Memory Bus Encoding for Low Power: A Tutorial[1]

Wei-Chung Cheng and Massoud Pedram
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089
{wccheng, massoud}@sahand.usc.edu

Abstract

This paper contains a tutorial on bus-encoding techniques that target low power dissipation. Three general classes of codes, i.e., algebraic, permutation-based, and probability-based, are reviewed. A new mathematical framework for unifying the power-aware algebraic coding techniques based on the notion of leader sets is also presented.

1. Introduction

Modern electronic systems contain a dichotomy of simultaneously needing to be low power and high performance. This arises largely from their use in battery-operated portable (wearable) platforms. Even in fixed, power-rich platforms, the packaging and reliability costs associated with high power and high performance systems are forcing designers to look for ways to reduce power consumption. Power-efficient design requires reducing power dissipation in all parts of the design and during all stages of the design process subject to constraints on the system performance and quality of service (QoS). Sophisticated power-aware, high-level language compilers, dynamic power management policies, memory management, bus-encoding techniques, and hardware design tools are demanded to meet these often-conflicting design requirements. This paper focuses on the low power bus-encoding problem.

The major building blocks of a computer system include the CPU, the memory controller, the memory chips, and the communication channels dedicated to providing the means for data transfer between the CPU and the memory. These channels tend to support heavy traffic and often constitute the performance bottleneck in many systems. At the same time, the energy dissipation per memory bus access is quite high, which in turn limits the power efficiency of the overall system.

In a computer system, the bus can be an on-chip bus [20], a local bus between the CPU and the memory controller [5], or a memory bus between the memory controller (which may be on-chip or off-chip) and the memory devices. The emphasis of this paper is on low power encoding techniques for the memory bus. A widely used class of codes is the *block code* where the encoded message consists of code words *w* of fixed length (number of letters) forming a subset of all words of the same length. Each code word has a message part *m*, which can be chosen arbitrarily and a check part *c*, which is used to complete the message part to a proper code word. If an error of transmission has occurred, resulting in a word *u* which is not a code word, it is then possible to see that there has been an error, and it may be possible to locate the original code word as the closest code word to *u*. For *bit codes*, i.e., codes where the letters are 0 and 1, the distance $d(w,w')$ between two words *w* and *w'* is defined as the number of digit positions where the two words differ.

For any encoding scheme, the encoder and decoder functions are the inverse of one another; therefore, the encoding and decoding functions are usually discussed together in the literature. There is, however, a fundamental difference between the two functions. The encoder must consider the target objective (e.g., low transition count on the bus) and as a result exploits an optimization algorithm to generate the code words from the source words, whereas the decoder does a straightforward decoding of the code words without attention to why the encoding took place in one way or the other. From a mathematical point of view, it is therefore easier to explain and analyze the behavior of the decoder, rather than the encoder. This is the approach we adopted in this paper. Although we sometimes provide hints about the encoding algorithms, the emphasis is on the decoding process.

2. Low Power Coding Techniques

Generally speaking, a *code C* is a bijection *T* from a set *A* of letters to another set *B* of letters. A message *M* written with the letters of *A* is encoded by *T* to a coded message *TM* written in the letters of *B*, and the original message is recovered by applying T^{-1} to *TM*, $M=T^{-1}TM$. In a classical example, *A* and *B* consist of the same letters, and *T* is a permutation of them.

Low power bus codes can be classified as algebraic, permutation, or probabilistic. Algebraic codes refer to those codes that are produced by encoders that take two or more operands (e.g. the current source word, the previous source word, the previous code word, etc.) to produce the current code word using arithmetic or bit-level logic operations. Permutation codes refer to a permutation of a set of source words. Probabilistic codes are generated by encoders that examine the

[1] This work is sponsored in part by DARPA PAC/C program under contract number DAAB07-00-C-L516.

probability distribution of source words or pairs of source words and use this distribution to assign codes to the source words (Level signaling) or pairs of source words (Transition Signaling). In all cases, the objective is to minimize the number of transitions when transmitting all of the source words on the bus. The overhead of the encoder/decoder circuitry is often ignored.

In the remainder of this paper, we will use the following terminology. The original data to be encoded will be referred to as *source words* and denoted by s_i. The encoded data will be referred to as the *code words* and denoted by c_i. The index i refers to the time stamp of the words. Let S denote the set of all source words and C the set of all code words. We use w_s and w_c to denote the bit widths of the source and code words, respectively. In general, w_s is different from w_c .The encoder takes the source words as input and produces the code words based on some algorithm or policy. The decoder takes the code words as input and produces the source words.

2.1. Algebraic Codes

Although algebraic coding is well developed, it has seldom been mentioned in the low power bus encoding literature for the following reasons. Conventionally, algebraic coding is used for error correction in a noisy communication channel. Not only is power consideration absent in the coding theory, but there is also a need for excessive redundancy in order to perform error detection or correction. For reduced-transition bus encoding, the temporal sequence of code words needs to be considered, and coding theory helps little in this respect.

Error-correct code and information entropy, two branches of applied mathematics, have been applied to computer systems. For the conventional setup of coding theory, there are source encoding and channel encoding. Source encoding reduces the data (i.e., compresses it toward the lower bound, which is the entropy), and channel encoding inserts redundancy to correct common errors generated in the noisy channel. Because the goal of coding theory is to perform error correction and its redundancy is high, it has not been used in power-aware bus encoding.

2.1.1. Example Codes

In the following, we provide a brief explanation of a number of power-aware bus-encoding techniques. Because of space limitations, in each case we only explain how the decoder works. The encoder, of course, works in a dual manner with the decoder, but its operation is based on some optimization algorithms that would require additional explanation. For details about the encoding algorithms, please refer to the appropriate references.

Bus-Invert Code [18]: *The Bus-Invert encoding technique uses an extra signal (INV) to indicate the "polarity" of the data. Let the Bus-Invert code word be denoted as INV@x where @ is the concatenation operator, and x denotes either the source word or its one's complement. The Bus-Invert decoder takes the code word and produces the corresponding source word as follows. If the INV signal is set, the result is one's complement of x; otherwise it is x.*

Bus-Invert encoding is a simple, yet efficient technique for reducing transitions on a bus. In [11], it is proved that the Bus-Invert code minimizes the bus switching activity if the data is random and only one extra bit is available to the encoder. We refer to this type of added redundancy to a bus as a *Bus-Invert Code in Space*.

Partial Bus-Invert Code [17]: *The Partial Bus-Invert encoding technique breaks the source word into two parts u@v. Let the partial Bus-Invert code word be denoted as INV@x@y where x is the same as u and y denotes v or its one's complement. The Bus-Invert decoder takes the code word and produces the corresponding source word as follows. If the INV signal is set, the result is x concatenated with the one's complement of y; otherwise it is x@y.*

Partial Bus-Invert encoding is effective when certain bits of the source words in the data stream exhibit strong spatio-temporal correlations. The key idea is to identify such bits, group them together, and then apply the Bus-Invert coding technique only to these bits. Given the input data stream, the problem of determining this bit grouping to minimize the expected switching activity on the bus using the Partial Bus-Invert encoding is proven to be NP-complete [17].

Interleaving Partial Bus-Invert Code [21]: *It is similar to the Partial Bus-Invert Code, with the difference that the bit-width and the position of the x and y bits are dynamically changed.*

In [21], Field Programmable Gate Array (FPGA) devices are used to dynamically change the x and y bit groupings. A heuristic off-line algorithm was proposed to divide a given input stream into several sub-sequences while considering the runtime and power cost overhead of reconfiguring the encoder circuit. Note that FPGA devices are not necessary to realize this scheme. By adding extra redundant bits, we can achieve the same effect of changing the x and y bit partitions at runtime even when using a fixed encoding function.

M-bit Bus-Invert Code: *The M-bit Bus-Invert encoding technique breaks the source word into M parts* $u_1@u_2…@u_M$. *Let the M-bit Bus-Invert code word be denoted as* $INV_1@INV_2…@INV_M@x_1@x_2@…@x_M$ *where* x_i *is* u_i *or its one's complement. The M-bit Bus-Invert decoder takes the code word and produces the corresponding source word as follows. If the* INV_i *signal is set, the result is the one's complement of* x_i; *otherwise it is* x_i.

This is an obvious generalization of the Partial Bus-Invert code where there is more than one group of correlated bits, and we use different Bus-Invert signals for each one to allow independent control of the encode values for each group of bits. Note that the *M*-bit Bus-Invert reduces the number of transitions on the data bits. However, there is an extra cost associated with the transitions on the redundant *INV* bits, which may offset the reduction in the switching activity of the original set of bits.

Bus-Invert Code in Time: *This is a coding scheme in which the decoder decodes the last N code words based on an INVERT word received as the N+1st code word, that is, the i-th source word is the Bus-Invert of the i-th code word based on the i-th bit of the INVERT word.*

This technique assumes that the input data is transmitted in packets and will be decoded as a fixed-length block of source words in one shot. The redundant Bus-Invert bits for the words in a word block are added at one time as an additional word that follows the block. Thus, redundancy is added in time.

Two-dimensional Code [19]: *This is a hybrid of the Bus-Invert codes in space and time.*

Two-dimensional code uses both the spatial and temporal redundancy to enlarge the encoding space (i.e., the set E). The degree of spatial redundancy should be limited or else the increased bus width increases the implementation cost and the switching activity. Similarly, temporal redundancy imposes an extra bus cycle for every *N* bus cycles and, therefore, reduces the system performance. In [19], a signal modulation scheme is proposed to hide the extra cycle. However, the signal modulation scheme requires a dramatic change in the implementation of the physical layer.

Transition Signaling Code: *The Transition Signaling decoder produces the source word by doing an exclusive-OR operation on the current code word and the previous source word, i.e.,* $s_i=c_i \oplus s_{i-1}$ *where* $\oplus$ *denotes the XOR operation.*

This code does not introduce any redundancy to the bus (either in time or in space). However it requires the encoder to remember the previous source word. This implies the need for a memory element in the decoder circuit.

Offset Code [9]: *The Offset code is similar to the Transition Signaling code except that the Offset decoder uses the arithmetic addition (in two's complement) of the previous source word and the current code word, i.e.,* $s_i=c_i+s_{i-1}$.

The Offset code tends to reduce the dynamic range of the values transmitted on the bus, and hence it can reduce the number of bits that switch from one bus value to the next. In the extreme case of sequential data access with a fixed *stride value*, only the stride value is transmitted on the bus, i.e., we arrive at the T0 code.

T0 Code [2]: *The T0 encoding technique uses an extra signal (INC) to indicate the sequentiality of the data. Let the T0 code word be denoted as INC@x where x denotes either the current source word or is a "don't-care" word. The T0 decoder takes the code word and produces the corresponding source word as follows. If the INC signal is set, the result is the previous source word plus a stride value S* ($s_i=s_{i-1} + S$)*; otherwise it is x* ($s_i=x$).

Notice that for the increment function, the stride value is positive whereas for the decrement function, it is negative. The stride value may be fixed and known by the encoder/decoder a priori, or it may be sent on the bus as explained next. When INC is set to 1, the value on the bus is a don't-care. Normally this value is set to the previous word on the bus to minimize the bus activity. However, it may be replaced with the stride value itself. In this way, we can have variable stride values, which will reduce the number of times that the INC signal will be set to zero. However, changing the stride value on the fly will cause switching activity on the bus. One can use one-hot coding or some other technique to minimize the activity when the stride value is changed.

T0-XOR Code [9]: *The T0-XOR decoder produces the source word as follows:* $s_i= c_{i-1} \oplus c_i \oplus (s_{i-1}+S)$.

This code is a hybrid of Transition Signaling and T0 code. In contrast to the T0 code, which requires a redundant INC line, the T0-XOR code uses the XOR function to avoid the introduction of the redundant line to the bus.

Offset-XOR Code [9]: *The Offset-XOR decoder produces the source word as follows:* $s_i=(c_{i-1} \oplus c_i)+s_{i-1}$.

Again, there is no need to add a redundant bit because of the decorrelating effect of the Transition Signaling. In [9]**,** other variations of T0, Offset, and Transition Signaling codes are described.

Dual Mode Code [1]: *The Dual Mode encoding technique classifies the source word into either address or data. For the data source words, it uses the Bus-Invert code or any of its variants whereas for address source words it uses the T0 code or any of its variants.*

This encoding technique is an example of algorithms that examine the nature of the transmitted words on the bus (in this case, whether the word is address or data).

Limited-weight Code [18]: *A k-limited-weight code is a code having at most k one's per word. This can be achieved by adding appropriate redundant lines.*

These codes are useful in conjunction with transition signaling. Thus, a *k*-limited-weight code would guarantee at most *k* transitions per bus cycle.

Working-Zone Code [14]: *The Working-Zone encoding technique generates a code word as PRESENT@IDEN@OFFSET where PRESENT bit denotes a hit or miss of the working-zone, IDEN denotes the identifier for the current working-zone, and OFFSET denotes the one-hot coded offset value within that zone. The Working-zone decoder takes the code word and produces the corresponding source word as follows. If PRESENT is set to one, then the decoder produces zone(IDEN)+offset where zone(IDEN) is obtained from a look-up table relating the starting addresses of the working-zones to their identifiers, and offset is the decoded value of the OFFSET. Otherwise, the decoder outputs the current code word.*

This encoding scheme attempts to exploit the locality of reference that is usually present in the software programs. The proposed encoding technique partitions the address space into working-zones whose starting addresses are stored in a number of registers. A bit is used to denote a hit or a miss of the working-zone. When there is a miss, the full address is transmitted on the bus; otherwise, the bus is used to transmit the offset which is one-hot coded. Additional lines are used to transmit the identifiers of the working-zone. A miss of the working-zone means that either the working-zone starting address is not stored in the registers or that the offset is too large to be one-hot coded. For the case in which the number of zones is larger than the number of registers, a replacement policy is implemented. In [14], it is suggested to use Transition Signaling on the offset to further reduce the number of transitions on that portion of the bus.

Codebook-based Code [10]: *The Codebook-based encoding technique uses a code word with two parts ID@x where ID denotes the index of a pattern stored in a codebook, and x denotes either the source word or its XOR with the pattern in the codebook. The decoder produces the source word as follows:* $s_i=c_i \oplus pattern(ID)$ *where pattern(ID) refers to the codebook pattern corresponding to ID.*

The Codebook-based code can be thought of as a generalized version of the Bus-Invert code. The codebook contains the set of patterns and their corresponding ID's. The patterns are chosen so that the average Hamming distance between a source word and the "best" pattern in the codebook is minimized. The encoder compares each source word with all of the patterns in the codebook to find the pattern that has the minimum Hamming distance from the source word and then produces the code word as the concatenation of the ID bits and the XOR function of that pattern and the source word. Both the encoder and the decoder know the codebook, which can be initialized offline and/or updated online.

2.1.2. A Unifying Framework

We will define a code based on algebraic principles.

Group: A group $(S,*)$ is a set S together with a binary operation $*$ defined on S for which the following properties hold:

(1) Closure: For all $a, b \in S$, $a * b \in S$.
(2) Identity: There is an element $e \in S$ such that $e * a = a * e = a$ for all $a \in S$.
(3) Associativity: For all $a, b, c \in S$, we have $(a * b) * c = a * (b * c)$.
(4) Inversion: For each $a \in S$, there exists a unique element $b \in S$ *such that* $a * b = b * a = e$.

Code: A code *f* is a mapping from a set of symbols *S* (alphabet) to a set of binary numbers *C*. If every element in *C* has the same length, then *f* is a block code. Otherwise, *f* is a variable-length code (e.g., Huffman code).

Given the source word set *S* and the code word set *C*, one can define any mapping from *S* to *C* as a code. The purpose of encoding is to detect or correct error generated from a noisy communication channel or, in our case, to reduce transition count. The effectiveness of encoding depends on the choice of *C* and *f*.

Nearest Neighbor Decoding Algorithm: We consider a class of codes where the decoding function is based on a *Nearest Neighbor Decoding* (NND) algorithm. For this class of codes, the code words (including correctly and incorrectly received code words) are partitioned into groups where each group represents one source word (i.e., the correct intended word). A *leader* represents each group. After the decoder receives a code word, it calculates the distance between this word and each of the leaders by an XOR operation. The leader with the minimum distance is selected, and the represented source word is recognized.

Let $L \subset C$ denote a *leader set* containing some special elements *l*. The decoding process may be expressed by the following equation: $c_i * l = s_i$ where $c_i, l \in C$, $s_i \in S$ and $*$ is a binary operation over E. Note that if there exists more than one element in the leader set, the decoder decides which leader element is to be used during the decoding based on the NND algorithm. However, in many cases, the decoder is told what leader should be used by explicitly sending the information about the leader with the data.

Example 1: Consider a 4-bit bus with Bus-Invert encoding. In this case, $C = 2^5$, $S = 2^4$, $L = \{00000, 11111\}$, $*$ is the bitwise Exclusive-OR (XOR $\oplus$) operation, $c_i = 00000$, and $s_i = c_i * 11111$. During the decoding process, we use *l=11111* when INV=1, otherwise we use *l=00000*.

Using this terminology and notation, the formal statement of the Bus-Invert decoding is: $s_i = c_i \oplus l_{INV}$, where $l_0 = 00...0$, $l_1 = 11...1$.

The binary operation $*$ is used to encode and decode the code words. There are two types of operations in use:

a) XOR (): Examples include Bus-Invert and Transition Signaling codes.

b) Binary Addition (+): Examples include Working-zone and T0 codes.

Note that XOR is equivalent to the Hamming distance between the two operands and that binary addition requires an inverse operation, i.e., the binary subtraction. The XOR and binary addition/subtraction operations give rise to a group of codes that are easy to implement in practice.

Two parameters are used to characterize the leader set: size and scope.

Size

a) $|L| = 1$: For example, the Transition Signaling code implicitly uses the previous code word as the leader without introducing any redundant bits to the bus.

b) $|L| = 2$: It is necessary to add one extra bit to the bus. For example, the Bus-Invert code uses leader set $\{0, 0^{-1}\}$. Notice that the two leaders are complementary, which is, in general, not required (e.g., Partial Bus-Invert code).

c) $|L| \geq 3$: For example, the M-bit Bus-Invert and Working-zone codes use three or more leaders.

Scope

We define the *scope* of a leader as a window of size *W* where all of the source words in that window are decoded using the leader. Consider the following cases:

a) $W = \infty$: Examples are the Bus-Invert and *M*-bit Bus-Invert codes. Note that the leader set *L* is fixed.

b) *W* = *t*: Examples are the Working-zone and Dual Mode codes. Note that *L* may be fixed or adaptively changed as *t* itself may be changing. For example, with Codebook and Adaptive coding, *L* is changed on the fly.

c) W = 1: Examples are the Transition Signaling and T0 codes. Note that L={*previous-code-word*} for Transition Signaling code, and L={*previous-source-word*} for T0 code.

Comparing the sizes of E and A, we discuss the following three cases:

a) $|E| = |A|$: The code word has no redundant bits, that is if N bits are required to uniquely describe all elements of *A*, then all elements of *E* use the same number of bits. If *E=A*, then the coding is a permutation such as Gray or Pyramid code.

b) $|E| > |A|$: The code words have redundant bits. Most encoding schemes are redundant. Usually, the extra bits are used to indicate the leader. For example, the *INV* bit of the Bus-Invert code is used to select between the two leaders 0 and 0^{-1}.

c) $|E| < |A|$: Multiple source words are mapped to the same code word. In this case, the different source words are assigned to different leaders for the purpose of avoiding ambiguity in the decoding. Examples include the Working-zone and Codebook-based codes.

2.2. Permutation Codes

If redundancy is not feasible on the memory bus, the encoding function becomes a *permutation*, i.e., a one-to-one and on-to mapping from a set *S* to itself. Without any knowledge of the access pattern, the only locality that can be made use of is sequentiality. The address flow of instruction segments or large arrays is a good example. For a simple bus and sequential access, Gray code is optimal because the transition between consequent data is exactly one, the minimum.

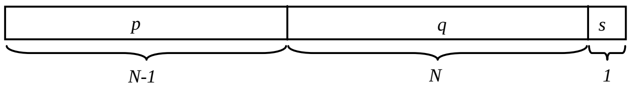
Gray Code [20]: *Only one bit difference between two consecutive words. Let the source word* $s = \langle b_{n-1,} b_{n-2,} \ldots, b_{1,} b_0 \rangle$ *and the code word* $c = \langle g_{n-1,} g_{n-2,} \ldots, g_{1,} g_0 \rangle$. *The encoding function from Binary to Gray code is* $g_n = b_n$, $g_i = b_{i+1} \oplus b_i$. *The decoding function from Gray to Binary code is* $b_n = g_n$, $b_i = b_{i+1} \oplus g_i$.

Dynamic RAM (DRAM) is omnipresent in computer systems because of its high density and low cost. The long access time of DRAM, which is its major shortcoming compared to the Static RAM (SRAM), has been significantly improved during the past few years by developing DRAM technologies such as Page Mode, EDO, Synchronous DRAM, Rambus DRAM, and DDR DRAM [8].

Because of the physical layout and the reduction in pin number, a DRAM address bus is always multiplexed. Typically, an address is divided into Row address and Column address. These two addresses are transmitted on the bus one after the other and distinguished by two control signals Row Address Strobe (RAS) and Column Address Strobe (CAS). The newer Rambus DRAM channel uses a packetized bus, which can be considered *m*-way multiplexed, and the address fields are interpreted according to the packet format [15].

Due to address multiplexing, the switching activity on the bus is generated in a completely different way. Hence, any one of the above encoding schemes either needs modification to work or does not work at all. We will present how this problem can be formulated and solved.

Pyramid Code [6]: *Let us partition a source word x into three fields p, q, and s as follows:*



The Pyramid code for x is given by:

$$\mathrm{M}(p,q,s) = \begin{cases} \langle p^s, 0 \rangle^s, p = q \\ \langle q+s, p \rangle^s, p > q \\ \langle \overline{q} + \overline{s}, \overline{p} \rangle^s, p < q \end{cases} \qquad x^s = \begin{cases} x, s = 0 \\ \overline{x}, s = 1 \end{cases} \quad \langle x, y \rangle^s = \begin{cases} \langle x, y \rangle, s = 0 \\ \langle y, x \rangle, s = 1 \end{cases}$$

It has been shown in [6] that the Pyramid code produces the minimum transition count for sequential access on a multiplexed bus. The code remains quite effective in reducing the power dissipation of the multiplexed bus even when the sequentiality of the addresses is interrupted every four addresses.

Data Ordering-based Code [12][13]: *This is a coding scheme in which the encoder takes a block of N source words and produces a block of N code words where the code words* c_i *are a permutation* π *of source word* s_j. *Both the encoder and the decoder know the permutation function* π.

For cache write-back or built-in self-test systems, changing the data flow does not affect the original semantics. Hence, a data-ordering problem can be stated to minimize the bus transitions. Given a set of data, the encoder looks for the optimal order that generates the minimum switching activities on the bus. Since the data-ordering problem is NP-complete, a bounded-error approximation algorithm was proposed in [12] to solve the offline version of this problem. To further reduce transitions, a Bus-Invert signal can be added to the Data Ordering-based code in [13].

2.3. Probabilistic Codes

Entropy-reducing Code [11]: *This code refers to a group of codes that attempt to reduce the entropy rate of the source given a fixed level of redundancy in the bus. The key idea is to compute the error between the current source word and its predicted value followed by a coding algorithm that minimizes the transition activity. The result is then sent on the bus using the Transition Signaling technique and is decoded accordingly.*

The rationale for this class of codes is that the power savings obtainable by encoding depend on the entropy rate of the incoming source data and on the amount of redundancy in the code. The higher the entropy rate, the lower the energy savings that can be achieved by encoding the source words for a specified level of redundant bits on the bus.

Beach Code [1]: *The Beach encoder analyzes the word-level correlations between source words to assign codes with small*

Hamming distance to data words that are likely to be sent on the bus in two consecutive clock cycles.

The Beach code is a subset of the entropy-reducing codes. The Beach encoder and decoder do not, however, use decorrelator and correlator blocks.

Probability-based Codes [4]: *These codes are generated based on a general codec architecture that uses encoder/decoder functions based on the current and previous values of the source and code words and decorrelator/correlator functions that implement a Transition Signaling scheme on the bus.*

These codes start with the assumption that a detailed statistical characterization of the data source is available, that is, the stationary probability distribution of all pairs of consecutive values in the input stream is known. The *Exact Encoding* function uses an exponential table (in the bit width of the bus) that stores all possible pairs of source words and their joint occurrence probability to assign a minimum of transition activity codes to each pair of source words (Transition Signaling). *Clustered Encoding* uses a spatial partitioning of the bits into groups (or clusters) of bits, which are then individually coded for minimum transition activity while considering the complete set of transition probability statistics for each cluster. *Discretized Encoding* accounts for temporal correlations between the *M* most probable source word pairs and, hence, completely accounts for the intra-word spatial correlations while ignoring some of the inter-word temporal correlations. Both encoding techniques assume a priori knowledge of the input source. *Adaptive Encoding* does not require a priori knowledge of the input source statistics. Instead, it operates on the basis of approximate information collected by observation of the input word stream over a widow of fixed size *S*.

3. Conclusions

This paper reviewed a number of bus-encoding techniques that target low power dissipation. Three general classes of codes, i.e., algebraic, permutation-based, and probability-based, were analyzed. A new mathematical framework for unifying the power-aware algebraic coding techniques based on the notion of leader sets was also presented.

4. References

[1] L. Benini, G. DeMicheli, E. Macii, M. Poncino, and S. Quer, "System-level power optimization of special purpose applications: The beach solution," *Proc. of Int'l Symp. on Low Power Electronics and Design*, Monterey, CA, pp. 24-29, Aug. 1997,

[2] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, and C. Silvano, "Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems," *Proc. of The Seventh Great Lakes Symp. on VLSI*, pp. 77-82, 1997.

[3] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, and C. Silvano, "Address bus encoding techniques for system-level power optimization," *Proc. of Design, Automation and Test in Europe*, Paris, France, pp. 861-866, Feb. 1998.

[4] L. Benini, A. Macii, E. Macii, M. Poncino and R. Scarsi, "Synthesis of low-overhead interface for power-efficient communication over wide busses," *Proc. of Design Automation Conf.*, pp. 128-133, 1999.

[5] N. Chang, K. Kim, and J. Cho, "Bus encoding for low-power high-performance memory systems," *Proc. of Design Automation Conf.*, pp. 800–805, 2000.

[6] W. C. Cheng and M. Pedram, "Power-optimal encoding for DRAM address bus," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 250-252, 2000.

[7] W. C. Cheng and M. Pedram, "Low power techniques for address encoding and memory allocation," *Proc. of Asia and South Pacific Design Automation Conference*, Jan. 2001.

[8] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary DRAM architectures," *Proc. of Int'l Symp. on Computer Architecture*, pp. 222-233, 1999.

[9] W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano, "Power optimization of system-level address buses based on software profiling," *Proc. of the Eighth Int'l Workshop on Hardware/Software Codesign*, pp. 29-33, 2000.

[10] S. Komatsu, M. Ikeda and K. Asada, "Low power chip interface based on bus data encoding with adaptive code-book method," *Proc. of the Ninth Great Lakes Symp. on VLSI*, pp. 368-371, 1999.

[11] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "A coding framework for low-power address and data busses," *IEEE Trans. on VLSI*, Vol. 7, No. 2, pp. 212-221, June 1999.

[12] R. Murgai, M. Fujita, and A. Oliveria, "Using complementation and resequencing to minimize transitions," *Proc. of Design Automation Conf.*, pp. 694-697, 1998.

[13] R. Murgai and M. Fujita, "On reducing transition through data modifications," *Proc. of Design, Automation and Test in Europe Conf.* and Exhibition, pp. 82-88, 1999.

[14] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," *Proc. of Int'l Symp. on Low Power Electronics and Design*, Monterey, CA, pp. 202-207, Aug. 1997.

[15] Rambus Inc, "Rambus Signaling Technologies: RSL, QRSL and SerDes Technology Overview," June 2000.

[16] N. R. Shanbhag, "A mathematical basis for power-reduction in digital VLSI systems," *IEEE Trans. on Circuit and Systems II: Analog and Digital Signal Processing*, Vol. 44, No. 11, pp. 935-951, Nov. 1997.

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

[18] M. R. Stan and W. P. Burleson, "Coding a terminated bus for low power," *Proc. of Fifth Great Lakes Symp. on VLSI*, pp. 70–73, 1995.

[19] M. R. Stan and W. P. Burleson, "Two-dimensional codes for low power," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 335-340, 1996.

[20] C. L. Su, C. Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processors," *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp. 24-30, 1994.

[21] S. Yoo and K. Choi, "Interleaving partial bus-invert coding for low power reconfiguration of FPGAs," *Proc. of* the *Sixth Int'l Conf. on VLSI and CAD*, pp. 549-552, *1999*.