Bus-Invert Coding for Low-Power I/O

Mircea R. Stan, *Member, IEEE,* and Wayne P. Burleson, *Member, IEEE*

***Abstract*— Technology trends and especially portable applications drive the quest for low-power VLSI design. Solutions that involve algorithmic, structural or physical transformations are sought. The focus is on developing low-power circuits without affecting too much the performance (area, latency, period). For CMOS circuits most power is dissipated as dynamic power for charging and discharging node capacitances. This is why many promising results in low-power design are obtained by minimizing the number of transitions inside the CMOS circuit. While it is generally accepted that because of the large capacitances involved much of the power dissipated by an IC is at the I/O little has been specifically done for decreasing the I/O power dissipation.**

We propose the *Bus-Invert* method of coding the I/O which lowers the bus activity and thus decreases the I/O peak power dissipation by 50% and the I/O average power dissipation by up to 25%. The method is general but applies best for dealing with buses. This is fortunate because buses are indeed most likely to have very large capacitances associated with them and consequently dissipate a lot of power.

***Index Terms*—low-power dissipation, CMOS VLSI, coding.**

I. Introduction

CMOS VLSI is intrinsically a low-power technology [23]. When compared to TTL, ECL or GaAs at similar levels of integration the power dissipated by CMOS is several orders of magnitude lower. This is one of the major reasons for the widespread acceptance of CMOS in all kinds of applications from consumer appliances to some supercomputers. The power dissipated in a CMOS circuit can be classified as static power dissipation (overlap current and DC static) and dynamic power dissipation. The static power dissipated by a CMOS VLSI gate is in the nanowatt range [23], [9] and for the purpose of this paper will be ignored. The dynamic power dissipated by a CMOS circuit is of the form [23], [14]:

$$P_{chip} \propto \sum_{i=1}^{N} C_{load_i} \cdot V_{dd}^2 \cdot f \cdot p_{t_i} \qquad (1)$$

where the sum is done over all the N nodes of the circuit, C_{load_i} is the load capacitance at node i, V_{dd} is the power supply voltage, f is the frequency and p_{t_i} is the activity factor at node i.

The special interest in low-power CMOS design is relatively recent and is due to an increased interest in portable applications. For achieving low-power in CMOS circuits one or more of the terms V_{dd}, C_{load_i}, f and p_{t_i} must be minimized. Decreasing V_{dd} has a quadratic effect and is thus a very efficient way of decreasing the power dissipation [3]. Even then the decrease in power obtained by lower V_{dd} is not of several orders of magnitude as required by portable applications. This is why lowering V_{dd} must be done in conjunction with other methods for decreasing the power dissipation even further. Lowering the activity factor is a very promising way of further decreasing the power dissipation. It can be viewed at different *levels of abstraction* [3]:

- In the *behavioral* domain advances at the *algorithmic* level can dramatically decrease the number of transitions by lowering the number of steps necessary for a given computation [3].
- In the *structural* domain at the *logic* level there are approaches [18], [11] that consider the switching probabilities associated with different gate implementations. For random inputs the output of an AND gate is more likely to be 0 than 1 while the output of an OR gate is more likely to be 1 than 0. An XOR gate has equal probabilities for 0 or 1. By properly choosing the multilevel implementation of a boolean function the circuit can be optimized such that the total number of transitions is minimized.
- At the *logic* and *layout* levels the effect of *glitches* must be considered. Glitches represent unnecessary and unwanted transitions that contribute among other things to increased noise and power dissipation. Among general circuit topologies balanced-trees are less likely to generate glitches than chain topologies [4] when properly balanced also at the layout level.

The *peak* power dissipation is determined by the maximum instantaneous value of the activity factor while the *average* power is determined by the average activity factor. Decreasing the average power dissipation is generally the target of low-power techniques but the maximum activity factor is also important because of its relationship with the simultaneous switching noise and the resulting ground bounce [13]. Figs. 1 and 2 illustrate two cases that have the same average but very different maximum switching activities.

In Section II we propose *coding* as a method of decreasing the activity factor on buses. Section III explains the *Bus-Invert* method suitable for data buses while section IV considers coding on an address bus. Coding the I/O was proposed in [21], [16] as a noise reduction method. Codes were developed for reducing the I/O transmitted noise for both terminated and unterminated lines. Since transitions are a common cause in CMOS for both noise and power dissipation many of the methods used in [21], [16] can be applied for low-power design. This was independently done in [20]. There is also a patent [8] that proposes coding the I/O for reduced state changes. The *Bus-Invert* method can be characterized as an example of what in [21] is called "starvation coding" and in [20] is called "limited-weight coding".

Manuscript received April 26, 1993; revised April 15, 1994 and August 11, 1994. This work was supported in part by the NSF under Grant MIP-9 208267.

The authors are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 USA.

IEEE Log Number 9408370.

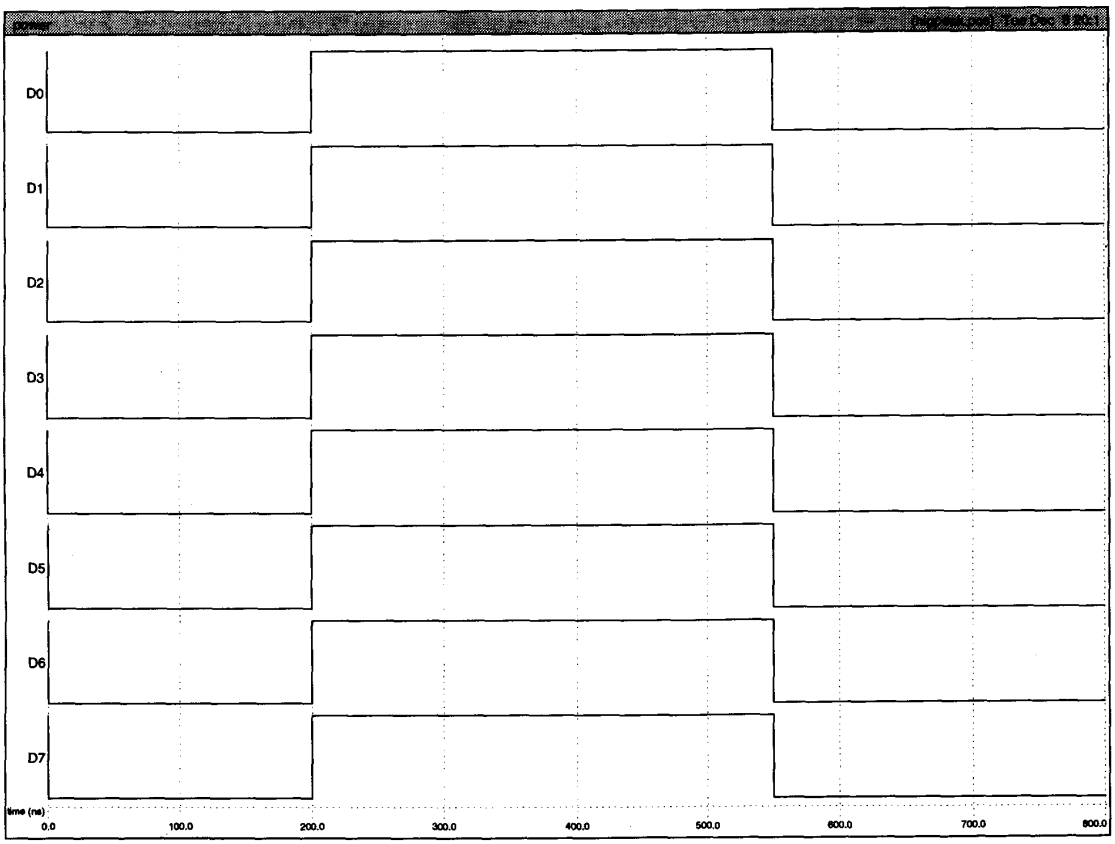


Fig. 1. An eight-bit bus on which all eight lines toggle at the same time and which has a high peak (worst-case) power dissipation. There are 16 transitions over 16 clock cycles (average 1 transition per clock cycle).

II. Coding for Low-Power I/O

The floorplan of a VLSI IC is being formed of the internal circuit surrounded by the I/O padframe. The power dissipated at the I/O can be as low as 10% [7] and as high as 80% [15] of the total power dissipation. For circuits optimized for low-power the power dissipated at the I/O is typically around 50% of the total [24]. This large I/O power dissipation is a consequence of the huge (compared with the internal circuit) dimensions of the devices in the I/O pads and of the external capacitances due to I/O pins, wires and connected circuits. The devices in the I/O pads need to be large in order to drive the large external capacitances and this further increases their own parasitic capacitances. The effect of the large capacitances is twofold: long delays (which lead to lower performance) and high-power dissipation. The usual way of addressing I/O pads performance and power dissipation is at the layout and circuit level [13]. In this paper we consider I/O power dissipation at a higher level of abstraction.

The research on low-power design has focused on the internal circuit [3]–[6], [11], [18] despite the fact that the I/O power dissipation is at least an important part if not the dominant factor in power dissipation [15]. One reason is that the I/O is considered somehow "fixed" or "given." In high-level synthesis the I/O is likely to be a constraint on the design, part of the initial specification. The chip is designed such that it obeys a predefined I/O operation. When this "black-box" approach is taken there is little freedom to modify the I/O for low-power. We change that and look at coding in order to decrease the I/O activity.

In order to make (1) tractable a simplification that is generally made is to consider that each node has the same "average" load capacitance [18]. If V_{dd} and f are the same for all gates (1) becomes:

$$P_{chip} \propto C_{average} \cdot V_{dd}^2 \cdot f \cdot N(transitions) \qquad (2)$$

where $N(transitions)$ is the maximum (for peak power) or average (for average power) *total* number of transitions for the entire chip.

Considering the same "average" load capacitance for all the nodes can sometimes lead to wrong conclusions. Nodes with very large capacitances can dominate the total power dissipation. For example in [2] the power dissipation of different adders is first estimated with a simplified model and then actually measured on a fabricated chip. The estimation for the carry look-ahead adder was far from the actual measured value exactly because the influence of the large capacitance look-ahead tree was oversimplified. In this paper a simplified model is also used but this time with *two* different types of nodes: a small capacitance node typical for the internal circuit and a large capacitance node typical for the I/O. We also

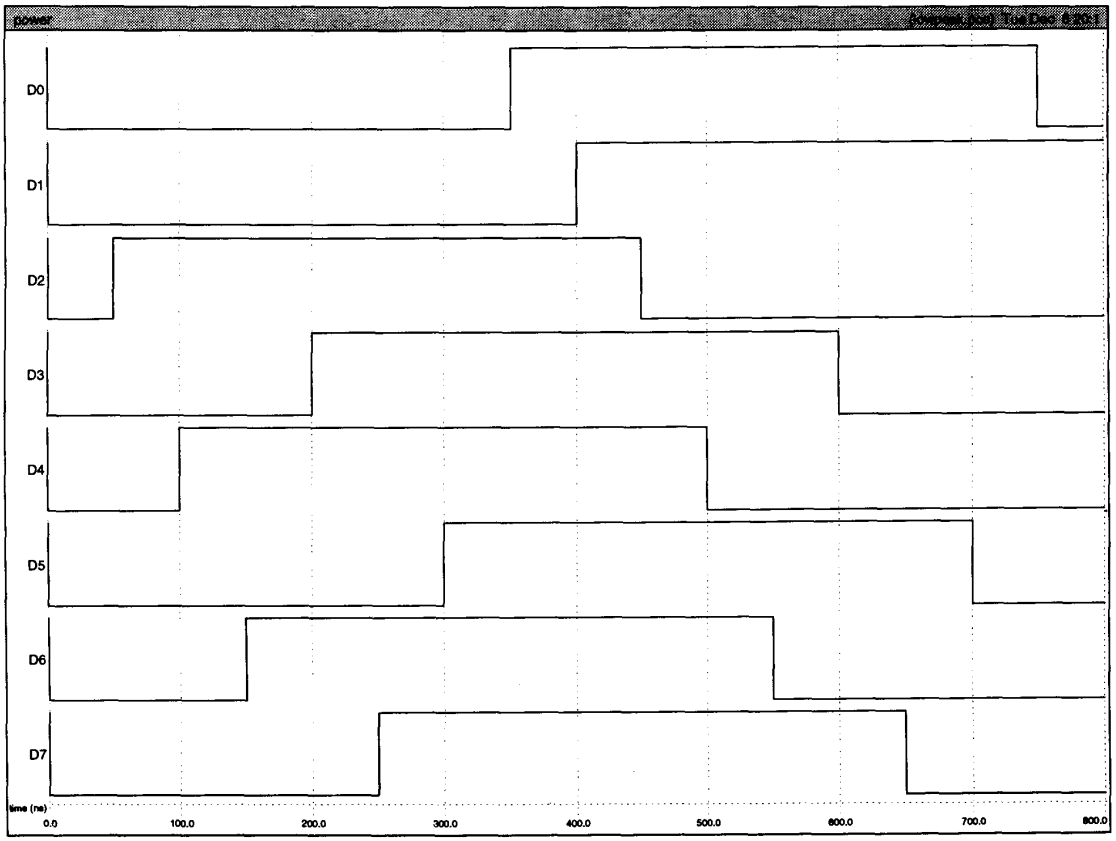


Fig. 2. An eight-bit bus on which the eight lines toggle at different moments and which has a low peak power dissipation. There are the same 16 transitions over 16 clock cycles and thus the same average power dissipation as in Fig. 1.

assume that the V_{dd} and f have already been optimized for low-power and thus the only way of further decreasing the power dissipation is to lower the number of transitions. This assumption is in the spirit of [18]. With these assumptions the total power can be written as a sum of the power dissipated by the internal circuit and the power dissipated at the I/O:

$$P_{chip} \propto C_{int} \cdot N(transitions)_{int} + C_{I/O} \cdot N(transitions)_{I/O}$$

The internal number of transitions $N(transitions)_{int}$is generally much larger than $N(transitions)_{I/O}$ because the number of internal nodes is much larger than the number of I/O nodes, while C_{int} is generally much smaller than $C_{I/O}$. The total power dissipated by the chip is then the sum of two comparable terms:

- P_{int} (average or peak): the product of a small internal capacitance and a large (average or maximum) number of transitions and,
- $P_{I/O}$ (average or peak): the product of a large I/O capacitance and a small (average or maximum) number of transitions.

We are now ready to state the idea behind our proposed method of decreasing the power dissipation: Code the data in order to decrease the number of transitions on the large capacitance side (I/O) even at the expense of slightly increasing the number of transitions on the low capacitance side (internal circuit).

Intuitively. the total power dissipation will decrease by decreasing the number of transitions on the high capacitance side. Because the number of internal transitions is already large, increasing it by a comparatively small amount is likely to be insignificant. A numerical example will help illustrate this. Let's consider an 8-bit wide data bus on which coding is used in order to decrease power dissipation. By coding (see section III) the maximum bus activity can be reduced by 50% and the average bus activity by at most 25%. The increase in the internal number of transitions is of the order of $n \log n$ where n is the bus width (see section III-D). With the power dissipated at the I/O 50% of the total we'll consider: $V_{dd} = 1V$, $f = 1MHz$, $N(transitions)_{I/O} = 8$, $C_{I/O} = 2500$ and $N(transitions)_{int} = 2500$, $C_{I/O} = 8$ (a conservative estimate that the I/O capacitances are two orders of magnitude larger than the internal capacitances). The power dissipated in the uncoded case will then be: $P_{uncoded} = 2500 \cdot 8 + 8 \cdot 2500 = 40000$.

If the above $N(transitions)_{I/O} = 8$ is the *average* number of transitions by coding it can be reduced to 6 (25%). At the same time $N(transitions)_{int}$ will be increased with $n \log n = 8 \cdot 3 = 24$. The power dissipated becomes:

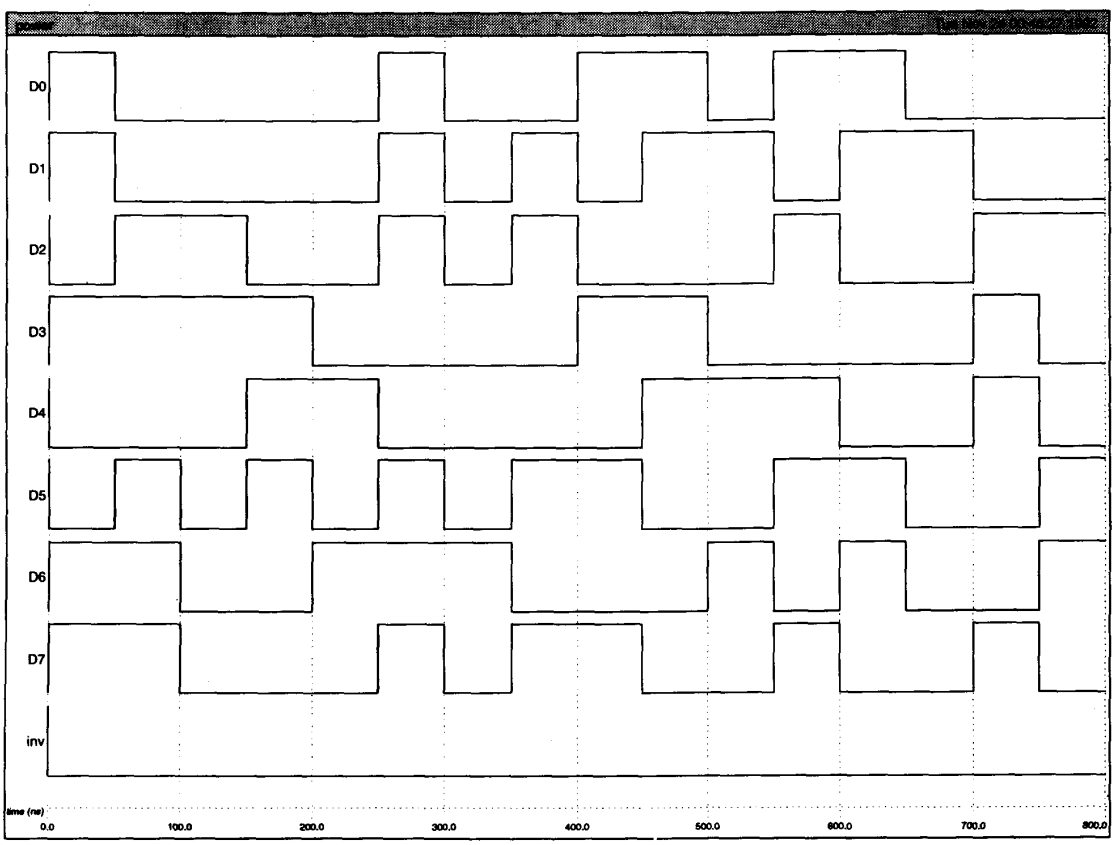


Fig. 3. A typical eight-bit synchronous data bus. The transitions between two consecutive time-slots are "clean." There are 64 transitions for a period of 16 time-slots. This represents an average of 4 transitions per time-slot, or 0.5 transitions per bus line per time-slot.

TABLE I
TYPICAL VALUES OF 33%, 50% OR 67% FOR THE PERCENTAGE OF I/O POWER DISSIPATION FROM THE TOTAL POWER DISSIPATION ARE CONSIDERED. WITH CODING THE MAXIMUM I/O POWER DISSIPATION CAN BE REDUCED BY HALF AND THE AVERAGE I/O POWER DISSIPATION CAN BE REDUCED BY 25%

If the amount of I/O power dissipation from total is:	If the I/O maximum power is lowered to:	the total maximum power is lowered to:	If the I/O average power is lowered to:	the total average power is lowered to:
33%		84%		91%
50%	50%	75%	75%	88%
67%		67%		83%

$P_{average} = 2500 \cdot 6 + 8 \cdot 2524 = 35192 \approx 88\%$ of $P_{uncoded}$.

If $N(transitions)_{I/O} = 8$ is the *maximum* number of transitions by coding it can be reduced to 4 (50%). The increase in $N(transitions)_{int}$ will be the same $n \log n = 8 \cdot 3 = 24$. The maximum power dissipated after coding is: $P_{maximum} = 2500 \cdot 4 + 8 \cdot 2524 = 30192 \approx 75\%$ of $P_{uncoded}$. As can be seen the savings in I/O power dissipation translate almost entirely into savings at the chip level with very little penalty due to the increase in the internal activity factor.

Table I gives an idea of the approximate decrease in total power dissipation when the I/O number of transitions is cut by 50% (for maximum) or by 25% (for average) by considering some typical values for the percentage of I/O power dissipation from the total power.

III. POWER DISSIPATION ON A DATA BUS

Buses are a typical model for I/O communication. Furthermore buses are generally heavily loaded and consequently dissipate a lot of power. Glitches can be a serious cause of increased power dissipation because they represent unnecessary transitions [3], [5]. This is why we consider a synchronous bus model in which the transitions between two consecutive time-slots are "clean" (Fig. 3). Latches are needed on the bus-drivers side for such "clean" transitions.

We'll consider the activity on a typical data bus to be characterized by a *random uniformly distributed* sequence of values. Of course this is just an approximation, in general there is a degree of redundancy which can be exploited for example by lossless compression techniques [25]. Data compression is an efficient method to decrease power dissipation and by removing the extra redundancy the data can be accurately approximated as a random sequence. Even if compression is not used we believe that a random process is a reasonable approximation for the sequence on a data bus for our purpose. The assumption of random uniformly distributed inputs is also conveniently made by most of the statistical power estimation methods [18]. With this assumption for any given time-slot the data on an n-bit wide bus can be any of 2^n possible values with equal probability. The average number of transitions per time-

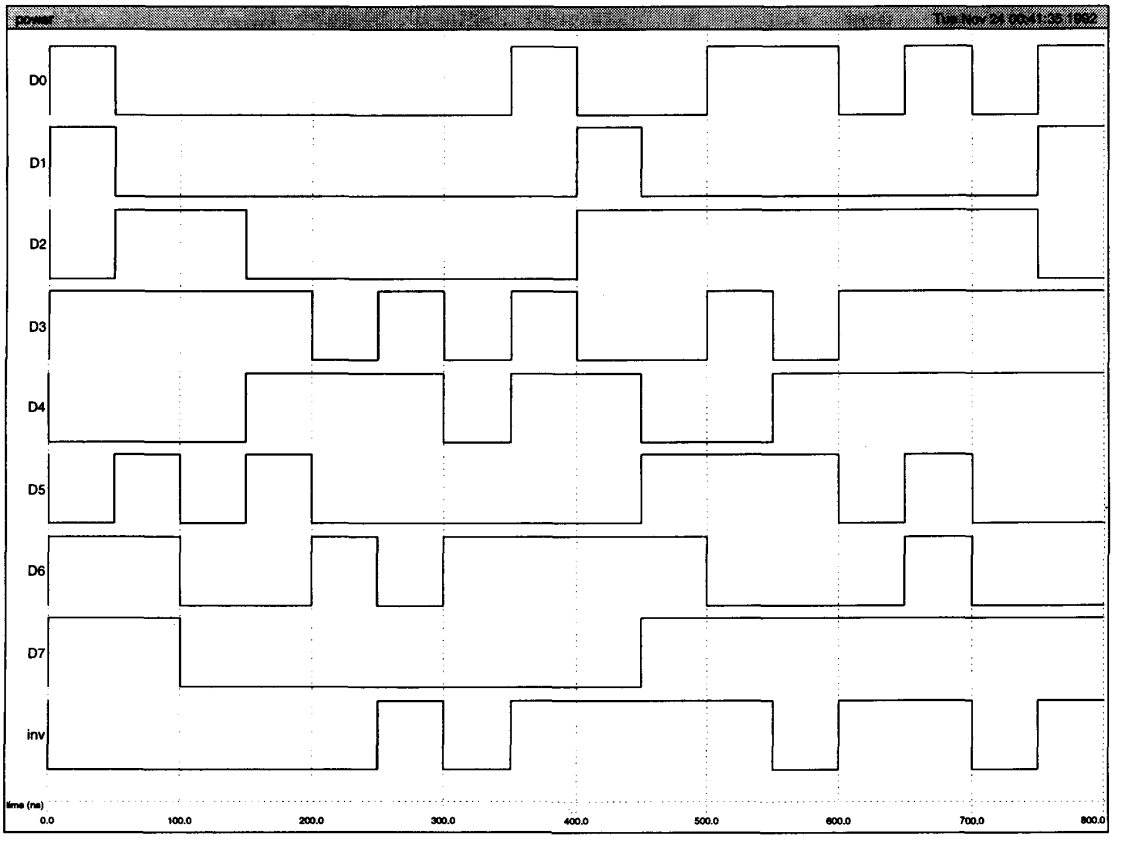


Fig. 4. The same sequence of data as in Fig. 3 coded using the $Bus-Invert$ method. There are now only 53 transitions over a period of 16 time-slots. This represents an average of 3.3 transitions per time-slot, or 0.41 transitions per bus line per time-slot. The maximum number of transitions for any time-slot is now 4.

slot will be $n/2$. For example on an eight-bit bus there will be an average of 4 transitions per time-slot or 0.5 transitions per bus-line per time-slot. Thus the average power dissipation for the I/O will be proportional with $n/2$. When all the bus-lines toggle at the same time (the probability of this happening in any time-slot is $1/2^n$) there will be a maximum of n transitions in a time-slot and thus the peak (worst-case) power dissipation is proportional with n. The *ground-bounce* is also worst in this case [13].

Fig. 3 shows a sequence of 16 time-slots for an 8-bit data bus. The values were pseudo-randomly generated and will be used further for illustration purposes:

```
D0 : 1000010011011000
D1 : 1000010101101100
D2 : 0110010100010011
D3 : 1111000011000010
D4 : 0001100001110010
D5 : 0101010110011001
D6 : 1100111000101001
D7 : 1100010110010010
```

The total number of transitions for these pseudo-random values for the 16 time-slots is 64 or an average of 4 transitions per time-slot, exactly as in the theoretical analysis. Because of the short time-frame the maximum number of transitions for this example is only 6. The question is: can we do better? Can we decrease the number of transitions for the I/O while transferring the same amount of information? If we consider the I/O as something "fixed" or as a "constraint" on the design then there is nothing we can do. But if we consider the I/O as another design variable then the problem becomes a typical coding case: Code the I/O in order to transfer the same amount of information in the same amount of time but with a lower number of transitions and thus decrease the total peak and average power dissipation.

A. Bus-Invert—Coding for Low-Power on a Data Bus

Let's denote as the *data value* the piece of information that has to be transmitted over the bus in a given time-slot. Then the *bus value* will denote the coded value (the actual value on the bus). Typically a code needs extra *control bits*. The *Bus-Invert* method proposed here uses one extra control bit called *invert*. By convention then $invert = 0$ the bus value will equal the data value. When $invert = 1$ the bus value will be the *inverted* data value. The peak power dissipation can then be decreased by half by coding the I/O as follows (*Bus-Invert* method):

1) Compute the Hamming distance (the number of bits in which they differ) between the present *bus value* (also counting the present invert line) and the next *data value*.

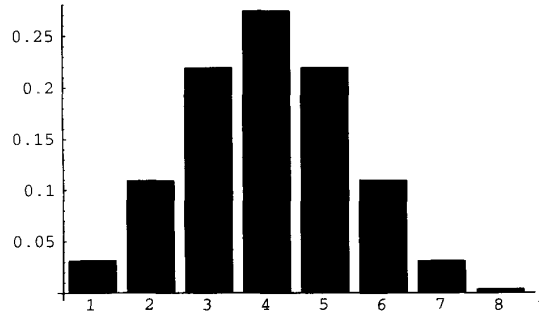


Fig. 5. The binomial distribution for the Hamming distances of the next data value. The maximum is at n/2=4.

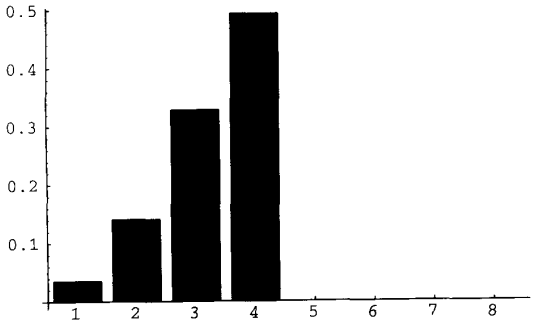


Fig. 6. The probability distribution for the Hamming distances of the next bus value for an 8-bit data bus encoded with the $Bus-Invert$ method. The transitions of the $invert$ signal are also counted.

2) If the Hamming distance is larger than $n/2$, set $invert = 1$ and make the next bus value equal to the inverted next data value.
3) Otherwise let $invert = 0$ and let the next bus value equal to the next data value.
4) At the receiver side the contents of the bus must be conditionally inverted according to the *invert* line, unless the data is not stored encoded as it is (e.g., in a RAM). In any case the value of *invert* must be transmitted over the bus (the method increases the number of bus lines from n to $n + 1$).

The *Bus-Invert* method generates a code that has the property that the *maximum* number of transitions per time-slot is reduced from n to $n/2$ and thus the peak power dissipation for the I/O is reduced by half. From the coding theory point of view the *Bus-Invert* code is a time-dependent Markovian code. Fig. 4 shows the same data sequence as Fig. 3 only this time using the *Bus-Invert* coding in order to decrease the number of transitions:

```
D0 : 1000000100110101
D1 : 1000000010000001
D2 : 0110000011111110
D3 : 1111010100101111
D4 : 0001110110011111
D5 : 0101000001110100
D6 : 1100101111000100
D7 : 1100000001111111
inv : 0000010111101101
```

While the maximum number of transitions is reduced by half the decrease in the *average* number of transitions is not as good. For an 8-bit bus for example the average number of transitions per time-slot by using the *Bus-Invert* coding becomes 3.27 (instead of 4), or 0.41 (instead of 0.5) transitions per bus-line per time-slot. This means that the *average* number of transitions is 81.8% as compared with an unencoded bus. There are two reasons why the performance of the *Bus-Invert* coding for decreasing the *average* number of transitions is not as good as for decreasing the *maximum* number of transitions:

- the *invert* line contributes itself with some transitions,
- the distribution of the Hamming distances for the next data values is *not* uniform.

Fig. 5 shows the distribution of the Hamming distance for the next *data value* with the probabilities that the next value will differ from the present one in 1, 2, 3, 4, 5, 6, 7, or 8 b positions. It is a *binomial* distribution where the maximum probability is for a Hamming distance of 4 (and is equal to $\frac{C_8^4}{2^8} = 0.27$). Unfortunately 4 (or $n/2$ in the general case) is exactly the value for which there is no gain in inverting the contents of the bus and thus the coding is ineffective for that value.

The distribution of the Hamming distances for the next *bus value* when using the *Bus-Invert* method is shown in Fig. 6. The very large probability (0.5) at 4 means that it is likely that the next value on the bus will differ from the present one in 4 positions. This is why the average number of transitions per time-slot is large (3.27), and thus the decrease in average power dissipation is less than for peak power dissipation.

Until now the particular case of an 8-bit bus was considered. How does the method perform on other bus widths? It turns out that as the width of the bus n increases the term corresponding to $n/2$ in the binomial distribution becomes dominant and the decrease in average power dissipation becomes even smaller. For example for a 16-bit data bus the average number of transitions per time-slot becomes 6.8 (compared to 8 for an unencoded bus) for an average of 0.43 transitions per bus-line per time-slot, which is 85% of the unencoded case (compare with 82% for an 8-bit bus). For an even wider bus (e.g., 64) the decrease in average power dissipation gets even smaller. The decrease in peak power dissipation remains the same 50% for any n.

B. The Bus-Invert Method is Optimal

With the assumption of random distribution for the sequence of data the *Bus-Invert* method is optimal in the sense that given the same redundancy (1 extra bus line) no other coding can achieve better reduction in the number of transitions. This can be seen by considering all possible next values on the bus. The next value can be the same as the present one and this will correspond to no transitions on the bus. The next value can differ in only one position (one transition) and there will be $C_{n+1}^1 = n+1$ such possible next values (there will be $C_n^1 = n$ values with $invert = 0$ and another one (all 1s data value) with $invert = 1$). Similarly there will be C_{n+1}^2 values that will generate two transitions, C_{n+1}^3 values that will generate three transitions, up to $C_{n+1}^{n/2}$ values that will generate $n/2$ transitions (assume n even without loss of generality). As can

be seen:

$$C_{n+1}^0 + C_{n+1}^1 + C_{n+1}^2 + \cdots + C_{n+1}^{n/2} = 2^n$$

and all 2^n possible next values are taken into account. The average number of transitions per bus cycle will be:

$$\frac{C_{n+1}^0 \cdot 0 + C_{n+1}^1 \cdot 1 + C_{n+1}^2 \cdot 2 + \cdots + C_{n+1}^{n/2} \cdot n/2}{2^n}$$

It can be seen that the *Bus-Invert* method uses *all* the patterns with at most $n/2$ transitions. This is what in [20] is called a *perfect limited-weight code* and in [21] is called a *starvation code*. Any other code with 2^n codewords can either use a permutation of these same patterns and then will exhibit the same average bus activity or use patterns with more than $n/2$ transitions and generate a larger bus activity. From this point of view the *Bus-Invert* method is optimal and any other coding method must use more than one extra line in order to further decrease the bus activity.

C. Partitioned Code for Lower Average Power

In order to decrease the average I/O power dissipation for wide buses the observation that the *Bus-Invert* method performs better for small n can be used to partition the bus into several narrower *subbuses*. Each of these subbuses can then be coded independently with its own *invert* signal. For example a 64-bit bus could be partitioned into eight 8-bit subbuses with a total of eight *invert* signals. Because of the assumption that the data to be transferred over the wide bus is random uniformly distributed, the statistics for the narrower subbuses will be independent and the sequence of data for each subbus will be random uniformly distributed. For example for a 64-bit bus partitioned into eight 8-bit subbuses the average number of transitions per time-slot will be 26.16 (8 times 3.27, the average for one 8-bit subbus) and the average number of transitions per bus-line per time-slot will be 0.41 (as for an 8-bit bus with one *invert* line). The maximum number of transitions is not improved by partitioning the bus and remains the same $n/2$.

Partitioning in order to get a lower average number of transitions has the obvious drawback of needing the extra *invert* lines, but as was shown in section 3.2 *any* code that tries to improve on the *Bus-Invert* has to use extra bits. On the other hand a partitioned code is no longer *optimal* and there are other codes that using the same number of extra bits as a partitioned code can achieve a lower bus activity. These are the codes that systematically use patterns with m or less number of transitions where $m < \frac{n}{2}$ and which are called "m-limited weight codes" in [20]. Unfortunately the *algorithmic* generation of such codes is not well understood at this moment and using look-up tables in ROM as suggested in [21] is out of the question because of the extra area and power dissipation.

For a partitioned *Bus-Invert* code the lowest number of transitions is obtained for subbuses of width 2. That means that for a *minimum* average number of transitions a 64 b bus would need to be partitioned into 32 2 b subbuses. Obviously the penalty induced by the large number of extra bus lines ($n/2$) will generally favor a coarser partitioning. The average number of transitions per bus-line per time-slot for a 2 b bus is 0.375 (as compared to 0.5 for an unencoded bus) or 75% of the unencoded case. Fig. 7 considers the previous example (see Figs. 3 and 4) of an 8 b bus, this time partitioned into four 2 b subbuses.

TABLE II
COMPARISON OF UNENCODED I/O AND CODED I/O WITH ONE OR MORE *invert* LINES. THE COMPARISON LOOKS AT THE AVERAGE AND MAXIMUM NUMBER OF TRANSITIONS PER TIME-SLOT, PER BUS-LINE PER TIME-SLOT, AND I/O POWER DISSIPATION FOR DIFFERENT BUS-WIDTHS

nr. of bus lines	mode	average transitions /time-slot	average transitions /bus-line	average I/O power dissipation	maximum transitions /time-slot	maximum transitions /bus-line	peak I/O power dissipation
2	unencoded	1	0.5	100%	2	1	100%
	1 *invert*	0.75	0.375	75%	1	0.5	50%
8	unencoded	4	0.5	100%	8	1	100%
	1 *invert*	3.27	0.409	81.8%	4	0.5	50%
	4 *invert*	3	0.375	75%			
16	unencoded	8	0.5	100%	16	1	100%
	1 *invert*	6.83	0.427	85.4%	8	0.5	50%

As can be seen from Table II the decrease in average power dissipation for the *Bus-Invert* method with or without partitioning is not very large but at least is consistent. It would be extremely "expensive" in terms of extra code bits (and consequently extra pins) to decrease the bus activity much more. Returning to our 8-bit data bus we saw that by using the *Bus-Invert* method with only one extra pin the maximum bus activity can be cut to 4. If it is needed to further cut it to 3 for example a *3-limited-weight code* is needed. The inequality that needs to be satisfied is [20], [21]:

$$C_{8+k}^0 + C_{8+k}^1 + C_{8+k}^2 + C_{8+k}^3 \geq 2^8 = 256 \qquad (3)$$

where k represents the extra code bits, $C_{8+k}^0 + \cdots + C_{8+k}^3$ is the number of codewords of length $8+k$ with at most 3 transitions and 2^8 is the required number of data patterns. By simple inspection it can be seen that the minimum k that satisfies (3) is $k = 4$. So 3 more bits are needed compared to the *Bus-Invert* method for a small extra decrease. In general the number of necessary extra code bits grows exponentially [21] and the method becomes nonpractical when a very small transition activity is needed. Another big problem for the limited-weight codes is that no general algorithmic way of generating them is known yet.

C. Implementation of the Bus-Invert Method

In order to implement the *Bus-Invert* coding scheme some extra circuitry is needed on the data-path which means extra area and sometimes lower performance. For an 8 b nonpartitioned bus a possible circuit for the driver side is shown in Fig. 8. There are 16 extra XOR gates and a majority voter (5 out of 9) circuit (because the *invert* line contributes with transitions the voter must also take it into account). The bus-drivers, bus-receivers and the latches are needed for the unencoded case also and do not represent an overhead. The majority voter can be implemented digitally with a tree of full adders (Fig. 9). Thus the circuit will generate on the order of $O(n \log n)$ extra internal transitions.

If the delay and extra power consumption of the digital majority voter is considered too large an alternate analog circuit can be used. The main observation here is that the computation of the *invert* line is not critical for data integrity.

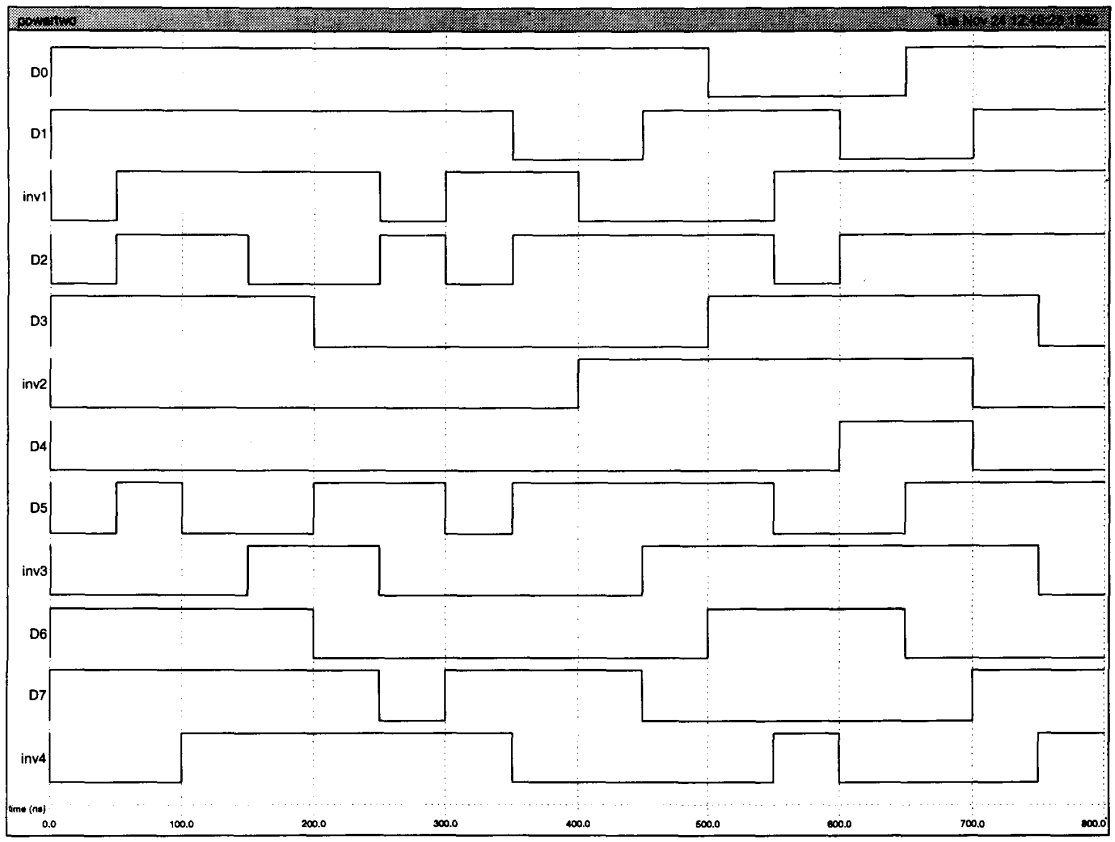


Fig. 7. An 8 b data bus partitioned into four 2 b subbuses. There are 48 transitions for a period of 16 time-slots. This represents an average of 3 transitions per time-slot, or 0.375 transitions per bus line per time-slot. The maximum number of transitions in any time-slot is 4. There is at most one transition per time-slot in each of the four subbuses.

If for example the Hamming distance is larger than $n/2$ but the majority voter decides that *invert* should be a 0, the data on the bus will not be inverted. This will only mean that the potential power savings will be lost but still the data will not be corrupted. With this observation the simpler analog circuit in Fig. 9 can be used [21] without fearing the possible "impreciseness" of such analog circuitry.

The receiver side is much simpler because it only needs to conditionally invert the bus contents in order to get the correct data value. The delay of the data-path is slightly increased but this tradeoff of performance versus low-power is common to almost all methods of decreasing power dissipation. If so desired the encoding and decoding operations can be pipelined for keeping the overall throughput unchanged.

IV. Low Power for an Address Bus

In section III the case of a random uniformly distributed sequence of values was considered and it was claimed that this is close to what happens on a data bus. For an address bus this assumption is generally no longer valid. In the extreme case of a circular FIFO implemented with a RAM and a counter [10] the RAM addresses are sequential. Generally, an address bus will tend to have a sequential behavior. Most of the time the number of transitions will be small and the overall average will be also small. For example for $n = 8$ and a pure sequential behavior the average number of transitions will be 1.99 per time-slot or only 0.25 per bus line per time-slot, much smaller than 0.5 as was the case with random uniformly distributed values. The *Bus-Invert* method will have a minimal impact on the average power dissipation in this case (1.82 average number of transitions per time-slot). The idea of *coding* can still be used in order to decrease the power dissipation for a sequential behavior.

A. Coding for Low-Power on an Address Bus

The Gray code is perfect for an address bus. With only one transition per time-slot or $1/n$ (0.125 for an 8-bit bus) transitions per bus line per time-slot it represents an improvement of 45% over the unencoded sequential case. The Gray code is easy to generate but the extra circuit will require extra area and will affect the performance. As for the *Bus-Invert* method the Gray coding can be pipelined for keeping the throughput constant. Although convenient for simplifying the theoretical analysis the model of sequential values on an address bus is generally too simplistic for a real system. Only some percentage of bus addresses are typically sequential [12] with the others being essentially random. In that case a mixed coding, Gray and *Bus-Invert* will give the best results for both peak and average power dissipation.

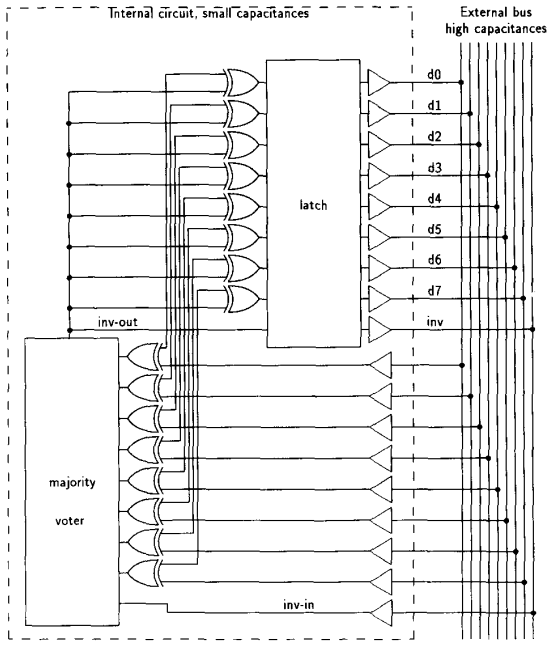


Fig. 8. Possible circuit for the driver side on an 8 b bus. There are 16 XOR gates, 8 for conditionally inverting the bus contents and 8 for comparing the present bus-value with the next data-value. The majority voter circuit decides according to the Hamming distance whether to invert or not the next value. The value of the present $invert$ line must be considered by the majority voter.

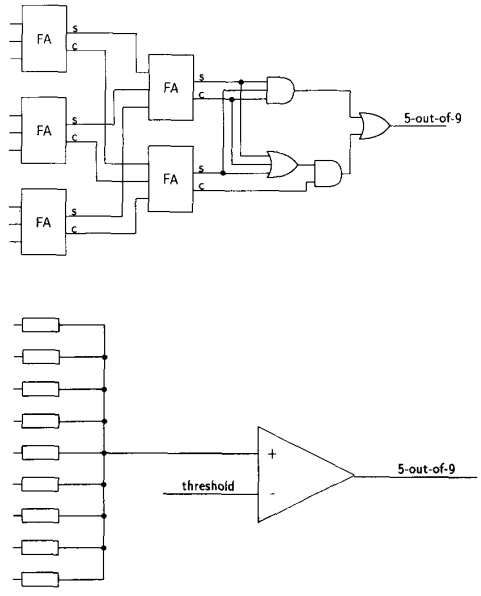


Fig. 9. Possible circuits for the 5 out of 9 majority voter. The digital voter is implemented as a tree of full-adders with the last stage of the tree simplified by taking advantage of "don't care" terms. The analog voter is simpler but less accurate and uses only resistors and a voltage comparator.

As an example of what it means to ignore the impact of design on power dissipation consider the idea of using a LFSR instead of a counter for generating addresses for a circular FIFO [19]. Because of its pseudo-random behavior the average number of transitions per bus line per time-slot for LFSR addressing will be 0.5 like in the case of a "random" data bus. This is much higher than for a counter and thus such a circuit would dissipate more power.

V. Conclusions and Future Work

Although the *Bus-Invert* method was explained in the particular setting of dynamic I/O power dissipation the same methods can be applied in any case where large capacitances are involved. Examples are: multichip module (MCM) interconnections, wafer-scale integration (WSI) intermodule connections and even on-chip buses. Furthermore it is likely that the method will also reduce the total I/O overlap current (because it reduces the number of switching I/O lines) which was ignored in our analysis but which in general contributes to additional power dissipation because of the large I/O loads. Special connections like the RGB connection to the display of a portable polygon render described in [22] can also use the technique. Depending on the ratio between the small (what was called "internal capacitance" in the paper) and large (what was called "I/O capacitance" in the paper) capacitances the applicability of the method can be considered in many other cases.

The proposed *Bus-Invert* method of decreasing power dissipation like other methods [3] represents a trade-off between performance and power dissipation. The performance decreases because the XOR's and the majority voter circuit increase the area and delay of the data-path. The majority voter has a delay of $O(\log n)$. The good result is that the smallest decrease in performance is obtained for $n = 2$ (e.g., for buses partitioned into 2 b subbuses) for which the lowest number of transitions is also obtained. Another trade-off is represented by the extra number of I/O pins needed. Answers to some of the above problems are as follows.

- The increase in the delay of the data-path. As was mentioned before lower performance is a common problem with all methods of decreasing power dissipation. Anyhow the *Bus-Invert* method represents an "absolute" method of decreasing power dissipation. By looking at the power-delay product which removes the effect of frequency (delay) on power dissipation, a clear improvement is obtained in the form of an absolute lower number of transitions. It is also relatively easy to pipeline the bus activity. The extra pipeline stage and the extra latency must then be considered.
- The increased number of I/O pins. As was mentioned before *ground-bounce* is a big problem for simultaneous switching in high speed designs [13]. That is why modern microprocessors use a large number of V_{dd} and *GND* pins. The *Bus-Invert* method has the side-effect of decreasing the maximum ground-bounce by approximately 50%. Thus circuits using the *Bus-Invert* method can use a lower number of V_{dd} and *GND* pins and by using the method the total number of pins might even decrease. Still a problem with the increased number of I/O pins is their additional *static* power dissipation which was ignored in our analysis.

- Both the driver and the receiver on the bus must use the *Bus-Invert* method in order to code and decode correctly the information on the bus. This means that the method must be applied at the *system* level, on all the circuits connected to the bus. An option here is to store the data already encoded in RAM in order to keep the memory subsystem unchanged, with the coding and decoding done only at the master. The power savings would be entirely obtained at write and since the sequence at read is likely to be the same as at write (e.g., for a block-oriented memory subsytem) the power savings will be obtained also at read.

In the future we plan to apply coding and in particular the *Bus-Invert* method in some practical application. For example the *Rambus* emerges as a promising bus standard [17] for very high-speed applications and is particularly attractive for I/O coding because it is relatively narrow (byte-wide), block-oriented (easy to pipeline) and proposed for portable applications. We also plan to further study the theory of limited-weight codes and in particular their relationship to error-correcting codes.

As a last comment, it is interesting to mention that the *Bus-Invert* method decreases the total power dissipation although both the *total* number of transitions increases (by counting the extra internal transitions) and the *total* capacitance increases (because of the extra circuitry). This is possible because the transitions get redistributed very nonuniformly, more on the low-capacitance side and less on the high-capacitance side. This unintuitive result is powerful and it contradicts a simple analysis where only the total number of transitions is considered without taking into account how large the node capacitances are.

ACKNOWLEDGMENT

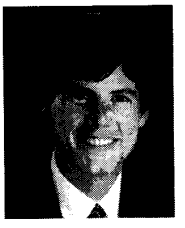
The authors would like to thank the anonymous reviewers for their constructive criticism and comments.

REFERENCES

[1] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*. Reading, MA: Addison-Wesley, 1990.
[2] T. K. Callaway and E. E. Swartzlander, "Estimating the power consumption of CMOS adders," in *11th Symp. Comp. Arithmetic*, Windsor, Ont., 1993, pp. 210-216.
[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circ.*, pp. 473-484, Apr. 1992.
[4] A. P. Chandrakasan, M. Potkonjak, J. Rabaey, and R. W. Brodersen, "HYPER-LP: A system for power minimization using architectural transformations," *ICCAD-92*, Santa Clara, CA, Nov. 1992, pp. 300-303.
[5] ——, "An approach to power minimization using transformations," *IEEE VLSI for Signal Processing Workshop*, 1992, CA.
[6] S. Devadas, K. Keutzer, and J. White, "Estimation of power dissipation in CMOS combinational circuits," in *IEEE Custom Integrated Circ. Conf.*, 1990, pp. 19.7.1-19.7.6.
[7] D. Dobberpuhl *et al.*, "A 200-MHz 64-bit dual-issue CMOS microprocessor," *IEEE J. Solid-State Circ.*, pp. 1555-1567, Nov. 1992.
[8] R. J. Fletcher, "Integrated circuit having outputs configured for reduced state changes," U.S. Patent 4,667,337, May, 1987.
[9] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis, Introduction to Chip and System Design*. Norwell, MA: Kluwer Academic Publishers, 1992.
[10] J. S. Gardner, "Designing with the IDT SyncFIFO: The architecture of the future," *1992 Synchronous (Clocked) FIFO Design Guide*, Integrated Device Technology AN-60, Santa Clara, CA, 1992, pp. 7-10.
[11] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," in *Proc. 29th DAC*, Anaheim, CA, June 1992, pp. 253-259.
[12] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*. Palo Alto, CA: Morgan Kaufmann Publishers, 1990.
[13] S. Kodical, "Simultaneous switching noise," *1993 IDT High-Speed CMOS Logic Design Guide*, Integrated Device Technology AN-47, Santa Clara, CA, 1993, pp. 41-47.
[14] F. Najm, "Transition density, a stochastic measure of activity in digital circuits," in *Proc. 28th DAC*, Anaheim, CA, June 1991, pp. 644-649.
[15] C. A. Neugebauer and R. O. Carlson, "Comparison of wafer scale integration with VLSI packaging approaches," *IEEE Trans. Components, Hybrids, and Manufact. Technol.*, pp. 184-189, June 1987.
[16] A. Park and R. Maeder, "Codes to reduce switching transients across VLSI I/O pins," *Computer Architecture News*, pp. 17-21, Sept. 1992.
[17] *Rambus-Architectural Overview*, Rambus Inc., Mountain View, CA, 1993.
[18] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer, "On average power dissipation and random pattern testability," in *ICCAD-92*, Santa Clara, CA, Nov. 1992, pp. 402-407.
[19] M. R. Stan, "Shift register generators for circular FIFOs," *Electronic Engineering*. London, England: Morgan Grampian House, Feb. 1991, pp. 26-27.
[20] M. R. Stan and W. P. Burleson, "Limited-weight codes for low power I/O," International Workshop on Low Power Design, Napa, CA, Apr. 1994.
[21] J. Tabor, "Noise reduction using low weight and constant weight coding techniques," Master's thesis, EECS Dept., MIT, May 1990.
[22] W.-C. Tan and T. H.-Y. Meng, "Low-power polygon renderer for computer graphics," *Int. Conf. A.S.A.P.*, 1993, pp. 200-213.
[23] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A Systems Perspective*. Reading, MA: Addison-Wesley Publishing Company, 1988.
[24] R. Wilson, "Low power and paradox," *Electronic Engineering Times*, pp. 38, Nov. 1, 1993.
[25] J. Ziv and A. Lempel, "A universal Algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, vol. 23, pp. 337-343, 1977.

Mircea R. Stan (M'94) received the Diploma in electronic engineering from the Polytechnic Institute of Bucharest, Romania in 1984, and the M.S. degree in electrical and computer engineering from the University of Massachusetts at Amherst in 1994.

From 1984 to 1990 he worked as an R & D Engineer at ITC - Research Institute for Computers, Bucharest, Romania. During 1991 he worked at Graphica Computer Corp., Tokyo, Japan, and in 1993 he also worked at Atis, Ltd., Atlanta, GA. Since 1991, he has been a Ph.D. student in the Department of Electrical and Computer Engineering at the University of Massachusetts at Amherst. His research interests include low-power system design, coding, mixed analog and digital VLSI, reconfigurable FPGA's and CAD for VLSI.



Wayne P. Burleson (M'89) received the B.S. and M.S. degrees in electrical engineering from MIT in 1983. He completed the Ph.D. degree in electrical and computer engineering in 1989 at the University of Colorado at Boulder.

Since 1990, he has been an Assistant Professor of Electrical and Computer Engineering at the University of Massachusetts at Amherst. From 1983-1986, he worked at VLSI Technology Inc., as a Custom Chip Designer for CMOS VLSI fax modems. He is currently developing a CAD system for VLSI arrays with applications in DSP and communications. He is also currently exploring advanced clocking schemes and verification techniques to meet the needs of modern highly pipelined architectures with high clock rates (100 Mhz and above). Additional research interests include architecture, design, CAD, and test of high-performance hardware in advanced technologies for DSP, arithmetic, coding, computer vision, robotics, and real-time computing.

Dr. Burleson received an NSF Research Initiation Award in 1991.