A CLASS OF IRREDUNDANT ENCODING TECHNIQUES FOR REDUCING BUS POWER

YAZDAN AGHAGHIRI
EE-Systems, University of Southern California
3740 McClintock Ave, Los Angeles, CA 90089, United States

FARZAN FALLAH
Fujitsu Laboratories of America,
595 Lawrence Expressway, Sunnyvale, CA 94086, United States

MASSOUD PEDRAM
EE-Systems, University of Southern California
3740 McClintock Ave, Los Angeles, CA 90089, United States

This paper proposes a number of encoding techniques for decreasing power dissipation on global buses. The best target for these techniques is a wide and highly capacitive memory bus. Switching activity of the bus is reduced by means of encoding the values that are conveyed over them. More precisely, three irredundant bus-encoding techniques are presented in this paper. These techniques decrease the bus activity by as much as 86% for instruction addresses without the need to add redundant bus lines. Having no redundancy means that exercising these techniques on any existing system does not require redesign and remanufacturing of the printed circuit board of the system. The power dissipation of the encoder and decoder blocks is insignificant in comparison with the power saved on the memory address bus. This makes these techniques capable of reducing the total power consumption.

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 higher is activity over a bus, the larger is its contribution in the total power consumption. Thus, an encoding that reduces the transitions over a bus can be very effective in reducing power dissipation of the bus drivers. However, power dissipation of the encoder and decoder logic blocks should be small enough so as not to offset the reduction in power that is achieved by eliminating some of the transitions on the bus. The emphasis of this paper is on encoding techniques that minimize the switched capacitance of the memory instruction address 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 quantitative comparison of these techniques using SPEC2000 benchmark is presented. To do this, all of the above methods have been simulated to compare their performance with regards 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.

2 Previous Work

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

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

$\mathbf{B}^{(t)}$: 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 make use of introducing redundancy to save power. More precisely, these techniques add one or more extra bits to the original bus. However, in practice the extra bus lines are not desirable in many systems because the extra bits require chip interface modifications and often cause incompatibility with standards. 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 systems. 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 bus. 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 a value lower than *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 buses. However, this method is ineffective for address buses, which tend to exhibit a high degree of sequentiality. In [2], authors have investigated the effect of partitioning the bus and doing the bus-invert coding on these partitions separately. Obviously, the number of redundant lines required by this scheme is more than the original bus-invert scheme.

In [3], 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 encoding [10]. 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 combinations of the Bus-Invert and T0 encodings were proposed in [4]. 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 encoding 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 encoding 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 address bus.

In [5], a new encoding 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 [6], 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 and the zone identifier are sent. One extra bit is required to notify the receiver whether this encoding is in effect or the address itself is being sent.

In [7], 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 and the Hamming distance are sent over the bus. The authors improved their method by using an adaptive codebook in [8]. Thus, the codes in the codebook are replaced on the fly. As the program execution proceeds, only the 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. Usually, these techniques utilize 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 **one** in the code word, the bus toggles; in other positions the bus remains steady. This observation can be used to cast the low power bus-encoding problem to that of finding code words with the smallest average number of **one**'s in them. The most efficient one of these codes is the *T0-XOR* code, which was proposed by Fornaciari et al. in [10]. 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 T0 code). In the same work, the authors proposed another encoding technique, which is called *Offset-XOR*. The encoder works as follows:

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

Although not stated in [10], this encoding will become much more effective if the coding algorithm is modified as follows (resulting in a code that we 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 a power of 2, 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 **one**'s. Thus, it causes many transitions on the bus when it is XORed with the value on the bus. We refer to this problem as the "consecutive source word XOR problem". Later we will look at a similar case that degrades the performance of Offset-XOR-S.

Generally speaking, encoding techniques that exercise transition signaling perform poorly when the code word includes many **one**'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.

In [12], the authors have proposed a coding framework that is used for analyzing different bus encoding approaches. Many of the previous methods can be included in this framework.

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 a 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 example:

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

As one can see, when we reach the last row of the table, original T0 encoding will lead to ambiguity when the decoder is interpreting the values. 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 target address of the control flow instruction. That is why spatial redundancy was originally introduced into the T0 code. However, our technique presents a more efficient solution to this problem. To correctly handle backward branches with target addresses equal to the current bus value, an unusual pattern has to be sent to the receiver. By unusual we mean a pattern that can alert the receiver of the special case of target address being equal to the value of the bus. However, this cannot be a fixed pattern because we assume that jumps to any and all addresses are permissible (picking any fixed pattern to designate this case may create a potentially large activity on the bus, and at the same time, requires that particular fixed pattern not be used as a regular jump address).

The solution we adopt in T0-C is setting the code word to $b^{(t-1)} + S$. The reason is that this is the only pattern that the receiver does not expect from the sender. Notice that when the receiver sees a value of $b^{(t-1)} + S$, it recognizes that the trend of sequential addresses has been interrupted because the bus value has changed. On the other hand, when it extracts the new jump address from the bus, it detects that the jump address is the same as the next sequential address. However, receiver knows that if a special case were not encountered, there would be no need for the sender to send a new value on the bus. 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 show that this scheme works in all cases, let us consider the special 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 in the following table.

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

T0-C outperforms T0 due to the suppression of the redundant bit. The T0-C encoding decreases switching activity on an address bus by about 9% more than T0 code.

3.2 Offset-XOR-SM Code

The objective is to improve Offset-XOR-S code by properly encoding jumps with negative offset to reduce the bus activity. When we encounter a backward jump in an instruction trace, the resulting offset is negative. This negative number tends to have a small magnitude and when encoded in two's complement form, it contains many **one**'s.

In a typical application program, many small backward branches whose offsets are small negative numbers exist. Consider these offsets to be transition-signaled over the bus. A large number of bits switch on the bus because of these small negative numbers. For this reason, the performance (in terms of the average switching activity on the address bus) of the Offset-XOR and Offset-XOR-S codes is poor compared to known encoding 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 noticeably [10]. This is because the "small negative offset problem," which is the Achille's Heel of Offset-XOR code, occurs much more frequently than the "consecutive source word XOR problem," which is the similar problem for T0-XOR code. Indeed, as reported in [10], 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) is a function that inverts all bits of x except the most significant one.

In Offset-XOR-SM code, when Offset – S is not negative, it is transition-signaled over the bus. Therefore, sequential addresses do not cause any activity on the bus (Offset – S = 0). However, if Offset – S is negative, then all bits except the MSB are inverted and then transition-signaled over the bus. Some examples of this inversion for a typical 32-bit bus have been presented in the following table.

X	LSBInv(X)
00000001, (+1)	**00000001**
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 3% improvement over T0-XOR code as they are reported in [10].[1]

3.3 Offset-XOR-SMC Code

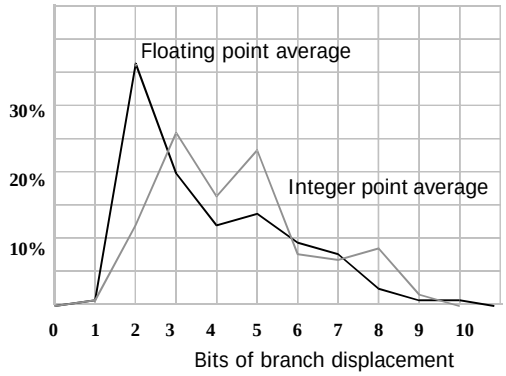

Figure 1- Percentage of branch instructions based on the required bits to represent their displacement for SPEC95 benchmarks

The bus switching activity can be reduced even further if a more complex encoder and decoder are used. In this section, we describe a new method 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 **one**'s in the code word. This decreases the number of transitions when the code word is transition signaled.

Offset-XOR-SMC uses 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 before transition signaling are used to index into the codebook, producing a K-bit code word that will replace the original K LSB bits. In practice a small K will be sufficient. This is because displacements of control flow instructions are typically small numbers. As one can see in Figure 1 more than 95% of branch displacements in SPEC95 benchmark programs can be represented with 10 bits. Therefore, in our implementation we chose K = 10. In general, K should be determined depending on the magnitude of the most frequent jumps in a program and constraints on the size of the codebook. To decrease the switching activity by this mapping, small numbers are mapped to numbers with few number of one's. If x_1 and x_2 are two K-bit numbers and $F(x_1)$ and $F(x_2)$ are their corresponding values from the codebook (i.e., the code words of x_1 and x_2), then F must be defined in 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 the binary representation of `y`.

Assuming that `CB(x)` modifies the K LSB bits of x, Offset-XOR-SMC works as follows:

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

As it was mentioned before, in our experiments, 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 with a

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

single **one**. The next 45 entries are 10-bit numbers with exactly two **one**'s, and so on. An important point in the actual implementation of the codebook is that the entries of the codebook can be organized in a fashion that if two numbers are complements of each other, their code words will also be complements of one another. Table 1 presents an example of such an organization for a three-bit codebook. This observation is used to reduce 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 3% saving in switching activity compared to Offset-XOR-SM code.

Table 1- Example of an efficient organization for a three-bit codebook

X	000	001	010	011	100	101	110	111
CB(X)	000	001	010	100	011	101	110	111

4 Experimental Results

To evaluate the proposed encoding techniques, we generated detailed instruction address bus traces for a number of SPEC2000 benchmark programs using a simulator called *Simplescalar* [13]. The SPEC2000 programs were chosen primarily because precompiled codes were already available for them. Simplescaler is an academic architecture widely used in computer architecture related research projects. Control flow instructions in this architecture are branches and different kind of jumps such as Jump, Jump and Link, etc. All other instructions are sequential instructions. For each of the benchmark programs, more than 15 million instruction addresses were generated. We first analyzed these traces to determine the contribution of each category of instructions in total number of transitions on the bus. The results are presented in Figure 2. Despite the fact that the number of transitions caused by a single sequential instruction is usually much smaller than a control flow instruction, most of the activity is due to sequential instructions. This is because of the larger number of sequential instructions in a typical program.

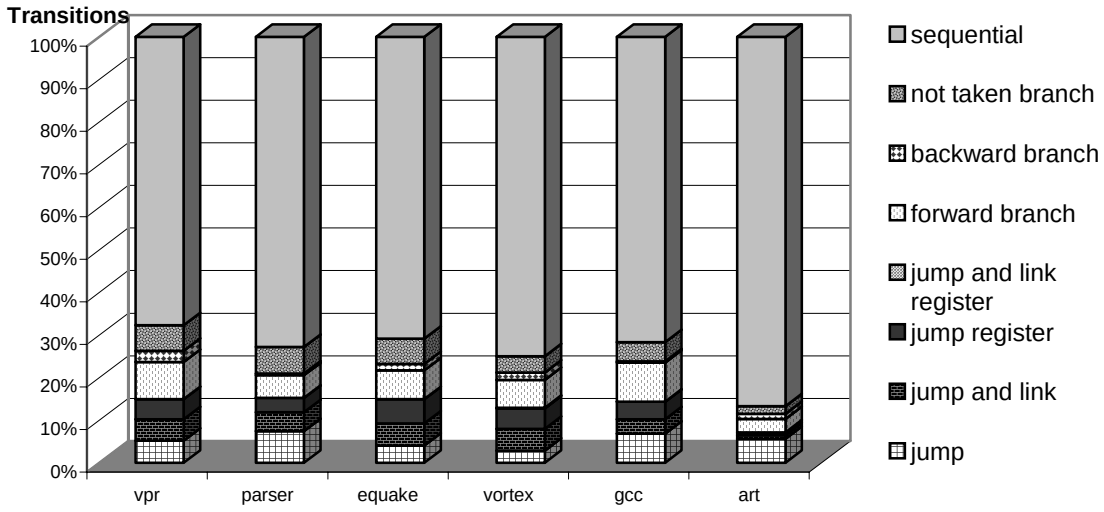

Figure 2- Contribution of different type of instructions to total number of bus transitions.

Next, different encoding techniques were applied to measure their effect in reducing switching activity. The simulation results can be seen in Table 2. The "base" in the table refers to the total transition count without any encoding. Other columns show the transition counts when different encoding techniques are applied to the addresses and also the percentage reduction in total activity.

Figure 3, 4 and 5 show the encoders for the three proposed encoding techniques. The decoders have not been shown because of their similarities with the encoders.

To estimate the actual overhead of the above encoder blocks, 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, 0.18µm 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* [11]. In this tool, dynamic power consumption based on transitions is accounted for the total power consumption.

Table 2- Switching activity of SPEC2000 traces in millions for different encoding techniques and saving percentage.

	Base	T0	T0-C	Offset-XOR-S	T0-XOR	Offset-XOR-SM	Offset-XOR-SMC
vpr	22.94	9.17	6.45	17.24	5.56	4.75	4.05
	0%	40.0%	28.1%	75.2%	24.2%	20.7%	17.7%
parser	21.27	6.02	4.17	13.61	3.79	3.13	2.39
	0%	28.3%	19.6%	64.0%	17.8%	14.7%	11.2%
equake	22.65	7.94	5.59	10.88	5.21	4.24	3.60
	0%	35.1%	24.7%	48.0%	23.0%	18.7%	15.9%
vortex	22.17	5.61	4.34	9.90	4.00	3.47	3.03
	0%	25.3%	19.6%	44.6%	18.1%	15.7%	13.7%
gcc	22.46	8.14	5.74	11.65	5.20	4.57	3.63
	0%	36.2%	25.5%	51.9%	23.1%	20.3%	16.2%
art	20.06	4.65	3.33	9.80	2.68	2.02	1.40
	0%	23.2%	16.6%	48.9%	13.3%	10.1%	7.0%
Average	0%	68.7%	77.6%	44.6%	80.1%	83.3%	86.4%

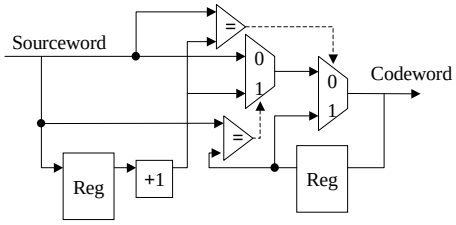

Figure 3- T0-C Encoder

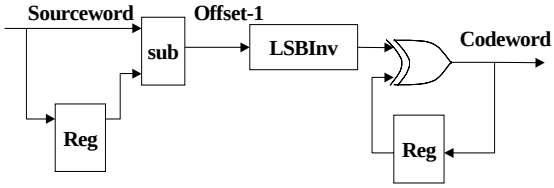

Figure 4- Offset-XOR-SM Encoder

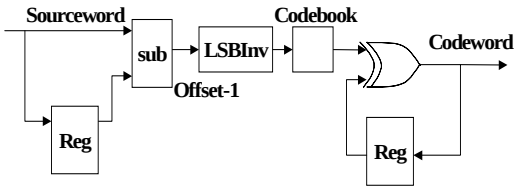

Figure 5- Offset-XOR-SM Encoder

Based on the input vectors and mapped design, estimation of the encoder power was done by sim-power. The results for a 100 MHz system clock are reported in Table 3. The actual power saved in a system depends on the capacitance of the bus lines. Depending on the line capacitance, one of the encoding techniques might perform better than others. In Figure 6, percentage of total bus power saved versus I/O capacitance per line is compared for several encoding techniques. Obviously, as the line capacitance grows, Offset-XOR-SMC outperforms the other techniques.

Table 3- 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 (μW)	266	642	740	1822

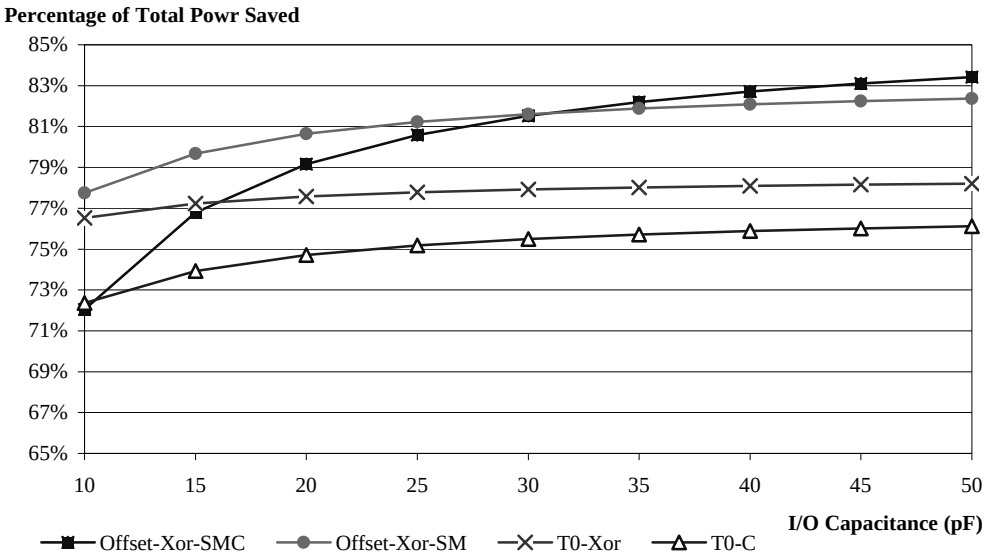

Figure 6- Comparison of total power savings for several encoding techniques.

5 Conclusion

We introduced three different encoding techniques in this paper. The first two need very simple encoders, while the third method, which is the most effective one in decreasing the switching activity, uses a more complex encoder. All the proposed techniques are redundant techniques meaning that they don't require any extra lines for encoding and decoding; therefore, they can be employed in a system with minimal redesign overhead. Our experimental results show that the power consumed in the encoders and decoders is quite smaller than the power reduction of the bus. Therefore, our techniques can reduce the actual power consumption of the systems.

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. Y. Shin, S. I. Chae, K. Choi, "Partial Bus-Invert Coding for Power Optimization of System Level Bus," *Proc. International Symposium on Low Power Electronics and Design*, pp. 127-129, Aug. 1998.
3. 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," *Proc. Seventh Great Lakes Symposium on VLSI*, pp. 77-82, Mar. 1997.
4. L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Address Bus Encoding Techniques for System-Level Power Optimization," *Proc. Design Automation and Test in Europe*, pp. 861-866, 1998.
5. L. Benini, G. De Michelli, E. Macii, M. Poncino, and S. Quer, "System-Level Power Optimization of Special Purpose Applications: The Beach Solution," *Proc. International Symposium on Low Power Electronics and Design*, pp. 24-29, Aug. 1997.
6. E. Musoll , T. Lang, J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," *Proc. International Symposium on Low Power Electronics and Design*, pp. 202-207, 1997.

7. M. Ikeda, K. Asada, “Bus Data Coding with Zero Suppression for Low Power Chip Interfaces,” *Notes of the International Workshop on Logic and Architecture Synthesis*, pp. 267-274, Dec. 1996.

8. S. Komatsu, M. Ikeda, K. Asada, “Low Power Chip Interface based on Bus Data Encoding with Adaptive Code-book Method,” *Proc. Ninth Great Lakes Symposium on VLSI*, pp. 368-371, 1999.

9. J. Hennessy and D. Patterson, *Computer Architecture, A Quantitative Approach*, Second Edition, Morgan Kaufmann Publishers, 1996

10. W. Fornaciari, M. Polentarutti, D.Sciuto, and C. Silvano, “Power Optimization of System-Level Address Buses Based on Software Profiling,” *Proc. International Symposium on Hardware/Software Codesign*, pp. 29-33, Apr. 2000.

11. S. Iman and M. Pedram, “POSE: Power Optimization and Synthesis Environment,” *Proc. 33rd Design Automation Conference*, pp. 21-26, Jun. 1996.

12. S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, “A Coding Framework for Low-Power Address and Data Buses,” *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 7, No. 2, pp. 212-221, June 1999.

13. http://www.simplescalar.org/