cardinality of the codeword nodes (i.e., $|B_{c_i}|$). We remove a codeword with the least value of $|B_{c_i}|$ because its transition to the next valid codeword is likely to be the least flexible, limiting the possibility of reducing the Z value for the consecutive transitions. If there are multiple such codewords with the same $|B_{c_i}|$ values, we choose the one with the largest $g(c_i)$ value because its removal is more likely to reduce the Z value on the average. Note that the mapping table produced after the i^{th} iteration has $|A|$ less rows than that at the $(i-1)^{th}$ iteration.

Figures 4 summarizes the overall procedure of GMap-resource, codewords that are likely to incur high transition costs are removed until memory constraint is met or there are no more than $|A|$ number of codewords left.

TABLE I
COMPARISONS OF POWER CONSUMPTION OF GMap AND GMap-resource OVER EXISTING SHIELD METHODS.

	Total power consumption, Cost: (#bit-lines, req. memory)						
Design	Compress		Laplace		Linear		
bit-width	16		16		16		
# instructions	176915		323930		485503		
Methods	Power	Cost	Power	Cost	Power	Cost	
Shield	2119353	31	4994726	31	5576707	31	
[6]+Shield	2107832	33	4956873	33	5816507	33	
[9]+Shield	2071269	33	4856614	33	5862607	33	
EN_Shield-lp[11]	1701000	27	4108606	27	4211227	27	
GMap	781616	27, 1.00	1541687	27, 1.00	2396435	27, 1.00	
GMap-resource	887930	27, 0.16	1834283	27, 0.04	2771543	27, 0.15	
Design	Lowpass		SOR		Wavelet		
bit-width	16		32		16		
# instructions	101107		240592		245		Normalized
Methods	Power	Cost	Power	Cost	Power	Cost	avg. power
Shield	1506380	31	9856623	63	2807	31	1.41
[6]+Shield	1522263	33	6265728	65	2954	33	1.23
[9]+Shield	1501961	33	7327673	65	3086	33	1.35
EN_Shield-lp[11]	1102983	27	4973588	55	2085	27	1.00
GMap	458768	27, 1.00	1908913	55, 1.00	1078	27, 1.00	0.45
GMap-resource	504232	27, 0.07	2239761	55, 0.04	1180	27, 0.10	0.52

IV. EXPERIMENTAL RESULTS

We implemented the proposed low-power encoding techniques GMap and GMap-resource in C++ and executed on the Intel Pentium IV computer on benchmark designs: Compress, Laplace, Linear, Lowpass, SOR and Wavelet[12]. The results are compared with the results by existing algorithms Naive shielding, a combined bus-invert[6] and naive shielding, a combined coupling-driven[9] and naive shielding, and EN_shield-lp[11].

Table 1 shows the effectiveness of our encoding techiniques over existing encoding techniques. Note that all the encoding techniques guarantee complete elimination of crosstalk delays. We measured the power consumption in terms of the quantity of Z in *Eq.2*. *#bit-lines* indicates the number of total bit lines required including shield and invert lines and *req. memory* indicates relative mapping table size required for GMAP or GMAP-resource. Due to the time complexity of EN_shield-lp, GMap, and GMap-resource, we partitioned the given data into 4-bit segments and applied the encoding schemes to each segment separately. In summary, our techniques reduce the power consumption up to 62% compared to the best known static encoding technique EN_shield-lp if there is a sufficiently large memory resource. Even under a resource constraint, we are able to reduce the amount of power consumption up to 55% with relatively little memory resource overhead but with using the same number of bit lines.

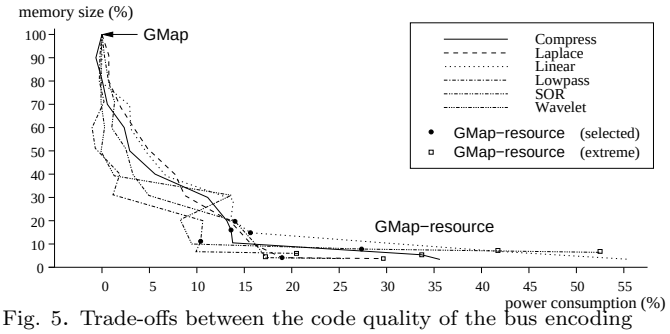


Fig. 5. Trade-offs between the code quality of the bus encoding and memory resource overhead explored by GMap and GMap-resource.

Figure 5 shows the trade-offs between power consumption and required memory overhead for each benchmark design that was explored by GMap and GMap-resource. The left starting point of each curve indicates the power and memory produced by GMap. The shape of the curves produced by GMap-resource clearly shows that the proposed techniques can be used effectively to find an instance of a bus encoding that satisfies the resource constraint, and at the same time maximally reduces the power consumption with no crosstalk delays.

V. CONCLUSIONS

This paper proposed a new design space exploration techniques for bus encoding in embedded SoC design. The proposed bus encoding techniques address the following three important design objectives simultaneously: (1) minimizing the power consumption of a bus; (2) completely eliminating the crosstalk delay on a bus; (3) satisfying the memory size constraint for implementation. From experiments on a set of benchmark designs, the proposed algorithm was shown to consume 48% less power on average over existing techniques with relatively little memory overhead.

REFERENCES

[1] B. Victor and K. Keutzer, "Bus Encoding to Prevent Crosstalk Delay," *Proc. ICCAD*, 2001.

[2] A. P. Chandrakasan and R. W. Broderson, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[3] C.L. Su, et al., "Saving Power in the Control Path of Embedded Processors," *IEEE Design and Test of Computers*, 1994.

[4] L. Benni, et al., "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-based Systems," *Proc. GLSVLSI*, 1997.

[5] L. Benni, et al., "System-level Power Optimization of Special Purpose Applications, the Beach Solution," *ISLPED*, 1997.

[6] M. R. Stan and W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE TVLSI*, 1995.

[7] Y. Shin, et al., "Partial Bus-Invert Coding for Power Optimization of Application-Specific Systems," *IEEE TVLSI*, 2001.

[8] S. Hong, et al., "Decomposition of Bus-Invert Coding for Low-Power I/O," *JCSC*, 2000.

[9] K.-W. Kim, et al., "Coupling-Driven Signal Encoding Scheme for Low-Power Interface Design," *Proc. ICCAD*, 2000.

[10] Y. Shin and T. Sakurai, "Coupling-Driven Bus Design for Low-Power Application-Specific Systems," *Proc. DAC*, 2001.

[11] C. Lyuh and T. Kim, "Low Power Bus Encoding with Crosstalk Delay Elimination," *Proc. ASIC/SOC*, 2002.

[12] P. R. Panda and N. Dutt, "1995 High-level Synthesis Design Repository," *Proc. ISSS*, 1995.