Irredundant Address Bus Encoding for Low Power

Yazdan Aghaghiri
University of Southern California
3740 McClintock Ave
Los Angeles, CA 90089
yazdan@sahand.usc.edu

Farzan Fallah
Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94086
farzan@fla.fujitsu.com

Massoud Pedram
University of Southern California
3740 McClintock Ave
Los Angeles, CA 90089
pedram@ceng.usc.edu

ABSTRACT

This paper proposes efficient encoding techniques for decreasing power dissipation on global buses. The best target for these techniques is a wide and highly capacitive memory bus. Building on T0 and Offset-Xor encoding techniques, we present three irredundant bus-encoding techniques. Our methods decrease switching activity up to 83% without the need for redundant bus lines. The power dissipation of encoder and decoder circuitry has also been calculated and shown to be small in comparison with the power savings on the memory address bus itself.

1 INTRODUCTION

With the increasing number of transistors on a chip and the rising operation frequencies, the total power dissipation of VLSI circuits is rapidly increasing, causing high temperatures on the chip surface that can lead to a variety of reliability problems. Thus low power design methodologies are receiving more attention. Meanwhile, many systems are becoming portable and wireless, functioning on the power provided by a battery pack with a limited energy supply. Again, low power design techniques are innovated to help increase the operation time of such systems before their battery pack needs to be refurbished or recharged.

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, a local bus between the CPU and the memory controller, or a memory bus between the memory controller (which may be on-chip or off-chip) and the memory devices. The bus may be used for addresses or data. The emphasis of this paper is on encoding techniques for the memory address bus that minimize the switched capacitance of the bus.

The remainder of this paper is organized as follows. In section 2 we provide a review of previous memory bus encoding techniques. In section 3.1 the *T0-C* method, which is an optimized version of *T0*, will be presented. In section 3.2 another method, called *Offset-Xor-SM* (an optimized version of *Offset-Xor*), will be introduced. In section 3.3, *Offset-Xor-SMC*, which is an extension of *Offset-Xor-SM* will be discussed. In section 4 all of the above methods are implemented to compare their effectiveness with regard to the previous methods. The encoder blocks have also been designed and synthesized to estimate the overhead of the encoding hardware. Concluding remarks are given in the last section. [1]

ISLPED '01, August 6-7, 2001, Huntington Beach, California, USA.

2 PREVIOUS WORK

In this section we examine previous work in low power bus encoding and compare various encoding techniques. We first introduce the terminology and notation that will be used throughout this paper:

$b^{(t)}$: Address value to be sent on the bus at time t (source word at time t).

$B^{(t)}$: Encoded value on the bus lines at time t (code word at time t).

S: Stride value, which is the difference between consecutive addresses in a sequential addressing mode.

A number of encoding techniques rely on introducing redundancy to save power. More precisely, these techniques add one or more extra bits to the original bus. However, the extra bus lines cannot be tolerated in many systems because the extra bits require hardware changes and often cause incompatibility with standard bus interfaces. Consequently, a great deal of effort has been spent in finding irredundant encoding techniques that reduce the switched capacitance on the bus while preserving compatibility with existing bus interfaces and the rest of the system. In the following paragraphs, we review a number of related works on bus encoding. This is not a comprehensive review and only includes work that is directly related to our proposed encoding techniques.

In [1] Stan and Burleson proposed the *Bus-Invert* method, which is explained next. Consider an N-bit (non-multiplexed) bus. The idea is that if the Hamming distance between two consecutive patterns is larger than $N/2$, then the second pattern can be inverted so as to reduce the inter-pattern Hamming distance to below $N/2$. One redundant bit is needed to distinguish between the original and inverted patterns on the bus. The Bus-Invert method tends to perform well when sending random patterns, which is often the case on data busses.

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

However, this method is largely ineffective on address buses, which tend to exhibit a high degree of sequentiality.

In [2] Benini et al. proposed the *T0* code, which exploits data sequentiality to reduce the switching activity on the address bus. The observation is that addresses are sequential except when control flow instructions are encountered or exceptions occur. T0 adds a redundant bus line, called *INC*. If the addresses are sequential, the sender freezes the value on the bus and sets the *INC* line. Otherwise, *INC* is de-asserted and the original address is sent. On average 60% reduction in address bus switching activity is achieved by T0 coding [9]. In this paper, we propose a T0-like encoding technique for an address bus, which does not require any redundant lines. We call this new encoding technique, *T0-Concise* or *T0-C* for short.

Several methods that are basically combinations of the Bus-Invert and T0 encodings were proposed in [3]. For instance, one of the introduced methods, called T0-BI, adds two redundant bits, named *INV* and *INC,* to the bus. If the addresses are sequential, T0 coding is applied and the bus is frozen, otherwise the new address, which is not sequential, is encoded based on the Bus-Invert coding. *INC* and *INV* bits are used to correctly decode the bus value on the receiver side. The major drawback of the coding methods introduced in this work is that they introduce redundant bits. At the same time, the best reported result shows only a 40% reduction in the switching activity for the instruction address bus.

In [4] a new coding technique called *the Beach Solution* was proposed. In this method, the address trace of software is profiled, and possible correlation between different signals of the profiled trace is extracted. This information is subsequently used to define encoding functions that reduce the total switching activity. However, this method is only applicable to systems where the application programs are fixed and known a priori since the encoding technique needs exact knowledge of the address bus trace. The power savings is reported as 42%.

In [5] Musoll et al. proposed an address bus encoding method that works based on the fact that, at any time during execution, a software program uses a limited number of working zones in the address space. Thus, instead of sending the address, its offset with regards to the previous reference in the same zone along with the zone identifier is sent. One extra bit is required to notify the receiver whether this coding is in effect or the address itself is being sent.

In [6], Ikeda et al. proposed using codebooks in the sender and the receiver. For every address, the code with the minimum Hamming distance to the address is found in the codebook. Subsequently, the selected code identifier along with the Hamming distance between the address and the selected code is sent over the bus. The authors improved their method by using an adaptive codebook in [7]. Thus the codes in the codebook are replaced on the fly. As the program execution proceeds, only codes whose nearby addresses are accessed by the program, remain in the codebook.

There is another class of encoding techniques that avoid the use of redundant bits. These techniques make use of the decorrelating characteristic of the Exclusive-Or (Xor) function as follows. Since, when using Xor, the code words are transition-signaled over the bus, in every position where there is a 1 in the code word, the bus will toggle and a switching will occur. This observation can be used to cast the low power encoding problem to that of finding code words with the smallest average number of 1's in them. The most efficient one of these codes is the *T0-Xor* code, which was proposed by Fornaciari et al. in [9]. The encoder works as follows:

$$B^{(t)} = b^{(t)} \oplus (b^{(t-1)} + S) \oplus B^{(t-1)}$$

It can be easily seen that when the addresses are sequential, no switching activity occurs (similar to the case of T0 code). In the same work, the authors proposed another encoding technique, which is called *Offset-Xor* code. The encoder works as follows:

$$B^{(t)} = (b^{(t)} - b^{(t-1)}) \oplus B^{(t-1)}$$

Although not stated in [9], this encoding will become much more effective if the coding algorithm is modified as follows (resulting in a code that we will call *Offset-Xor with Stride* or *Offset-Xor-S* for short):

$$B^{(t)} = (b^{(t)} - b^{(t-1)} - S) \oplus B^{(t-1)}$$

The reason for Offset-Xor-S improvement over Offset-Xor is that it avoids switching activity when sequential addresses are encoded. One important point to notice is that sometimes, even if the difference between $b^{(t)}$ and $b^{(t-1)} + S$ is small, their Hamming distance may be quite large. This usually occurs for source words $b^{(t)}$ and $b^{(t-1)}$ that are located at opposite sides of 2^N, e.g., 61 and 69 are located at the two sides of 64. In these cases, although the offset is small, $b^{(t)} \oplus (b^{(t-1)} + S)$ contains many ones and thus causes many transitions on the bus when it is Exclusive-Or'ed with the value on the bus. We refer to this problem as the "consecutive source word Xor problem". Later on we will look at a similar case that degrades the performance of Offset-Xor-S.

Generally speaking, encodings that use transition signaling perform poorly when the code word includes many 1's. In this paper, we present a new code, called *Offset-Xor with Stride and Mapped-offset* or *Offset-Xor-SM* for short, which addresses this shortcoming by applying a mapping function to the offsets in Offset-Xor-S code.

3 LOW POWER CODES

3.1 T0-C Code

The proposed code is an extension of T0 code. It improves T0 code in a number of important ways. First of all, it eliminates the redundant bit. Second, it results in higher power saving on the bus. Similar to T0 code, the basic saving happens as a result of freezing the bus when addresses are sequential.

Suppose that we suppress the redundant bit in T0 code. In other words, when $b^{(t)}$ and $b^{(t-1)}$ are sequential addresses, we simply freeze the bus, and in all other cases, we send the original source word on the bus. This simple scheme would fail, for example, when we encounter backward branches where the

branch target address is the same as the current (frozen) bus value. Consider the following simple example:

$b^{(t)}$	$B^{(t)}$
39	39
40	39
41	39
39	39 ?

As it can be seen, when we reach the last row of the table, no valid code word can be generated for source word 39. If we use 39 as the code word, the receiver (decoder) cannot determine whether the source word was 39 (backward jump) or 42 (next sequential address). So the problem occurs when the data on the bus is equal to the branch target itself. That is why spatial redundancy was originally introduced into the T0 code. However, there is a better way to resolve this problem. To correctly handle backward branches with target addresses equal to the current bus value, a special pattern has to be sent to the receiver. However, this cannot be a fixed pattern because we assume that jumps to any and all addresses are allowed (picking any fixed pattern to designate this case may create a potentially large activity on the bus, and at the same time, requires that the fixed pattern not be used as a regular jump address).

In T0-C when such a case occurs, we set the code word to $b^{(t-1)} + S$. The reason is that this is the only pattern that the receiver should not expect from the sender. Notice that when the receiver sees a value of $b^{(t-1)} + S$, it knows that the sequential addressing has been stopped because the bus value has changed. On the other hand, when it computes the new jump address, it recognizes that this jump address is the same as the next sequential address. Therefore, if in fact a special case were not encountered, there would be no need for the sender to unfreeze the bus value. This special case is, of course, when the target of the backward jump is the same as the current value on the bus. The decoder is aware of this, and the ambiguity is resolved! To-C encoder works as follows:

```
if {b(t) == b(t-1) + S}
      B(t) = B(t-1)
else if {B(t-1) != b(t)}
      B(t) = b(t)
else
      B(t) = b(t-1) + S
```

On the receiver side, when the $b^{(t-1)} + S$ value is received, the previous value on the bus is regarded as the branch target. For the previous example we will have:

$b^{(t)}$	$B^{(t)}$
39	39
40	39
41	39
39	42
40	42

Now to make sure that this scheme works in all cases, let us consider the case when $\{b^{(t)} = b^{(t-1)}\}$. This is a jump instruction where the branch target is the branching instruction itself, that is, the instruction is waiting for an external event. Obviously, the first time this instruction iterates, $B^{(t-1)}$ is not equal to $b^{(t)}$. Therefore, because we have a simple jump in this case, we simply send $b^{(t)}$. The next time this instruction executes, the encoder recognizes it as the special case and will thus send $b^{(t-1)} + S$ on the bus. This case is illustrated below.

$b^{(t)}$	$B^{(t)}$
39	39
39	40
39	39
39	40

The T0-C code decreases switching activity on an address bus about 14% more than T0 code.

3.2 Offset-Xor-SM Code

Our objective is to improve Offset-Xor-S code by properly encoding jumps with negative offset so as to reduce the bus activity.

Based on statistics reported in [8], more than 95% of all the branches in any program have offsets that need less than 8 bits to be binary coded. If we encounter a backward jump in an instruction trace, the resulting offset will be negative. This negative number tends to have a small magnitude, and therefore, when it is encoded in two's complement form, it will contain many 1's.

In a typical application program, many small backward jumps exist, and the offsets of all these jumps are small negative numbers. Consider these offsets are to be transition-signaled over the bus, a large number of bit switching occurs on the bus because of them. For this reason, the performance (in terms of the average activity on the address bus) of the Offset-Xor and Offset-Xor-S codes is poor compared to known coding techniques such as T0. We will refer to this problem as the "small negative offset problem."

In practice, although T0-Xor and Offset-Xor are very much alike, T0-Xor code outperforms Offset-Xor code noticably [9]. This is because of the fact that the "small negative offset problem," which is the Achille's Heel of Offset-Xor code, shows up much more frequently than the "consecutive source word Xor problem," which is the key problem for T0-Xor code. Indeed, as reported in [9], the switching activity reduction for T0-Xor is 74% versus 41% for Offset-Xor.

In the following paragraphs, we describe a new coding technique (i.e., Offset-Xor-SM) to solve the "small negative offset problem."

Offset-Xor-SM encoder works as follows:

$$B^{(t)} = B^{(t-1)} \oplus \mathrm{LSBInv}((b^{(t)} - b^{(t-1)}) - S)$$

where the LSBInv(x) function inverts all bits of x except the most significant one.

In Offset-Xor-SM code, when the offset – S is positive, it is transition-signaled over the bus. Therefore, sequential addresses do not cause any activity on the bus (Offset – S = 0). However, if the Offset – S is negative, then all the bits except the MSB bit are inverted and then transition-signaled over the bus. This inversion will cause the following mapping for a typical 32-bit bus.

Original offset	Modified offset
FFFFFFFF, (-1)	80000000
FFFFFFFE, (-2)	80000001
FFFFFFF5, (-10)	80000009
80000000	FFFFFFFF

Unlike the two's complement representation, in Offset-Xor-SM small negative numbers cause only a few transitions on the bus. The extra hardware that this method imposes on Offset-Xor is negligible. With this mapping we can achieve more than 40% improvement over Offset-Xor code and about 5% improvement over T0-Xor code as they are reported in [9].[2]

3.3 Offset-Xor-SMC Code

Using a more complex encoder and decoder can decrease bus switching activity further. This method can be easily trimmed to fit to a specific application. The more capacitive the external buses are, the more complex encoding circuits can be used and the more power will be saved. In the following, we describe a new code called *Offset-Xor with Stride, Offset-mapping and Codebook* or for short *Offset-Xor-SMC* that uses a fixed codebook to reduce the number of 1's in the code words.

The idea is to embed a K-bit to K-bit mapping function (or codebook) in both the sender and the receiver sides. The K least significant bits of the output of Offset-Xor-SM are used to index into the codebook, producing a K-bit code word that will replace the original K LSB bits of the code word. In practice, we use K=10 because, based on [8], most of the branch displacements of a typical program need maximum of 10 bits to be represented. In general, K can be determined depending on the magnitude of the most frequent jumps in a program and constraints on the size of the codebook. In order to decrease the switching activity by this mapping, numbers are mapped in a manner such that smaller numbers map to numbers with few number of ones in them. If x1 and x2 are two K-bit numbers and $F(x_1)$ and $F(x_2)$ are the corresponding values from the codebook (i.e., the code words of x_1 and x_2), then F must be defined in such a way that:

$$\text{If } (x_1 < x_2) \text{ then}$$

$$\text{NumOnes}(F(x_1)) <= \text{NumOnes}(F(x_2))$$

where NumOnes(y) denotes the number of ones in binary representation of y.

Offset-Xor-SMC works as follows:

$$B^{(t)} = B^{(t-1)} \oplus CB(\text{LSBInv}(b^{(t)}-b^{(t-1)}-S))$$

CB(x) modifies the K LSB bits of the offset.

In our experiment, 10 bits are mapped by the codebook. The first code word of the codebook is 0, and the next 10 code words are 10-bit binary numbers that only have a single 1. The next 45 entries are 10-bit numbers with exactly two 1's, and so on. An important point in the actual implementation of the codebook is that if two numbers are complements of each other, their code words will also be complements of one another. This observation is used to divide the number of entries in our codebook by a factor of two and thus significantly reduces the codebook hardware overhead. Offset-Xor-SMC code yields an extra 4% saving compared to Offset-Xor-SM code.

4 IMPLEMENTATION RESULTS

To evaluate the proposed encoding techniques, we generated detailed address bus traces for a number of SPEC95 benchmarks using a simulator called *simplescalar* [10]. The SPEC95 programs were chosen primarily because precompiled codes were already available for them. For each test bench, more than 10 million addresses were generated by simulation. Then different encoding techniques were applied to measure the change in switching activity. The simulation results can be seen in Table 1. The "base case" in Table 1 refers to the total switching without encoding. Other columns show the corresponding bit-level transition counts for different encoding techniques and their percentage **reduction**.

For the set of programs that we used the switching activity saving of Offset-Xor and Offset-Xor-S is less than what has been reported in [9]. In fact as it can be seen for one of the benchmark programs, Offset-Xor-S is actually increasing the activity. On the other hand, Offset-Xor-SMC reduces the switching activity on the address bus by an average of 83.1%. If the codebook size were reduced so as to map only the 8 LSB bits, the average saving would be 81.4%.

Figure 2, 3 and 4 show the encoders for the three proposed encoding techniques. The decoders have not been shown because the decoders simply do the reverse functions on the code words to extract the source word, and therefore, are easy to construct.

To estimate the actual overhead of the above encoders circuits, first, we generated the net list of each encoder/decoder circuit in Berkeley Logic Interchange Format (BLIF). The netlists were optimized using the SIS script.rugged and mapped to a 1.5-volt, 0.18µ CMOS library using the SIS technology mapper. I/O voltage was assumed to be 3.3v. Instruction addresses of the benchmark programs were then fed into a gate-level logic simulation program named *sim-power* to estimate the power consumption of the encoders. The results for a **100 MHz** system clock are reported in Table 2. In Figure 1, percentage of total power saved versus IO capacitance per line is compared for different encoding techniques.

5 CONCLUSION

We introduced three different encoding techniques in this paper, the first two need very simple piece of hardware, the third method, although the most effective in decreasing the switching activity has more hardware overhead. The idea of putting a codebook in sender and receiver and combining this with previous methods opens a gateway to a new class of bus encoding techniques to be explored in future.

[2] The well-known sign magnitude representation is not used to solve the problem. The reason is that converting numbers represented in the two's complement to the sign-magnitude representation requires more complex hardware compared to our proposed scheme. Furthermore, the greatest negative number in two's complement form does not have any representation in the sign-magnitude form.

Table 1- Switching activity of SPEC 95 traces in millions for different codings / percentage saving

Benchmark	Base Case	T0	T0-C	Offset-Xor-S	T0-Xor	Offset-Xor-SM	Offset-Xor-SMC
Compress	7.628	1.164	0.937	6.470	0.928	1.120	0.868
	0%	84.7%	87.7%	15.1%	87.8%	85.3%	88.6%
Li	33.576	14.118	10.217	25.959	10.249	7.964	6.681
	0%	57.9%	69.5%	22.6%	69.4%	76.2%	80.1%
Go	33.480	11.934	8.779	19.102	7.930	6.194	5.146
	0%	64.3%	73.7%	42.9%	76.3%	81.4%	84.6%
M88ksim	34.868	16.729	11.816	33.806	12.523	8.527	7.195
	0%	52.0%	66.1%	3.0%	64.0%	75.5%	79.3%
Vortex	24.479	12.174	7.342	36.041	3.751	5.101	3.912
	0%	50.2%	70.0%	-47%	84.6%	79.1%	84.0%
Perl	21.458	8.003	6.088	13.957	5.691	4.489	3.920
	0%	62.7%	71.6%	34.9%	73.4%	79.0%	81.6%
Average saving	0%	62.0%	73.1%	11.9%	75.0%	79.4%	83.1%

Table 2- Encoder hardware synthesis and power estimation

	T0-Xor	T0-C	Offset-Xor-SM	Offset-Xor-SMC
Number of literals	440	767	661	2693
Area of Encoder (in thousands)	334	410	399	1043
Number of gates	306	386	379	1136
Power dissipated by encoder & decoder (uW)	266	642	740	1822

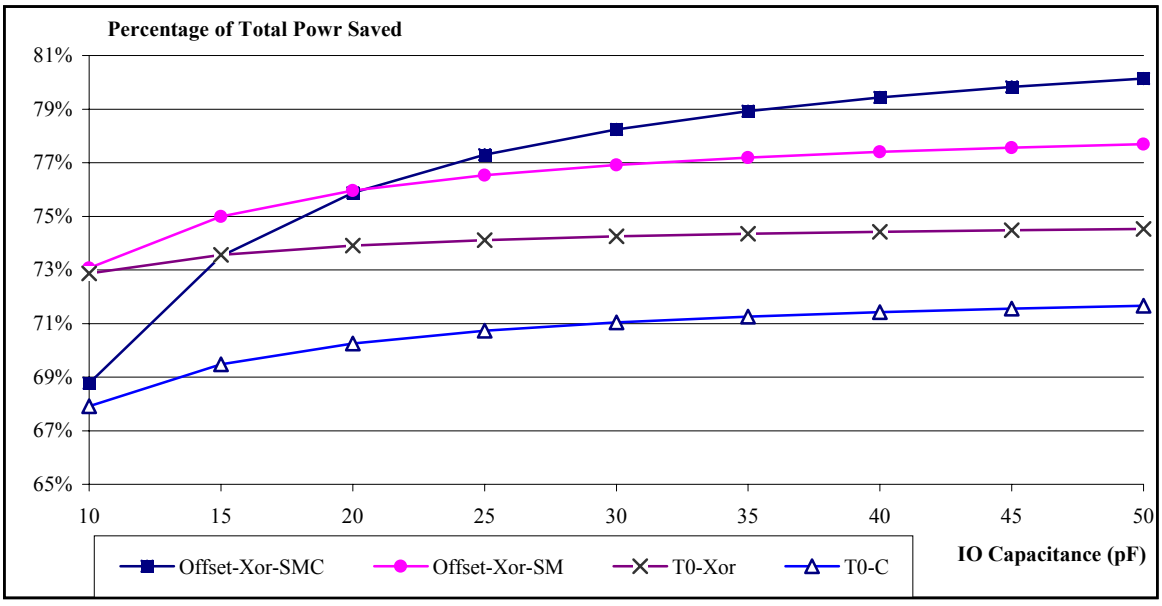

Figure 1- Comparison of total power savings of different encoding techniques

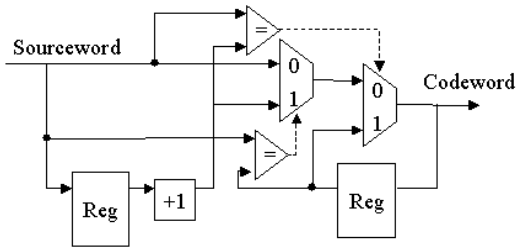

Figure 2- T0-C Encoder

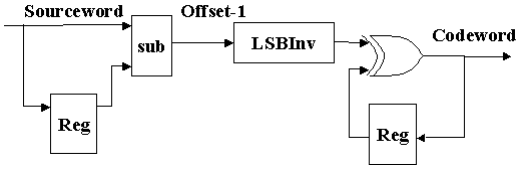

Figure 3- Offset-Xor-SM Encoder

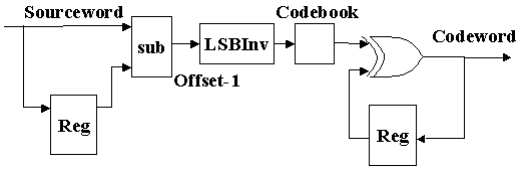

Figure 4- Offset-Xor-SMC Encoder

6 REFERENCES

1. M. R. Stan, W. P. Burleson, "Bus-Invert Coding for low-Power I/O," *IEEE Transactions on Very Large Scale Integration Systems*, Vol.3, No. 1, pp. 49-58, Mar. 1995.
2. L. Benini, G. De Micheli, E. Macii, D. Sciuto, C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Buses in Low-Power Microprocessor-Based Systems," IEEE *7th Great Lakes Symposium on VLSI,* Urbana, IL, pp. 77-82, Mar. 1997.
3. L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Address Bus Encoding Techniques for System-Level Power Optimization," *Design Automation and Test in Europe,* pp. 861-866, 1998.
4. L. Benini, G. De Michelli, E. Macii, M. Poncino, and S. Quer, "System-Level Power Optimization of Special Purpose Applications: The Beach Solution," *IEEE Symposium on Low Power Electronics and Design,* pp. 24-29, Aug. 1997.
5. E. Musoll , T. Lang, J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," *Int'l Symposium on Low Power Electronics and Design*, pp.202-207, 1997.
6. M. Ikeda, K. Asada, "Bus Data Coding with Zero Suppression for Low Power Chip Interfaces", *Int'l Workshop on Logic and Architecture Synthesis*, pp.267-274, Dec. 1996.
7. S. Komatsu, M. Ikeda, K. Asada, " Low Power Chip Interface based on Bus Data Encoding with Adaptive Code-book Method", *Ninth Great Lakes Symposium*, pp368-371, 1999.
8. Hennessy, Patterson, *Computer Architecture, A Quantitative Approach*, Second Edition, Morgan Kaufmann Publishers, 1996
9. W. Fornaciari, M. Polentarutti, D.Sciuto, and C. Silvano, "Power Optimization of System-Level Address Buses Based on Software Profiling," *CODES*, pp. 29-33, 2000.
10. http://www.simplescalar.org/.