Limited-weight codes for low-power I/O *

Mircea R. Stan Wayne P. Burleson
ECE Dept., Univ. of Massachusetts, Amherst, MA 01003
mstan@risky.ecs.umass.edu

Abstract

Capacitances for the I/O tend to be several orders of magnitude larger than for the internal circuit. Decreasing the number of transitions at the I/O will then translate into large savings in power dissipation for the entire circuit. A way of decreasing the number of transitions at the I/O at the expense of slightly increasing the number of internal transitions is to use coding. We propose a method that uses bit-encoding with transition signaling and word-encoding with limited-weight codes. By a proper choice of the limited-weight code parameters different I/O transition activity values can be obtained. For $\frac{n}{2}$-limited-weight codes, theoretical results similar to the previously proposed Bus-Invert method are obtained.

1 Introduction

Compared to other technologies like TTL or ECL, CMOS is a low-power technology. This is especially true when one looks at the standby or static power dissipation which is practically nil for static CMOS. The problem is that because of shrinking device sizes (sub-micron), increasing die size and increasing clock frequencies (hundreds of MHz) the dynamic power dissipation increases very much and as a result many new CMOS ICs are extremely power hungry [3].

The dynamic power dissipation for a CMOS gate is [2]:

$$P_{dynamic} = C \cdot Vdd^2 \cdot f \cdot p_t$$

where C is the load capacitance, Vdd the power supply voltage, f the clock frequency and p_t is the *activity factor* or the average number of transitions per clock cycle. Usually p_t is < 1 but for poor designs p_t can be > 1 because of glitches. For the entire chip the total dynamic power dissipation will be:

$$P_{dynamic\ total} = \sum_{all\ gates} C_i \cdot Vdd_i^2 \cdot f_i \cdot p_{t_i} \quad (1)$$

*Research supported in part by NSF grant MIP-9208687

A simplification that is generally made is to consider that each gate has the same "average" load capacitance. If Vdd and f are the same for all gates formula (1) becomes:

$$P_{dynamic\ total} = C_{average} \cdot Vdd^2 \cdot f \cdot n_t \quad (2)$$

where n_t is the *total* number of transitions per clock cycle for the entire circuit.

Vdd is a quadratic term and large power savings can be obtained by decreasing it. In the following sections it will be assumed that Vdd has already been decreased to its "optimal" value [2] and that the capacitances in the circuit have already been optimized through layout and circuit restructuring. The only way to further decrease the dynamic power dissipation will then be to decrease the activity factor.

2 Coding for low-power I/O

Methods of achieving low-power try to minimize n_t for the average load capacitance [7], but assuming a unique $C_{average}$ for the entire circuit can be a dangerous oversimplification. Typically the load capacitances for the I/O are several orders of magnitude higher than for the internal circuit. Only a few transitions saved at the I/O can save as much power as hundreds or even thousands of transitions in the internal circuit. It is then convenient to consider two different average load capacitances [8]:

- one for the internal circuit, $C_{av\ int}$,
- another one for the I/O, $C_{av\ I/O}$.

With this, formula (2) becomes:

$$P_{dynamic\ total} = P_{dynamic\ int} + P_{dynamic\ I/O}$$

where $P_{dynamic\ I/O}$ can be as low as 10% [3] and as high as 80% [4] of the total dynamic power dissipation. Typically for current ICs, $P_{dynamic\ I/O}$ is half of the total power budget [10, 11]. Because $C_{av\ I/O} \gg C_{av\ int}$ it follows then that $n_{t\ int} \gg n_{t\ I/O}$. This is also confirmed by Rent's rule which states that if the complexity of the I/O is $O(n)$, the complexity of the internal circuit will be $O(n^k)$ with $k > 1$ [1]. Significant savings in total power dissipation can be achieved by decreasing $n_{t\ I/O}$ by *coding* the I/O [8] even at the expense of slightly increasing $n_{t\ int}$ (generally by $O(n \log n)$ [8]).

The following is a numerical example for the above affirmation. If $Vdd = 1\mathrm{V}$, $f = 1\mathrm{MHz}$, $C_{av\ I/O} = 1000$, $n_{t\ I/O} = 10$, $C_{av\ int} = 10$ and $n_{t\ int} = 1000$ it follows that for the uncoded case:

$$P_{uncoded} = P_{dynamic\ int} + P_{dynamic\ I/O} = 20,000$$

If by *coding* the I/O $n_{t\ I/O}$ halves to $n_{t\ I/O} = 5$ while $n_{t\ int}$ increases by 35 (which is $\approx 10 \log 10$) $n_{t\ internal}$ $= 1000 + 35$, the total power dissipation becomes:

$$P_{coded} = 1000 \cdot 5 + 10 \cdot 1035 = 15,350 \approx \frac{3}{4} P_{uncoded}$$

.

In what follows we shall differentiate between two types of encodings:

- bit-encoding,
- word-encoding.

Bit-encoding is the way 0's and 1's are represented as electrical signals (this is sometimes called bit-modulation or signaling). For example a common bit-encoding is active-high level signaling which uses a *high-level* voltage for a 1 and a *low-level* voltage for a 0.

Word-encoding is the assignment of patterns of 0's and 1's to each *word* of information to be transmitted. A common word-encoding is the nonredundant one in which n bits are used to encode 2^n possible patterns.

The method proposed in this paper uses both bit-encoding and word-encoding in order to decrease the I/O activity and thus achieve lower power dissipation. Coding for noise-reduction at the I/O was proposed in [9]. Because transitions are the cause of both noise and power dissipation in CMOS some of our results are similar to those from [9].

2.1 Bus-Invert method

A method for coding the I/O for low-power was presented in [8]. It is a *word-encoding* method that uses redundancy in order to save power. Because [8] has not been published yet, we'll shortly present the main ideas here. The model used is an n-bit synchronous data bus on which the 2^n patterns are equally probable and succesive patterns are uncorrelated (see fig. 1 reprinted here from [8]). Latches are used at the outputs in order to avoid glitches ensuring "clean" transitions between succesive patterns.

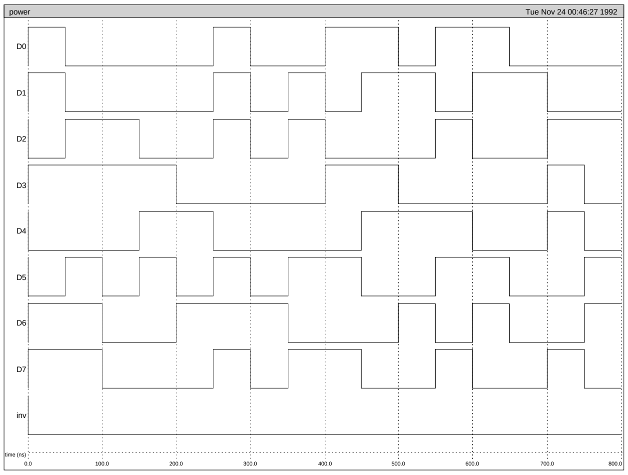

Figure 1: A typical eight-bit time-slotted data bus. The transitions between two consecutive time-slots are "clean". There are 64 transitions for a period of 16 time slots. This represents an average of 4 transitions per time slot, or 0.5 transitions per bus line per time slot. The maximum number of transitions in any time slot for this example is 6, less than the theoretical maximum of 8.

Under these assumptions the probabilities of a 0 or a 1 on the bus are each equal to $\frac{1}{2}$ and there can be between 0 and n transitions per clock cycle. There will be an average of $\frac{n}{2}$ transitions per cycle or $\frac{1}{2}$ transitions per bus line per cycle. The Bus-Invert method uses an extra bus line called *invert*. The method looks at two consecutive data patterns on the bus. If the Hamming distance [5] between the next pattern and the present one is $\leq \frac{n}{2}$ then no action is taken on the present pattern and the next value of *invert* is left 0. If the distance between the next and the present pattern is $> \frac{n}{2}$ then the next pattern on the bus gets *inverted* and the next value of *invert* is set to 1. With this method the maximum number of transitions per cycle is lowered from n to only $\frac{n}{2}$, half of the value for uncoded I/O (see fig. 2). This represents a 50% reduction in I/O dynamic *peak power* dissipation.

The average number of transitions is also lowered although by less than 25% because of the binomial distribution of the distance between consecutive patterns [8]. The Bus-Invert method is very easy to implement [8] with very good results in low-power. It would be especially interesting to include it as an optional part

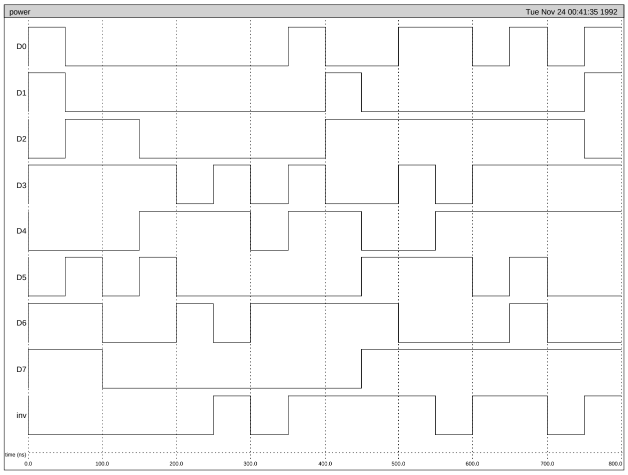

Figure 2: The same sequence of values as in fig. 1 coded using the $Bus - Invert$ method. There are now only 53 transitions. This represents an average of 3.3 transitions per time slot, or 0.41 transitions per bus line per time slot. The maximum number of transitions for any time slot is now only 4.

of a bus specification (e.g. Rambus [6]).

3 Transition encoding for low-power I/O

There can be other approaches to coding the I/O for low-power. For example one can try to use *bit-encoding* in order to save on power. Let us first assume that the probabilities of a 0 and a 1 on a bit line are not equal to $\frac{1}{2}$. Without loss of generality we shall assume that $p(1) < p(0)$ (a similar argument can be used when $p(1) > p(0)$). Under these conditions with the normal level signaling (1-high, 0-low) there will be an average of:

$$p(0) \cdot p(1) + p(1) \cdot p(0) = 2 \cdot p(1) \cdot (1 - p(1))$$

number of transitions per bus line per cycle time. As a numerical example if $p(1) = \frac{1}{4}$, $p(0) = \frac{3}{4}$ then there will be $2 \cdot \frac{1}{4} \cdot \frac{3}{4} = \frac{3}{8}$ transitions per bus line per cycle time. A pseudo-random sequence with $p(1) = \frac{1}{4}$ was generated for illustration purposes:

```
D0:1000010000100000
D1:0000100100000101
D2:0000000010011011
D3:0110010001001001
```

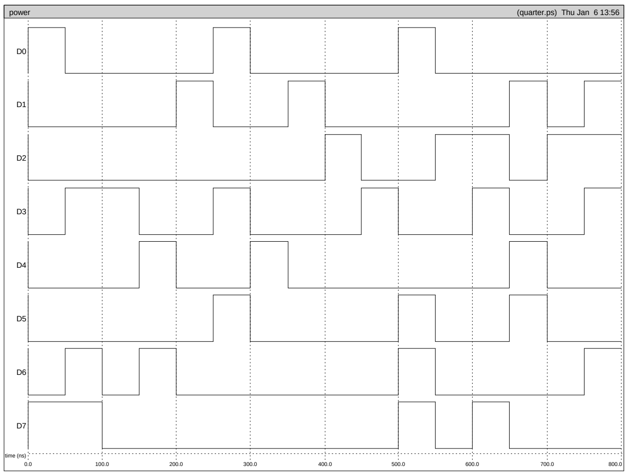

Figure 3: An 8-bit bus on which $p(1) = \frac{1}{4}$ using level signaling. There are 32 ones in a 16-cycle period (out of $16 \cdot 8 = 128$ different bit values). With *level* signaling there are 56 transitions at the I/O.

```
D4:0001001000000100
D5:0000010000100100
D6:0101000000100001
D7:1100000000101000
```

With level signaling the sequence becomes (`H` denotes a *high* level and `L` denotes a *low* level):

```
D0:HLLLLHLLLLHLLLLL
D1:LLLLHLLHLLLLLHLH
D2:LLLLLLLLHLLHHLHH
D3:LHHLLHLLLHLLHLLH
D4:LLLHLLHLLLLLLHLL
D5:LLLLLHLLLLHLLHLL
D6:LHLHLLLLLLHLLLLH
D7:HHLLLLLLLLHLHLLL
```

The corresponding waveforms can be seen in fig. 3.

Can the number of transitions be decreased by some form of encoding? In order to accomplish this we can use *transition* signaling instead of the usual *level* signaling. In this way a 1 will be encoded as a *transition* (from low to high or from high to low) and a 0 as a lack of transition. In this way the number of transitions per bus line per cycle will become only $p(1)$.

For our numerical example with $p(1) = \frac{1}{4}$, $p(0) = \frac{3}{4}$ there will be $\frac{1}{4}$ transitions per bus line per cycle or $\frac{2}{3}$ of the number of transitions with level encoding. The previous sequence using transition bit-encoding (assuming

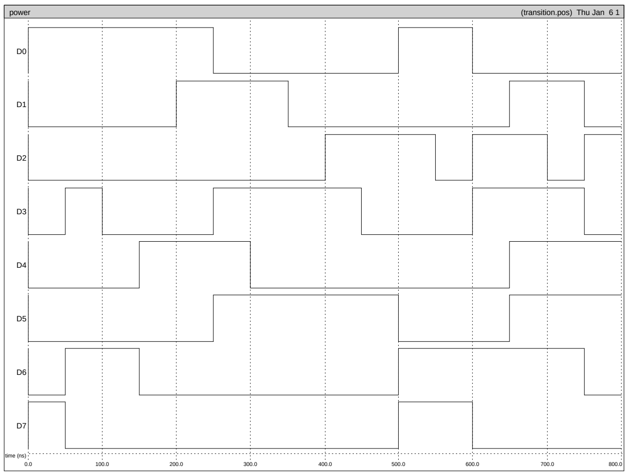

Figure 4: The same data as in figure 3 but this time using *transition* signaling. There are now only 32 transitions (instead of 56).

an initial state in which all lines are low) becomes (see also fig. 4):

```
D0:HHHHHLLLLLHHHHHH
D1:LLLLHHHLLLLLLHHL
D2:LLLLLLLLHHHLHHLH
D3:LHLLLHHHHLLLHHHL
D4:LLLHHHLLLLLLLHHH
D5:LLLLLHHHHHLLLHHH
D6:LHHLLLLLLLHHHHHL
D7:HLLLLLLLLLHHLLLL
```

3.1 Limited-weight codes for transition signaling

In order to take advantage of the savings in power possible by transition signaling the probabilities of a 0 and a 1 must be as different as possible. If $p(0) = p(1) = \frac{1}{2}$ the number of transitions per bus line per cycle for transition signaling will be $\frac{1}{2}$, the same as for the "normal" level signaling. One way of getting $p(0) \neq p(1)$ is to use *word-encoding*.

Some definitions are now necessary. The *weight* of a codeword is the number of bits equal to 1 in that word. A *constant-weight* code is a set of codewords, all having the same weight , m. We define here a *limited-weight* code as a set of codewords with weights less than or equal to some value m (in [9] these are called starvation codes). Thus a limited-weight code is in fact a collection of all the constant weight codes with weight $\leq m$. Limited-weight codes are very useful as a means of controlling $p(0)$ and $p(1)$. For a bus k-bits wide and an m-limited-weight code, if $m = k$ the nonredundant "standard" encoding for which $p(0) = p(1) = \frac{1}{2}$ is obtained. If $m = 1$ (1-limited-weight-code) $p(1) \approx \frac{1}{k}$ and $p(0) \approx \frac{k-1}{k}$.

3.2 $\frac{n}{2}$-limited-weight codes for transition encoding

Any value between 1 and k can be chosen for m with different resulting values of $p(0)$ and $p(1)$. One problem that should be apparent is that for small m (and consequently small $p(1)$ which is desirable) the number of "available" codewords gets small. For $m = k$ there will be the "normal" 2^k number of available codewords while for $m = 1$ (one-hot encoding) there will be only $k+1$ available codewords. If there are 2^n possible patterns to be transmitted over a k-wide bus ($k \geq n$) with an m-limited-weight ($m \leq k$) code a necessary condition is that the number of limited-weight codewords be $\geq 2^n$ (this relation was also derived in [9]):

$$\binom{k}{0} + \binom{k}{1} + .. + \binom{k}{m} \geq 2^n \qquad (3)$$

By starting with n (the bus-width for the nonredundant case or 2^n possible patterns), equation (3) will enable us to determine k and m for different levels of bus activity and consequently power dissipation.

For example if $k = n+1$ and $m = \frac{n}{2}$ (3) becomes an equality:

$$\binom{n+1}{0} + \binom{n+1}{1} + .. + \binom{n+1}{\frac{n}{2}} = 2^n \qquad (4)$$

Because $k = n+1$ represents a minimal redundancy > 0 and because m should be as small as possible for our purposes, it follows that there is no need to investigate the values $\frac{n}{2} < m \leq k$. An $\frac{n}{2}$-limited-weight-code is "perfect" in the sense that all possible codewords are used (because 4 is met with equality). It is also very simple to generate: by starting with a "normal" n-bit nonredundant encoding an extra $(n+1)^{st}$ bit must be added. If the "normal" encoded codeword has weight $\leq \frac{n}{2}$, then the extra bit is left 0 and appended to the original n bits to form the $\frac{n}{2}$-limited-weight codeword. If the "normal" encoded codeword has weight $> \frac{n}{2}$ then the $\frac{n}{2}$-limited-weight codeword is obtained by inverting all the original bits and by appending the extra bit $=$ 1. As can be seen this encoding method has the same "flavor" as the Bus-Invert method discussed in section 2.1. In fact, all the numerical values (n, m , k) as well as the statistical ones (maximum and average number of transitions per bus line per cycle) are exactly the same

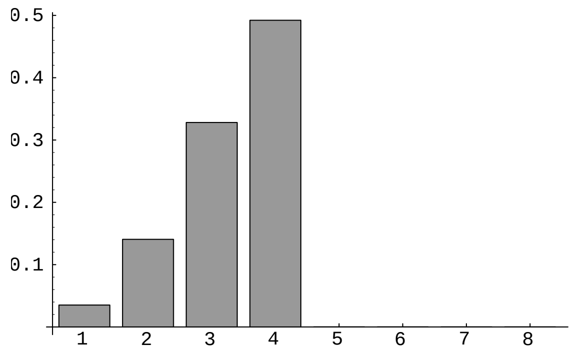

Figure 5: The probability distribution for the weight of the codewords of a 4-limited-weight code for $k = 9$.

for the Bus-Invert method with *level* signaling and the $\frac{n}{2}$-limited-weight code with *transition* signaling. All the conclusions for the Bus-Invert method [8] are also valid for $\frac{n}{2}$-limited-weight encoding:

- the maximum number of transitions per cycle is cut by half (to $\frac{n}{2}$) compared to the unencoded case
- the average number of transitions per cycle is cut by less than 25% because of the binomial distribution of patterns (see fig. 5)
- the best theoretical results in decreasing the average number of transitions per cycle (25%) are obtained for $n = 2$, $k = 3$, $m = 1$.

3.3 m-limited-weight codes

Any $1 < m < \frac{n}{2}$ could be chosen in order to further decrease the number of transitons per cycle. There are however some problems:

- for $1 < m < \frac{n}{2}$ the codes for arbitratry n, m, k are not perfect. Generally,

$$\binom{k}{0} + \binom{k}{1} + .. + \binom{k}{m} > 2^n$$

  and there will be unused codewords. There is also no obvious easy way of encoding for arbitrary n, m, k. In section 2 it was assumed that encoding the I/O does not increase *very* much the number of internal transitions. If a ROM is needed for encoding (as proposed in [9]), that condition will not be satisfied anymore.

- for small m (thus small number of transitions) k tends to increase very rapidly. For example, for n = 8 and m = 4 ($\frac{n}{2}$-limited-weight code) only one extra bit is needed, k = 8+1 = 9, but if one wants m = 3 then it is necessary that:

$$\binom{k}{0} + \binom{k}{1} + \binom{k}{2} + \binom{k}{3} \geq 2^8 = 256$$

  By inspection it can be seen that the minimum value that satisfies the above is k = 12 or extra 4 bits for only a slight decrease in the number of transitions.

3.4 1-limited-weight codes for transition encoding

From the preceeding discussion it is clear that $m = 1$ (minimum possible) has very little chance to be useful in practice except for very small n due to the exponential increase in the number of required bus lines. We mention it though because:

- it offers the absolute minimum of (slightly less than) 1 transition per cycle (it is needed to have *some* transitions in order to transmit information)
- with $k = 2^n$ - 1 it represents a "perfect" code
- the "best" $\frac{n}{2}$-limited-weight code is given by n = 2, k = 3 and m = 1 (see sect 3.2) which is also a 1-limited-weight code.

4 Implementation issues

1-limited-weight encoding (practically one-hot encoding with one less bit-line because of the 0 pattern which is encoded as all zeros) is likely to be applied in practice only for very small n. We saw its use with n = 2 and k = 3. For n = 3, k = 7 and 4 extra bus lines are necessary. For $n > 3$ k becomes perhaps too large to be applicable.

$\frac{n}{2}$-limited-weight encoding has the same performance as the Bus-Invert method but decoding is easier for the Bus-Invert method. For $\frac{n}{2}$-limited-weight encoding an *edge* detect circuit is necessary. It is then likely that circuits similar to the transition detectors used in magnetic recording peripheral devices will be necessary. Glitches or other spike-noise that may trigger the transition detector for the limited-weight coding method might be a problem.

m-limited-weight codes with $m < \frac{n}{2}$ for low-power I/O could be useful but only for particular values of m, n, k, for which an easy to encode and decode code could be found.

5 Conclusion and future work

We have presented a unified framework that uses both bit-encoding and word-encoding for developing codes for low-power I/O. Here I/O is used in the most general way and it can be module, subsystem, die, package, or multichip module (MCM) level. The only requirement is that the capacitances for the I/O be larger than the internal ones. From this point of view the techniques might be even used for data-buses "on-chip". Transition signaling together with limited-weight word-encoding offer good savings in the number of transitions for the I/O. Compared to the previously proposed Bus-Invert [8] method they offer a larger range of possible I/O transition activities. From a practical point-of-view though, the Bus-Invert method remains more attractive because:

- it uses level signaling which is easier to implement,
- for $m < \frac{n}{2}$ the extra number of bits for m-limited-weight codes becomes too large to be practical.

Coding the I/O for low-power has some drawbacks:

- it needs extra I/O pins. This may not be a problem since by reducing the maximum number of transitions on the bus the simultaneous switching noise and ground bounce [9] decrease, and the number of Vdd and GND pins can be decreased.
- the latency or throughput of the bus can increase because of the extra delay necessary for encoding. In order not to decrease the throughput an extra level of pipeline can be used at the I/O but then the latency will be increased.
- it must be implemented at the *system* level. This requirement can be somehow relaxed when using different buses for different subsystems (e.g. a separate memory bus like the Rambus [6]). The low-power methods can then be implemented gradually, e.g. first on the memory bus and then on the other buses.

The best ways to include the Bus-Invert or the limited-weight code methods in practical designs are then to:

- design an entire *chip-set* including special memory chips that use the method or,
- include it as an optional part of a bus "standard" as the Rambus [6], or,
- use it in order to decrease power dissipation for special power-hungry I/O connections like the RGB output to the display for portable applications mentioned in [10].

We are still in the first phases of our research on coding the I/O for low-power. Practical circuits that use these methods will be developed in the future, hopefully with the cooperation of interested chip manufacturers.

References

[1] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*, Addison-Wesley, 1990.

[2] A. P. Chandrakasan, S. Sheng, R. W. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, pp. 473-484, April 1992.

[3] D. Dobberpuhl et al. "A 200-MHz 64-bit Dual-Issue CMOS Microprocessor", *IEEE Journal of Solid-State Circuits*, pp. 1555-1567, Nov. 1992.

[4] C. A. Neugebauer, R. O. Carlson, "Comparison of Wafer Scale Integration with VLSI Packaging Approaches", *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, pp. 184-189, June 1987.

[5] T. R. N. Rao, E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice Hall, 1989.

[6] *Rambus - Architectural Overview*, Rambus Inc., Mountain View, CA, 1993. Contact Steven Ray , `ray@rambus.com`.

[7] A. Shen, A. Ghosh, S. Devadas, K. Keutzer, "On Average Power Dissipation and Random Pattern Testability", *ICCAD-92*, pp. 402-407, Nov. 1992, Santa Clara, CA.

[8] M. R. Stan, W. P. Burleson, "Bus-Invert method uses coding for low power I/O", submitted to *IEEE Transactions on VLSI.*

[9] J. Tabor, *Noise Reduction Using Low Weight and Constant Weight Coding Techniques*, Master's Thesis, EECS Dept., MIT, May 1990.

[10] W.-C. Tan, T. H.-Y. Meng, "Low-power polygon renderer for computer graphics", *Int. Conf. on A.S.A.P.*, pp. 200-213, 1993.

[11] R. Wilson, "Low power and paradox", *Electronic Engineering Times*, pp. 38, November 1, 1993.