Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems

Luca Benini [§] Giovanni De Micheli [§] Enrico Macii [*] Donatella Sciuto [‡] Cristina Silvano [#]

[‡] Politecnico di Milano
Dip. di Elettronica e Informazione
Milano, ITALY 20133

[*] Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

[§] Stanford University
Computer Systems Laboratory
Stanford, CA 94305

[#] Università di Brescia
Dip. di Elettronica per l'Automazione
Brescia, ITALY 25123

Abstract

In microprocessor-based systems, large power savings can be achieved through reduction of the transition activity of the on- and off-chip busses. This is because the total capacitance being switched when a voltage change occurs on a bus line is usually sensibly larger than the capacitive load that must be charged/discharged when internal nodes toggle. In this paper, we propose an encoding scheme which is suitable for reducing the switching activity on the lines of an address bus. The technique relies on the observation that, in a remarkable number of cases, patterns traveling onto address busses are consecutive. Under this condition it may therefore be possible, for the devices located at the receiving end of the bus, to automatically calculate the address to be received at the next clock cycle; consequently, the transmission of the new pattern can be avoided, resulting in an overall switching activity decrease. We present analytical and experimental analyses showing the improved performance of our encoding scheme when compared to both binary and Gray addressing schemes, the latter being widely accepted as the most efficient method for address bus encoding. We also propose power and timing efficient implementations of the encoding and the decoding logic, and we discuss the applicability of the technique to real microprocessor-based designs.

1 Introduction and Motivation

The switching activity on system-level busses is often responsible for a substantial fraction of the total power consumption for large VLSI systems. Large loads are usually connected to off-chip busses due to I/O pins, long external board wires, and board-level connected devices. In order to drive these large board-level capacitances, the sizes of the devices in the I/O pads need to be much larger than the average on-chip features, increasing also the pads intrinsic parasitic capacitance. Minimizing the switching activity on off-chip busses may thus have a sizable impact on power dissipation.

In this work we focus on microprocessor-based systems. Data and address busses are the core of the interface between a microprocessor and the external world. The increasing gap between the speed of the microprocessor and the speed of the system interface has pushed CPU designers to increase the bandwidth of the data transfers. Moreover, modern software applications span a very large address space. As a result, both data and address busses have become very wide: Existing CPUs such as the DEC Alpha AXP have a 64-bit wide address space. With very wide address and data busses the power dissipation on bus interfaces is becoming a primary concern.

Encoding techniques for limiting the number of signal transitions on the bus lines have been the subject of recent investigation. In [1], Stan and Burleson have proposed a bit encoding approach for the reduction of the average number of switchings occurring on a bus. The basic observation which has originated their work is that using a transition-based encoding instead of a level encoding may limit the number of transitions in the case of non-equiprobable input lines. The technique of [1] first encodes the data words in such a way that the probabilities of each bit become as unbalanced as possible (using *limited weight codes*), and then applies transition encoding at the bit level.

In a later work [2], Stan and Burleson have proposed the *bus-invert code*. This scheme uses redundancy to save power. If the Hamming distance between two successive patterns is larger than $N/2$ (where N is the bus width), the new pattern is transmitted with inverted polarity, thereby achieving a maximum of $N/2$ signal transitions on the bus. An extra line I is needed to signal to the receiving end of the bus which polarity is used for the transmission of the incoming pattern. If the words transmitted on the bus are independent and uniformly distributed, the average number of transitions per clock cycle is also lowered by less than 25% of the original value, due to the binomial distribution of the distance between consecutive patterns. The drawback of this approach is that it requires an extra bus line.

The encoding methods discussed so far perform well when no information about possible data correlation is available. In particular, they work fine when data patterns to be transmitted are randomly distributed in time (e.g., data exchange between a microprocessor and the data cache). Therefore, they seem to be appropriate for encoding the information traveling on data busses. This is because, except for some specific applications such as arithmetic and DSP circuits, patterns being transmitted over these busses usually have very limited correlation.

When the objective shifts to address bus encoding, a radically different behavior is observed. The addresses generated by a running microprocessor are often consecutive, since instructions are stored in adjacent sections of the memory space, and structured data are stored in consecutive memory locations for better locality. Clearly, there are exceptions to this behavior (control-flow instructions cause interruptions in the sequence of consecutive addresses on the instruction flow, and data not stored in arrays are often addressed without any regular pattern), and techniques for determining a mapping of the data to the physical memory which reduces the total switching activity on the address bus have been introduced by Panda and Dutt in [3]. In any case, we have that sequential addressing usually dominates.

1066-1395/97 $10.00 © 1997 IEEE

To exploit this unique property of address busses, Su *et al.* [4] have proposed to reduce the switching activity on communication devices of this type by adopting the *Gray code* for addresses. Gray code is particularly attractive since it guarantees single bit transitions when consecutive addresses are accessed. The results reported in [4] show that the number of bit switches is reduced by 37%, on average, on several benchmark programs when standard binary encoding is replaced by Gray encoding.

Although Gray code is suitable for reducing the switching activity, we need to consider the power overhead caused by the presence of additional circuitry for encoding and decoding. It is unrealistic to assume that the address computation units, the data-path, the memory decoders and even the compiler could be modified to generate Gray code addresses. Therefore, a Gray encoder must be placed at the transmitting end of the bus, and a Gray decoder is required at all receiving ends. In [4] the subtle trade-off between cost of encoding/decoding and the savings on the address busses is not discussed. Moreover, as it will shown later in the paper, Gray code does not achieve the minimum switching activity. In [5], some architectural solutions are proposed for the realization of the Gray addressing circuitry, and their performance are compared to the pure binary addressing in the case of a 16-bit address bus. In addition, the issue of modifying the Gray code so as to preserve the one-transition property for consecutive addresses of byte-addressable machines is extensively discussed.

In view of the discussion above, in this paper, we focus on the problem of reducing the switching activity on address busses through application of a dedicated encoding scheme. The mechanism we propose is somewhat related to the bus-invert method of [2], in the sense that both approaches rely on the addition of a redundant line to reduce the total number of transitions that may happen when streams of patterns are transmitted over the bus.

The main idea exploited by our encoding scheme, called in the sequel the *T0 code*, is that of avoiding the transfer of consecutive addresses on the bus by using a redundant line, *INC*, to transfer to the receiving sub-system the information on the sequentiality of the addresses. When two addresses in the stream to be transmitted are consecutive, the *INC* line is set to 1, the address bus lines are frozen (to avoid unnecessary switchings), and the new address is computed directly by the receiver. On the other hand, when two addresses are not consecutive, the *INC* line is driven to 0 and the bus lines operate normally.

With the hypothesis of infinite streams of consecutive addresses, the T0 code enjoys the property of zero transitions occurring on the bus. Therefore, it outperforms the Gray code since, under the same assumption, Gray addressing requires one line switching per each pair of patterns. Moreover, as it will be shown in the paper, the T0 code performs better than the Gray code even in the more realistic case of streams of consecutive addresses of limited lengths.

The increments between consecutive patterns can be parametric, reflecting the addressability scheme adopted in the given architecture. In this respect, our code has the same capabilities of the Gray scheme [5].

Although the T0 encoder and decoder are more area demanding than the Gray ones, the T0 bus interface turns out to be faster (the critical delay of a Gray decoder grows linearly with the number of bus lines to be decoded, while in the T0 decoder we propose the critical delay has logarithmic behavior). Performance of the coding-decoding scheme is essential, since in modern microprocessor-based systems bus width and clock rate are both constantly increasing.

In spite of the fact that the T0 encoding and decoding logic is more expensive, in terms of area, than the Gray one, the power savings achieved by bus encoding are not offset by the energy consumed by the additional circuitry. To support this claim, we present detailed circuit-level implementations of such additional devices, and we report power dissipation results obtained through simulation of a large set of properly selected input patterns.

In summary, in this work we address the problem of reducing the total switching activity on address busses in microprocessor-based systems. More specifically, we propose a novel encoding scheme with improved performance compared to the Gray code, and we study in detail the trade-off between coding cost and savings. We discuss the design of the encoder and the decoder and we accurately estimate their cost in terms of power, area, and timing.

2 Asymptotic Zero-Transition Encoding

Let us consider the ideal case of an infinite stream of consecutive instructions. On such stream, Gray code achieves its asymptotic best performance of 1 transition per emitted address. It appears that Gray code is the best possible for reducing the switching activity, because one bit difference is the minimum needed to distinguish two binary numbers. The key observation that allows us to improve upon this result is realizing that Gray code is optimum only for irredundant codes, that is, codes that employ exactly N-bit patterns to encode a maximum of 2^N data words. If we add redundancy to the code, we can achieve better performance.

Let us provide an additional redundant line, *INC*, to the address bus. Its purpose is to signal with value one that a consecutive stream of addresses is output on the bus. If *INC* is high, all other lines on the bus are frozen. When the redundant line is driven to zero, the remaining bus lines are used as standard binary codes for the new addresses. Obviously this redundant code outperforms the Gray code on the ideal stream of consecutive addresses. Since all addresses of the ideal stream are consecutive, the *INC* line is always high, and the bus lines never transition. As a consequence, the asymptotic performance of our code is zero transitions per emitted consecutive address.

More formally, our encoding scheme can be described as follows:

$$(\mathbf{B}^{(t)}, \mathrm{INC}^{(t)}) = \begin{cases} (\mathbf{B}^{(t-1)}, 1) & \text{if } t > 0 \wedge \mathbf{b}^{(t)} = \mathbf{b}^{(t-1)} + \mathbf{S} \\ (\mathbf{b}^{(t)}, 0) & \text{otherwise} \end{cases} \quad (1)$$

where $\mathbf{B}^{(t)}$ is the value on the encoded bus lines at time t, $INC^{(t)}$ is the additional bus line, $\mathbf{b}^{(t)}$ is the address value at time t and $\mathbf{S}$ is a constant power of 2, that we call *stride*.

The corresponding decoding scheme can be formally defined as follows:

$$\mathbf{b}^{(t)} = \begin{cases} (\mathbf{b}^{(t-1)} + \mathbf{S}) & \text{if } \mathrm{INC} = 1 \wedge t > 0 \\ \mathbf{B}^{(t)} & \text{if } \mathrm{INC} = 0 \end{cases} \quad (2)$$

Notice that the T0 code retains its zero-transition property even if the addresses are incremented by a constant stride equal to a power of two (as it is often the case for practical machines which are byte addressable, but that are able to access data or instructions aligned at word boundaries). Obviously, the stride does not correspond to the memory granularity, but to the memory word length or to the cache block size. Usually, the size of a cache block is a multiple of the word length, and typical values range from 4 to 32 bytes for first level caches, and from 2 to 256 bytes for second-level caches.

2.1 Performance of the T0 Code

We evaluate the performance of the T0 code in terms of the average number of switchings required by the transmission over the bus of different sequences of patterns. Since the code is designed specifically for patterns that satisfy, in a large number of cases, the sequentiality hypothesis, we study its behavior by encoding artificially generated streams in which out-of-sequence addresses are inserted with controlled probability.

For the experiment, streams of 100000 addresses have been generated with percentage of sequential addresses ranging from 0 to 100. The diagram of Figure 1 summarizes the results of our analysis. In particular, it clearly shows that the average number of transitions per bus line is smaller for the case of T0 addressing than for the pure binary encoding. As expected, the advantage of the T0 code becomes more remarkable as the percentage of in-sequence addresses contained into the address streams increases.

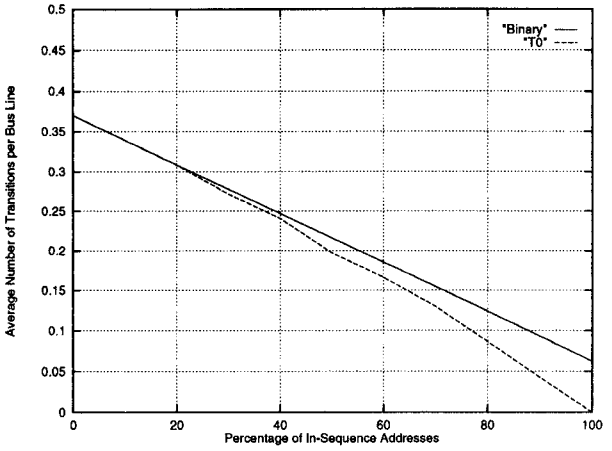

Figure 1: Performance for Artificially Generated Addresses.

The simulation of address streams generated ad-hoc substantiates the theoretical performance of the T0 code. However, in order to prove its applicability to real cases, sequences of addresses produced by real-life commercial microprocessors running complete programs must be considered.

In Table 1, we report the total number of transitions that occur on the 32-bit address bus of the MIPS microprocessor when different benchmarks are executed. We consider three cases:

- Transitions on the instruction address bus (I);
- Transitions on the data address bus (D);
- Transitions on a multiplexed address bus (M).

As expected, substantial reductions in the switching activities are observed on the instruction address streams: The probability of sequential addresses in such streams is very high across the benchmark set, and the T0 code is very effective in exploiting this property. Quite surprisingly, consecutive addresses occur with very low probability on the data address streams. This behavior is due to the fact that references to automatic variables such as loop counters destroy the sequentiality of the address streams even if array data structures are accessed sequentially. Since the probability of sequential addressing is very low, in this case the T0 encoding provides only a marginal advantage with respect to the binary encoding. The multiplexed bus has an intermediate behavior. Although the sequentiality of the addresses on the bus is somewhat reduced by the time multiplexing and by the inherent randomness of the data addresses, still the T0 encoding reduces the bus activity by a sizable amount.

Bench	Bus	Stream Length	Seq %	No. of Transitions		Sav %
				Binary	T0	
gzip	I	118743	60	231923	140209	40
	D	34325	52	130019	95519	27
	M	153069	46	827373	731795	12
gunzip	I	63525	65	117745	80797	31
	D	14534	11	94361	93717	1
	M	78060	53	475997	420835	12
gsview	I	404230	57	677511	470221	31
	D	112618	11	543738	531126	2
	M	516849	45	3032779	2758613	9
espresso	I	1751672	60	3014854	2239400	26
	D	570749	5	3925748	3915126	1
	M	2322422	44	15127600	13942044	8
nova	I	544994	53	959912	836708	13
	D	220050	10	1381975	1368763	1
	M	765045	36	5526484	5325856	4
jedi	I	14690248	54	23145174	18521606	20
	D	6145048	13	47268786	47156168	1
	M	20835297	37	152427486	142947798	6
latex	I	700316	71	1400540	200490	86
	D	200108	0	111822	111822	0
	M	900425	55	5802850	5002780	14
matlab	I	6400325	76	12000527	5800465	52
	D	1200110	0	5400774	5400762	0
	M	7600436	64	41205779	32805641	20
oracle	I	500325	70	880547	680485	23
	D	120110	0	466530	466528	0
	M	620436	56	3979397	3299339	17

Table 1: Performance for Real Addresses.

2.2 Comparison to the Gray Code

To accurately analyze the relative performance of the T0 code with respect to the Gray code, we have developed a probabilistic model of the transition activity on the bus lines. We call q the probability of having two consecutive addresses on the bus in two successive clock cycles. Moreover, we assume that when two non-consecutive addresses are issued on the bus, on average $N/2$ bus lines make a transition. This hypothesis is somewhat pessimistic, because it is equivalent to assuming that non-consecutive addresses are uniformly distributed over the full address space. In real computer systems, jumps and branches have usually some locality (for example, they have destinations within segment boundaries), and the number of transitions on the bus will be, on average, $K \leq N/2$. However, the exact value of K is irrelevant to our analysis, and we assume $K = N/2$ in the following discussion.

The average number of transitions, N_{tr}^{Gray}, produced by the Gray code for assigned values of q and N is given by the following equation:

$$N_{tr}^{\text{Gray}}(q, N) = (1 - q)\frac{N}{2} + q \tag{3}$$

This is because there are $N/2$ transitions (on average) occurring when the addresses are non-consecutive, and only one transition taking place when the addresses are consecutive.

The model for the T0 code is slightly more complex. We can describe the behavior of the code through the two-state Markov chain shown in Figure 2.

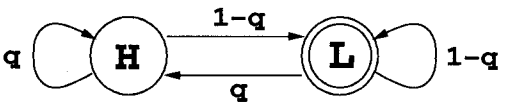

Figure 2: Markov Chain for the T0 Code.

In the following discussion, the reader is assumed to be familiar with the basic theory of Markov chains. Extensive background material can be found in [6]. State H of the chain represents the conditions for which INC is high (i.e., two consecutive addresses are sent over the bus), while state L represents the opposite case.

From the definition of the T0 code (Equation 1), state L is assumed to be the initial state. The conditional state transition probability from state L to state H is q, while it is $1-q$ if the self-loop of state L is taken. Similarly, the conditional state transition probability from state H to state L is $1-q$, while it is q if the self-loop of state H is traversed. All the conditional state transition probabilities are the edge labels of the Markov chain of Figure 2. To find the average number of transitions for the T0 code we need to compute the stationary state occupation probabilities of H and L.

The Markov chain of Figure 2 is irreducible and aperiodic [6]; therefore, we can compute the state probabilities by finding a left eigenvector of the unit eigenvalue for the transition probability matrix, P, associated to the chain:

$$\mathbf{P} = \begin{bmatrix} p_{LL} & p_{LH} \\ p_{HL} & p_{HH} \end{bmatrix} = \begin{bmatrix} (1-q) & q \\ (1-q) & q \end{bmatrix}$$

The eigenvector can be computed by solving the Chapman-Kolmogorov equations [6], here expressed in matrix form:

$$\begin{bmatrix} p_L & p_H \end{bmatrix} = \begin{bmatrix} p_L & p_H \end{bmatrix} \begin{bmatrix} (1-q) & q \\ (1-q) & q \end{bmatrix}$$

and by imposing the normality condition:

$$p_H + p_L = 1$$

The unknowns in the system of equations are the state probabilities: P_H and P_L. By solving the system we obtain $P_L = 1-q$ and $P_H = q$. Once the state probabilities are known, we can compute the total transition probabilities as follows:

$$\begin{array}{lcll} P_{LL} & = & P_L \cdot p_{LL} & = (1-q)^2 \\ P_{LH} & = & P_L \cdot p_{LH} & = (1-q)q \\ P_{HL} & = & P_H \cdot p_{HL} & = q(1-q) \\ P_{HH} & = & P_H \cdot p_{HH} & = q^2 \end{array}$$

The last step in our derivation is to obtain the average number of bus signal switchings for each arc in the Markov chain. For the LL arc we have, on average, $N/2$ transitions. For the LH arc we have one transition (the INC line goes high and all bus lines do not change value). For the HL arc we have $N/2+1$ transitions ($N/2$, on average, for the bus lines plus the falling transition for INC). Finally, no transitions are made in the HH arc. The average number of transitions for the T0 code is therefore:

$$N_{tr}^{T0}(q,N) = P_{LL}\frac{N}{2} + P_{LH} + P_{HL}(\frac{N}{2}+1) = -2q^2 + (2-\frac{N}{2})q + \frac{N}{2} \quad (4)$$

In Figure 3 we plot N_{tr}^{T0} and N_{tr}^{Gray} as a function of the probability q for a given value of N.

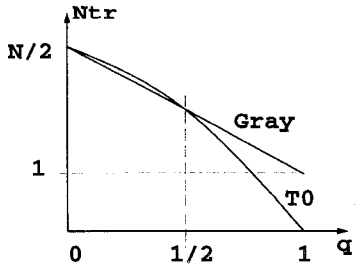

Figure 3: Comparison for the Theoretical Case.

To compare the performance of the two codes it is useful to obtain the value of q for which the two curves intersect. This can be done by solving the equation:

$$N_{tr}^{T0}(N,q) = N_{tr}^{\text{Gray}}(N,q)$$

We obtain $q = 1/2$. This is an interesting result for two reasons. First, the intersection point does not depend on N, hence the relative performance of the two codes is independent from the bus width. Second, the T0 code outperforms the Gray code when the probability of having two consecutive addresses is larger than $1/2$. As a consequence, the T0 code is convenient even if we have very short bursts of consecutive addresses. Although the performance difference is larger when $q = 1$, most address streams have $q > 1/2$.

Note that, if the worst case is considered, that is instead of N/2 transitions we substitute N transitions in equations 4 and 3, the same intersection on the two curves is obtained, i.e., $q = 1/2$, still independent of the bus width.

Experimental evidence supports the theoretical result presented above. The diagram of Figure 4 compares the average number of bus line transitions for the two encoding schemes when the address streams used to study the performance of the T0 code (see Figure 1) are supplied as input patterns.

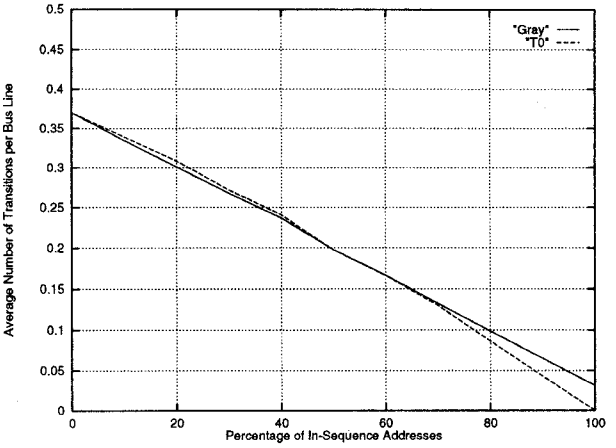

Figure 4: Comparison for Artificially Generated Addresses.

In Table 2, we compare the T0 code to the Gray code in the case of address streams produced by the MIPS microprocessor when the same benchmarks of Table 1 are used.

The data in the table show that the T0 code performs better than the Gray code for both the instruction address streams and the multiplexed address streams. For the data address streams, the two codes have similar performance, with a slight advantage of the Gray code. This result is encouraging, because it confirms the key conclusion we have drawn from our theoretical analysis: The T0 code outperforms the Gray code for those cases (i.e., streams with high values of q) when a sizable improvement is possible over the pure binary code. When the address streams are not sequential, T0, Gray and binary codes have similar performance; in this cases, the binary code is thus the right choice, since it does not require any encoding and decoding circuitry.

As a final remark, it should be noticed that for the data in the table the cross-over point of N_{tr}^{T0} and N_{tr}^{Gray} is actually a little below the value of 0.5 computed analytically, thus implying an even wider range of applications in which T0 addressing is preferable to Gray encoding.

Bench	Bus	Stream Length	Seq %	No. of Transitions		Sav %
				Gray	T0	
gzip	I	118743	60	202236	140209	31
	D	34325	52	120246	95519	21
	M	153069	46	912634	731795	20
.gunzip	I	63525	65	114054	80797	29
	D	14534	11	83460	93717	−12
	M	78060	53	415922	420835	1
gsview	I	404230	57	706200	470221	33
	D	112618	11	481030	531126	−10
	M	516849	45	3338226	2758613	17
espresso	I	1751672	60	3032322	2239400	26
	D	570749	5	3388350	3915126	−15
	M	2322422	44	14988848	13942044	7
nova	I	544994	53	1005574	836708	17
	D	220050	10	1168098	1368763	−17
	M	765045	36	6273820	5326856	15
jedi	I	14690248	54	22737020	18521606	19
	D	6145048	13	34107430	47156168	−38
	M	20835297	37	143739530	142947798	1
latex	I	700316	71	1400564	200490	86
	D	200108	0	222488	111822	50
	M	900425	55	5602624	5002780	11
matlab	I	6400325	76	10600598	5800465	45
	D	1200110	0	5805980	5400762	7
	M	7600436	64	36618100	32805641	10
oracle	I	500325	70	800574	680485	15
	D	120110	0	504802	466528	7
	M	620436	56	3399966	3299966	6

Table 2: Comparison for Real Addresses.

3 T0 Encoder and Decoder

To fully evaluate the effectiveness of the T0 code, we need to measure the cost of encoding/decoding binary addresses. In this section, we first propose architectures for the T0 encoder and decoder. Then, we discuss their implementations. Finally, we provide details about possible extensions to the case of variable strides and multiplexed busses.

3.1 Architecture

At a given clock cycle, t, the encoder computes the incremented address of cycle $t-1$ and compares it to the address generated at cycle t. If the incremented old $(t-1)$ address and the new (t) address are equal, the *INC* line is raised, and the old address is left on the bus. The encoder architecture is shown on the left of Figure 5. The incrementer can be programmable, to be able to flexibly define the constant increment **S**. The encoder inserts one cycle delay between the arrival of the address **b** and the output of the encoded bus **B**. We do not consider this delay an overhead of the encoding. Even if binary code is used (i.e., no encoding), the FFs on the output **B** would be needed because the address **b** is generated by complex logic which produces glitches and misaligned transitions. The FFs filter out glitches and align the transitions on **B** to the clock edge. Glitches on **B** must be avoided because **B** is connected to large output buffers that should always be driven by clean and fast edges, to eliminate excessive power dissipation and signal quality deterioration.

The decoder architecture is even simpler. At any given clock cycle, the last cycle's address is incremented. If the *INC* line is high, the old incremented value is used for addressing; otherwise, the value coming from the bus lines is selected. The decoder is depicted on the right of Figure 5.

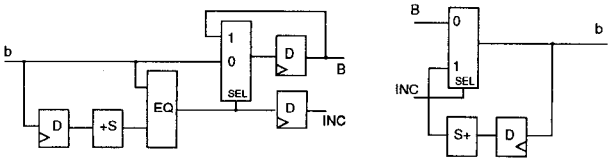

Figure 5: Block Diagrams of the T0 Code Encoder and Decoder.

The encoder/decoder implementation is optimized for minimum delay. If the speed constraints are not tight, low-power implementations should be considered. For example, it is possible to disable the incrementer in the decoder when $INC = 0$. In this case, however, the incrementer delay would be added to the critical path (starting from the late arriving signal *INC*), and performance would be penalized. We consider delay constraints, the most critical ones in high-performance microprocessors. Since the bus input **b** is produced by the complex address computation logic, it is expected to have a late arrival time. Thus, the critical path will be in the encoder from **b** through the comparator EQ and the control-to-output delay of the multiplexer (the setup time of the register controlling the bus **B** must be added to the critical path as well).

If the arrival time of **b** is not critical, the next critical path is through the incrementer, the comparator and the multiplexer. It is unlikely that this relatively simple logic will ever become the critical path of a complex microprocessor design, where much more complex tasks are usually performed in a single clock cycle. Consequently, we will discuss the possibility that the encoder could constrain the critical path because of the delay from the late arriving **b** to the output of the multiplexer (going through the comparator).

Fortunately, the combination of the T0 encoder and decoder is very fast on the critical path: The comparator can be implemented with an XOR tree structure (which has a logarithmic delay $D_{cmp} = K_{cmp}log(N)$) and the delay through a multiplexer is weakly dependent on the width of the bus. The dependence is due to the load on the control input which increases linearly with the width of the bus. If we drive the control input (SEL) with a tapered buffer, the delay has a logarithmic dependence on the bus width: $D_{mux} = K_{mux}log(N)$. In the formulas for D_{cmp} and D_{mux} the constants K_{cmp} and K_{mux} are technology dependent. The incrementers in the encoder and decoder are not strongly timing constrained, thus we can implement them in a power-efficient fashion, as long as their delays do not become critical.

Compared to the Gray encoder and decoder [5], our architectures are more area and power consuming. However, the performance of the Gray scheme is limited by the decoder (implemented as a chain of EXOR gates [5]), which has a delay $D_{gray} = K_{gray}N$. For wide busses in performance-constrained systems, the delay penalty of Gray addressing may be simply unacceptable, leaving the T0 code as the only alternative to standard binary encoding. For purely power-constrained systems, the designer's choice will be based on the trade-off between the additional power savings on the bus provided by the T0 code and the reduced power dissipation of the Gray encoder/decoder. Gray code would probably be the preferred choice for area-constrained systems where power dissipation is a secondary concern.

3.2 Implementation

The encoder and decoder architectures described in the previous section have been specified in Verilog HDL at the RT level, simulated for functional verification and a prototype has been synthesized using Synopsys `Design Compiler` with the Motorola M5C library designed for operation at $3.3V$.

The path from input **b** to the output of the multiplexer has been found to have a delay of $2.8ns$. If we assume a clock cycle of $10ns$, the decoder uses less than 30% of the clock cycle. The critical path for the decoder is much shorter than the one of the encoder, since it reduces to the delay through the control input of the multiplexer.

The small number of gate delays on the critical and the logarithmic dependence of the delay from the bus width indicate that the decoder-decoder achieve good performance. However, to complete our analysis we need to evaluate the power dissipation of the encoder and the decoder. We have obtained an estimate of the power dissipation of the gate-level implementation of the encoding/decoding circuitry by simulating the synthesized blocks with the streams of addresses used to plot the diagrams of Figures 1 and 4, and by collecting data on the switching activities. We used Synopsys Design Power to correlate such switching activities to power dissipation.

Although we obtained absolute power dissipation estimates for encoder and decoder, these values are are not what we are looking for, because we are interested in the relative power of the additional interface circuitry versus the power saved on the bus by the encoding scheme. Since our final purpose is to reduce the total power dissipation, the power consumed in the encoder and the decoder must be smaller than the power saved by adopting the T0 code on the bus.

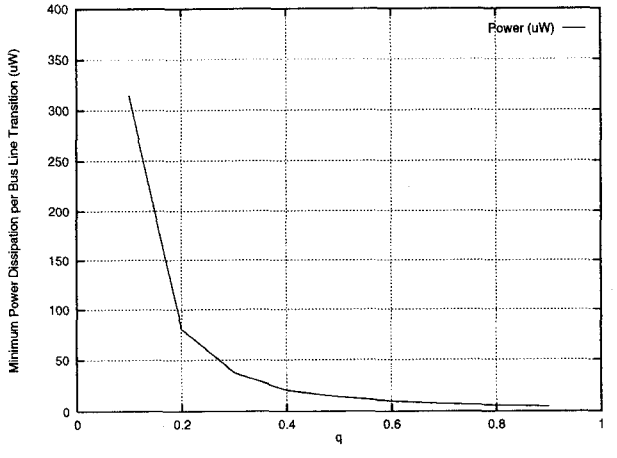

Figure 6: Practical Applicability of the T0 Code.

The trade-off between power dissipation of the encoder/decoder and the power savings on the bus lines is illustrated in Figure 6. On the abscissa of the graph we have plotted the probability of having sequential addresses on the bus (q). The ordinate is P_{tran}^{MIN}, the minimum power dissipation per bus transition, for which the power gain due to the reduced switching activity of T0 code overcomes the power dissipation of the T0 encoder and decoder. P_{tran}^{MIN} can be computed with the simple formula $P_{tran}^{MIN} = (P_{enc} + P_{dec})/N_{tpc}$, where $P_{enc} + P_{dec}$ is the power dissipation of encoder/decoder and N_{tpc} is the average number of transitions saved per clock cycle when T0 code is used (compared to binary code).

The actual minimum power values are technology dependent, therefore subject to drastic changes. However, the characteristic shape of the trade-off curve confirms the basic intuition: The T0 code is convenient only when the probability q of sequential addresses appearing on the bus is higher than a minimum technology-dependent threshold. The experiments in Sections 2.1 and 2.2 show that for both the instruction address streams and the multiplexed streams of real-life programs the value of q is well within the flat region of the trade-off curve; therefore, it may be convenient to use the T0 code. The data address streams are in the steep region of the curve, hence T0 encoding would not represent an attractive alternative to pure binary addressing.

The choice of using the T0 code for the multiplexed and instruction address streams strongly depends on the bus load and on the cleverness of the implementation of the encoder/decoder. When the circuitry is designed for maximum performance using standard cells and automatic synthesis (as it has been done for our prototypes), the T0 code becomes of interest for high loads, typical of off-chip busses. If a custom-designed optimized version of the encoding/decoding circuitry is available, the T0 code may become a viable alternative even for on-chip busses. Notice that power-efficient implementations of encoder/decoder translate the curve of Figure 6 toward lower values of P_{tran}^{MIN}, but do not alter significantly its shape. We are currently investigating custom designed low-power implementations of encoder and decoder that would make T0 code attractive even for on-chip address busses.

4 Conclusions and Future Work

In this paper we have proposed a new encoding scheme, called T0 code, which targets the minimization of the switching activity on address busses when the transmission of sequential addresses dominates.

The T0 code achieves zero-transition behavior in the theoretical case of infinite streams of in-sequence addresses. However, it provides more efficient performance than the Gray code also for short streams, under the assumption of a probability of consecutive addresses happening in successive clock cycles larger than 0.5. This conclusion has been discussed theoretically and confirmed by measurements on real address streams of programs running on a MIPS microprocessor.

The T0 code is a redundant code, since it requires an additional bus line among the communicating units to enable the data words decoding. However, the overhead is negligible if we consider the address bus width (32 or 64 bits) in current microprocessor-based systems.

For the purpose of carefully evaluating the power performance of the proposed encoding scheme, we have implemented encoding and decoding circuits, and we have analyzed their power consumption. This has allowed us to come up with some indications on whether the T0 encoding can be used through off-chip encoding/decoding interfaces. In spite of the fact that the obtained power savings were noticeable, it seems clear that the appropriate way of proceeding is to integrate the implementation of the encoding and decoding circuitry within the microprocessor and the memory controller, respectively. This may give further advantages since it may be possible to come up with more sophisticated encoders and decoders which exploit, at least in part, the existing logic already present on these chips. In particular, encoding information can be extracted directly from the microprocessor control unit, while decoding can sometimes be completely eliminated when the memory controller is driving special memory architectures, such as nibble-mode DRAMs.

References

[1] M. R. Stan, W. P. Burleson, "Limited-Weight Codes for Low-Power," IWLPD-94, pp. 209-214, Napa Valley, CA, April 1994.

[2] M. R. Stan, W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," IEEE Trans. on VLSI Systems, Vol. 3, No. 1, pp. 49-58, March 1995.

[3] P. R. Panda, N. D. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mappping," EDTC-96, pp. 63-67, Paris, France, March 1996.

[4] C. L. Su, C. Y. Tsui, A. M. Despain, "Saving Power in the Control Path of Embedded Processors," IEEE Design and Test, Vol. 11, No. 4, pp. 24-30, Winter 1994.

[5] H. Mehta, R. M. Owens, M. J. Irwin, "Some Issues in Gray Code Addressing," GLS-VLSI-96, pp. 178-180, Ames, IA, March 1996.

[6] K. S. Trivedi, Probability and Statistics with Reliability, Queueing, and Computer Science Applications, Prentice-Hall, 1982.