An Adaptive Dictionary Encoding Scheme for SOC Data Buses

Tiehan Lv, Princeton University
Jörg Henkel, NEC, USA
Haris Lekatsas, NEC, USA
Wayne Wolf, Princeton University

Abstract

As bus lengths on multi-hundred-million transistor SOCs (Systems-On-a-Chip) grow and as inter-wire capacitances of sub-0.10u technologies increase, the resulting high switching capacitances of buses (and interconnects in general) have a non-negligible impact on the power consumption of a whole SOC. In this paper, we address this problem by introducing our bus encoding technique 'ADES' that minimizes the power consumption of data buses through a dictionary-based encoding technique. We show that our technique saves between 18% and 40% of bus energy compared to the non-encoded cases using a large set of (freely-accessible) real-world applications. Furthermore, we compare our technique to the best-known data bus encoding techniques to date and it exceeds all of them in energy savings for the same set of applications. The additional hardware effort for our bus en/decoder is thereby very small.

1 Introduction

With the advent of Systems-on-Chip (SOC) that will reach one billion transistors within the next couple of years, the complexity and the physical length of bus systems/hierarchies will lead to an increased contribution to the total power consumption of an SOC. Most importantly, the closer geometrical proximity of adjacent bus lines will lead to effects that are almost negligible in technologies not as advanced as 0.10 micron and beyond. This is because two or more close bus lines form a parasitic capacitance between them. This effect not only leads to cross-talk and delay effects, but also leads to an increasing power consumption since the parasitic capacitance is charged and discharged when there is a voltage swing between two or more bus lines. Furthermore, this effect takes place *in addition* to the intrinsic capacitance of a bus line i.e. the parasitic capacitance between the bus line and various metal layers beneath. Hence, more energy is being consumed. There are several means to diminish or at least reduce the problem of inter-wire capacitances:

- Widen the distance between bus lines: typically not preferred since the total area of the bus systems grows too large.
- Use P&R tools (*place & route*) that avoid side-by-side routing of bus lines. This is what is actually done in the newest generation of P&R tools. However, the interconnect complexity of billion-transistor SOCs with multiple bus hierarchies and long buses with many cores connected to them will prevent a satisfying solution at a feasible routing time (complexity of the routing problem).
- Change the geometrical shape of bus lines: the bus lines themselves can be re-shaped. For example, the cross-sectional shape can be made narrower such that the distance between two bus lines increases *without* sacrificing space for the whole bus. However, the main disadvantage of this approach is that the cross-sectional area of a bus line is fixed since the *current-per-area* ratio is fixed for any certain technology. That typically leads to solutions where the bus lines are buried deeper into the substrate with the height being larger than the width of a bus line. However, even though the inter-wire capacitance decreases due to a decreasing distance between bus lines, it *does increase* due to the increased flank area of two opposing bus lines. In conclusion: what is won through a wider distance has to be, at least partly, given up through the effect of larger flank areas.
- Use bus-encoding techniques that take inter-wire capacitances into consideration when words are transmitted via a bus system.

We focus on the latter technique, namely on finding an energy-efficient bus encoding technique. The reason is that a bus encoding technique can be applied in addition to other techniques discussed above. We will show that our approach delivers higher energy savings than any other bus-encoding technique proposed so far. Furthermore, no a priori knowledge of the application is necessary. We conduct extensive experiments and find high energy-savings across different application domains. Targeting high correlation between adjacent bus lines, we deploy an adaptive dictionary-encoding scheme *('ADES')*. To exploit the time locality of an application, the dictionary used in our scheme is adaptive. The en/decoder can be implemented by small circuitry with low latency overheads and transistor costs. We achieve an average energy saving of 28%.

This paper is structured as follows: Section 2 introduces the most significant related work. Section 3 shows the bus energy model used. Our bus en/decoding scheme is described in Section 4. In Section 5, we compare our ADES scheme with three other encoding schemes *('Bus Invert'*, *'Working-Zone Encoding'* and

'Transition Pattern Coding'). Finally, Section 6 gives conclusions.

2 Related work

Early work on reducing bus power concentrated on reducing the bit transitions (Stan/Burleson [1]). The basic idea is to transfer an inverted word through the bus whenever it reduces the *Hamming Distance* between two successive bus transactions. An additional bus line is used to indicate whether the word is inverted or not. Later in [3], they generalize the *Bus-Invert Encoding* to a space-time redundant encoding. In this class of encoding schemes, low transition codes in an expanded code space are used to represent data words so as to reduce transition activities. In [13], another extension of a bus-invert scheme has been proposed by Kretzschmar et al. applying an adaptive local bus-invert to system buses. Henkel/Lekatsas [6] have proposed an encoding scheme for address buses by first changing the order of bus lines and then applying local bus-invert. Macchiarulo et al. [15] have shown that the layout of an address bus can be arranged for low power consumption.

Ramprasad et al. ([1]) introduced a general communication model to describe low power bus encoding systems. Their bus-encoding scheme consists of a source encoding part and a channel encoding part. The exploitation of source properties has been studied in [4] by Musoll et al. who proposed *Working-Zone Encoding* (WZE) to exploit locality of memory references. The basic observation is that applications favor a few '*working zones'* of their address spaces. If both sender and receiver keep a table of base addresses of these working zones, an address can be expressed as an offset along with an index of the working zone based address. Later, in [5], the WZE scheme is extended to multiplexed data buses. Panda/Dutt propose an approach to reduce address bus transitions [7].

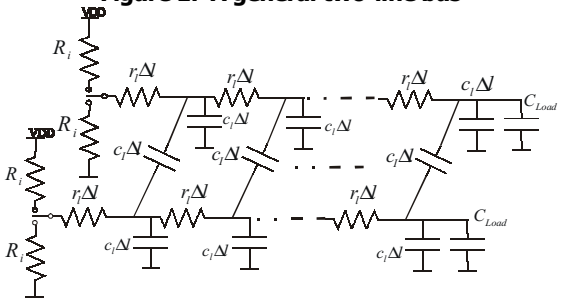
Approaches described above focus on reducing transition activities and/or focus on address buses only. Meanwhile, with emerging of deep sub-micron technologies (0.10u and beyond), inter-wire capacitances are no longer negligible. Sotiriadis/Chandrakasan showed in [8] that solely minimizing the number of the transitions may not lead to an optimal power reduction.

In our work, we take both inter-wire capacitance and source pattern properties into consideration (though our method is completely independent of the application that is running) and it outperforms all previous work on encoding techniques for general buses.

3 Bus Model

The bus power model we are using is based on the one proposed in [8] with some refinements as we consider the load capacitances of the en/decoder. Our model is shown in Figure 1, where R_i is the internal resistances of a bus

Figure 1: A general two-line bus



driver, r_l is the linear resistance of the bus lines, c_l is the linear capacitance to the ground and c_I is the linear inter-wire capacitance. We define a function $Eni(V_i^I, V_j^I, V_i^f, V_j^f)$ to describe the energy consumption of one transition related to inter-wire effects and another function $Ens(V^I, V^f)$ to describe the energy consumption related to a transition. Here superscript *I* stands for the voltage at the beginning of a bus transaction and *f* stands for the voltage at the end of a bus transaction. Label *i* and *j* mean the two different bus lines. Then, we can formulate the total energy consumption of a bus in the k-th transaction as,

$$En(k) = C_L V_{dd}^2 \cdot \{\rho \sum_{i=0}^{N-2} Eni[V_i(k-1), V_{i+1}(k-1), V_i(k), V_{i+1}(k)] + \sum_{i=0}^{N-1} Ens[V_i(k-1), V_i(k)]\}$$

where C_L is the total capacitance of a bus line to ground ,

$$\rho = \frac{\text{inter - wire capacitance}}{C_L},$$

$V_i(j)$ is the voltage swing on i-th bus line during the j-th transaction and N is the bit-width of the bus. We use this model later on calculate the energy consumption on the data bus.

4 Encoding and Decoding Schemes

Dictionary encoding techniques are widely used in data compression[12] (e.g. in UNIX' *compress*, *GIF* image compression etc). Dictionary techniques target at data source properties like recurring patterns that are kept in the so-called *dictionary* when identified. Hence, a small number of symbols can represent a large pattern. However, if a pattern is not in the dictionary, then more symbols than the original are needed to avoid ambiguity. Dictionary techniques are classified into two classes—*static* and *adaptive*. Adaptive techniques are used when there is not sufficient knowledge of the patterns and when the patterns tend to morph temporally. We use an adaptive technique and we will show why in the following.

4.1 Source Properties on Data Buses

Prior to developing an en/decoding scheme, we conducted extensive experiments to study the properties of

data sent via the data bus for a wide variety of applications from diverse domains. Some previous work has focused on exploiting the properties on address buses: T0 coding as in [9], WZE coding as in [4] and ACCS·LSIS as in [6].

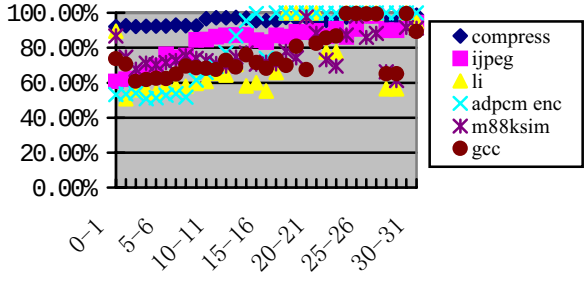
Our aim is to study whether there are properties on general data buses that can be exploited without having any knowledge of the application that is actually running.

Therefore, we analyzed the data streams between CPU and the data cache in a *SimpleScalar* simulator for a set of application programs including "*compress*", "*go*", "*gcc*", "*ijpeg*", "*li*", "*m88ksim*" and "*perl*" from SPEC95 and "*adpcm*" encoder/decoder from *MediaBench*. We used transition signaling [3], which means that a logical '1' represents a transition and a '0' represents the lack thereof. The correlations are obtained as follows: let $l_i(t)$ and $l_i(t-1)$ be the logical value of data bus line l_i at time t and t-1, respectively. The exclusive OR of these two variables will give the transition code for line i at time t. Then, the correlation between two lines i and j is as follows:

$$Correlation = \frac{\sum_{i=1}^{N} \overline{l_i(t) \oplus l_i(t-1) \oplus l_j(t) \oplus l_j(t-1)}}{N}$$

Note that a transition will add to the correlation when the inverse exclusive OR of the transition codes of the two lines i and j is equal to one, as depicted by the above equation.

Figure 2: Percent of correlation of transition signaling code on adjacent lines



In Figure 2, the correlation between any two adjacent lines is shown as a percentage (for a 32-bit data bus). Through investigations, we found out that the reason for these partly high correlations is due to one or more of the followings effects:

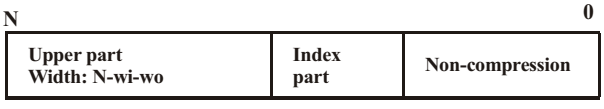
- The locality of calculation: in many cases, the result of an operation is close to one of the operands. For example, a loop variable is often increased or decreased by one.
- In modern processors (32-bit/64-bit), data bytes and half-words are often extended to words for easy of use. In addition, Boolean variables are usually stored in word format. In those cases, the upper part bit-lines of a word are either all '1's or '0's.
- During its running time, an application may use several sets of data simultaneously. While the lower parts of the values in each set may vary, the upper sub-words of the values in each data set tend to be the same.

Our analysis results suggest that the transition activities on adjacent lines have sufficiently high correlations that can be exploited by our ADES scheme

4.2 ADES: Adaptive Dictionary En/Decoding Schemes

The strong correlations of transitions on adjacent bus lines suggest that there are frequently recurring patterns in data streams. This leads to the definition of our dictionary encoding ADES. Since there is not sufficient knowledge on data characteristics to build static dictionaries, adaptive dictionaries are used in ADES. In order not to affect performance, the encoding and decoding circuits should be capable of executing within a clock cycle. Therefore, the bit-size of a pattern cannot exceed a word length. We have observed (Figure 2) that the lower parts of the data words on the data bus change quite frequently (i.e. lower correlation), so the encoding should not include the lower part

Figure 3: Word structure in ADES



ADES works as follows: An original data word is divided into three parts—*non-compressed part*, *index part* and *upper part* as shown in Figure 3. The encoder uses the index part of the current data to look-up the dictionary. Whenever there is a match (i.e. the upper part of the data word is in the dictionary), it then transmits the index part of the data word and the non-compressed part. Using the index part, the decoder is able to recover the upper part of the word from its dictionary. When a dictionary miss occurs, the encoder transmits the unchanged data word.

All entries in the dictionaries of the encoder and decoder are initiated to zeros. When a hit occurs, the content of the dictionaries remain unchanged. When a miss occurs, both entries corresponding to the current data word in the dictionaries are updated to the upper part of the current data word. The detailed encoding and decoding schemes are shown in Figure 4 and Figure 5, where a data word W is divided into three parts as shown in Figure 3:

- Non-compressed part: $W\langle 0 : wo-1 \rangle$
- Index part: $W\langle wo : wi+wo-1 \rangle$ and
- Upper part: $W\langle wi+wo : N-1 \rangle$.

'Wi' and *'wo'* are two integer parameters determining the width of the index part and the width of the non-compressed part, respectively. *'EncDict'* and *'DecDict'* are the two dictionaries maintained by the sender and the

receiver. The contents of the *'EncDict'* and *'DecDict'* are initiated as zero.

Figure 4: Encoding algorithm

Encoder :

($Data_k$ is the current word to be sent, Bus_k is the value sent on the bus.)

$Bus_k\langle 0:wi+wo-1\rangle = Data_k\langle 0:wi+wo-1\rangle;$

if $EncDict[Data_k\langle wo:wi+wo-1\rangle] = Data_k\langle wi+wo:N-1\rangle$ **then**

(match : do not change upper part of bus to save energy)

$Bus_k\langle N\rangle = 1;$ (signal a match)

else

(miss : send the original data)

$Bus_k\langle wi+wo:N-1\rangle = Data_k\langle wi+wo:N-1\rangle;$

$Bus_k\langle N\rangle = 0;$ (signal a miss)

(update dictionary)

$EncDict[Data_k\langle wo:wi+wo-1\rangle] = Data_k\langle wi+wo:N-1\rangle;$

Figure 5:Decoding algorithm

Decoder :

(Bus_k is the value received on bus, $Data_k$ is the output of the decoder.)

$Data_k\langle 0:wi+wo-1\rangle = Bus_k\langle 0:wi+wo-1\rangle;$

if $Bus_k\langle N\rangle = 1$ **then**

(match : recover data from dictionary)

$Data_k\langle wi+wo:N-1\rangle = DecDict[Bus_k\langle wo:wi+wo-1\rangle];$

else

(miss : use bus data)

$Data_k\langle wi+wo:N-1\rangle = Bus_k\langle wi+wo:N-1\rangle;$

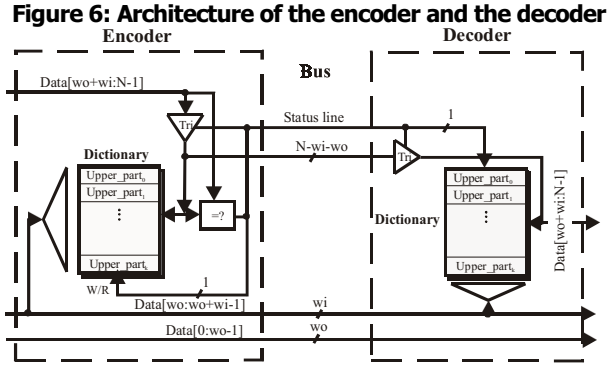
(update dictionary)

$DecDict[Bus_k\langle wo:wi+wo-1\rangle] = Bus_k\langle wi+wo:N-1\rangle;$

4.3 ADES Architecture

The block diagram of encoder and decoder are shown in Figure 6. The encoder is composed of one dictionary, one comparator and one tri-state gate. When a word is to be sent, the comparator in the encoder compares the output of the upper part of the data word and the output of the dictionary that uses the index part of the data word as an address. If the two values are identical, a match is signaled. The signal controls the tri-state buffer not to send any data in the upper part in order to reduce bit toggling. When the output of the dictionary is not equal to the upper part of the current word, a miss is indicated. The miss signal lets the tri-state gate pass the upper part of the data word and controls the dictionary update the active entry (the entry corresponding to the index part of the data word) to the upper part of the current data word. The status line of the bus is connected to the output of the comparator, so it contains the information whether there is a miss or match. When a word enters the decoder, the index part of the bus is fed into the dictionary in the decoder as an address. In case of a match, the signal on the status line shuts the tri-sate gate, the output of dictionary together with the index part and the non-compressed part of the bus form the decoder output. In case of a miss, the

Figure 6: Architecture of the encoder and the decoder



status signal lets the tri-state gate pass the upper part of the bus and writes the upper part of the bus into the dictionary. At the same time, the upper part of the bus together with the index part and non-compressed part of the bus form the decoder output.

A discussion of the cost of the encoder and decoder follows:

- Area:

The area cost of the encoder and decoder is almost solely determined by the cost of the two dictionaries. It can be estimated as

$$Area = 2 \cdot Reg\_File\_Size \cdot Line\_width \cdot Mem\_cell\_size\ .$$

Using *wo=6*, *wi=4* as an example, the register file size is 16 and the line width is 22. Assume we use a six-transistor SRAM cell. The total amount of circuit is about 4,500 transistors or about 750 gates. The area overhead for the additional line is small since we need only one.

- Energy/Power:

The largest part of the circuitry is SRAM, for each bus transaction, only a small percentage of the transistors show transition activities. The overall power consumption is less than 5% of the energy consumption of the data bus.

- Latency:

The two dictionary look-up processes are the most time consuming parts of the whole encoding and decoding process. The 16-register file can be organized as an SRAM with 8 rows and 2 columns. In order to compare with Intel Pentium III at 1.2GHz, we estimate the time overhead of the dictionary using a 0.10 μm technology. While a cycle of 1.2GHz Pentium III is 0.83 ns, the time overhead of the dictionary is about 0.4ns, less than half of a cycle. Still optimizations may further reduce the latency overhead. In addition, the table look-up activities of the encoder and the decoder can be overlapped. Thus, ADES encoding and decoding can be accomplished within one cycle @ 1.2GHz.

4.4 Bus Compaction Alternative

In addition to its capability of reducing bus energy consumption, the ADES schemes could also be used to reduce the size of a bus. The basic idea is that since we do not need all 32 lines during a hit, we use less bus lines.

When a miss occurs, more than one cycle is needed to send the whole word. The encoder and decoder do not require an additional cycle during a hit. Thus, the performance penalty is:

$$penalty = \frac{\#mem\ access\ inst}{\#total\ inst} \cdot missrate \cdot (miss\_penalty)$$

where *'miss_penalty'* is the number of additional cycles for a miss. To keep the *'miss_penalty'* at one cycle, 16 lines are needed for transferring a 32-bit word in two cycles. One additional line is needed for indicating 'miss/hit'. Thus, a total of 17 lines are required.

Therefore, for a 32-bit line data bus, we could deploy only 17 lines. If a hit occurs, the left 16 lines are used to transfer the index part and the offset part. Otherwise, two cycles are needed to transfer the 32-bit value through the left 16 lines. In this scheme, the compaction ratio of the bus would be 1:1.88. In the following experiments, we do not deploy bus compaction since our focus is solely on energy/power minimization.

5 Results and Comparison to Related Work

To evaluate our ADES scheme, we have conducted experiments on real-world and freely accessible applications using a *SimpleScalar* simulator. Different parameters (wi, wo, etc) are used and our scheme is directly compared (i.e. using the same set of applications) to several other works. The data buses are 32 bits wide throughout. We use the bus power-estimation model described in Section 3. In our experiments, the following

Table 1: Bus energy consumption of application programs (wo=3, wi=6)

App-lication	*com-press*	*gcc*	*ijpeg*	*li*	*m88k-sim*	*perl*
Raw	8.7e-6J	1.1e-2J	2.5e-3J	1.2e-2J	2.3e-5J	2.3e-2J
ADES	6.6e-6J	7.1e-3J	2.1e-3J	7.3e-3J	1.5e-5J	1.4e-2J
Energy Saved	25%	38%	18%	38%	34%	40%

application benchmarks are evaluated: "*compress*", "*go*", "*gcc*", "*ijpeg*", "*li*", "*m88ksim*" and "*perl*" in SPEC95 and "*adpcm*" en/decoder from the *MediaBench*. Parameters used for the calculation are:

$$0.10\ micron,\ c_x = 0.0583fF/\mu m\ [14]$$

$$l = 2cm,\ C_L = \frac{c_x l}{\rho} = 0.292pF,\ V_{dd} = 1.8V.$$

In Table 1, the ADES scheme is compared to the original (i.e. non-encoded) bus. By using ADES solely, up to a 40% of energy saving can be achieved (28% average). The effects of different *'wo'*, *'wi'* parameter sets are shown in Figure 7 (shows combined results for all applications). The best result for a parameter set achieved is 28% when *wo=3* and *wi=6*. However, a smaller *wi* is preferable since the encoding circuitry is smaller. Table 2

Figure 7: Percent of energy saved using ADES as a function of *wi* and *wo*

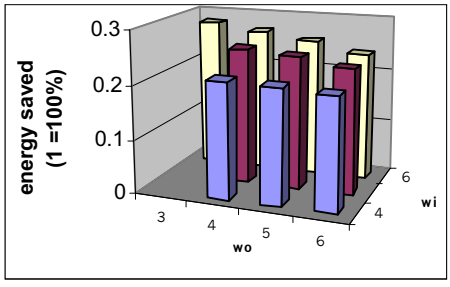


Table 2: Comparison among BI, WZE, TPC and ADES

Scheme	Avg. Energy per Mem Access	Avg. Energy Saved	Num. of Additional Lines	Number of Gates (approximately)	Delay
Raw	1.94e-11J	0%	N/A	N/A	N/A
BI4[2],[5]	1.90e-11J	2.5%	4	100	Low
WZE[4]	1.60e-11J	17.8%	4	1800[i]	High[4]
TPC[8][ii]	3.28e-11J	-68.9%	12	N/A	Low
ADES with BI[iii]	1.38e-11J	**28.9%**	**2**[iv]	750	Low

(comparison with others as an average over all applications), Table 3 (comparison with others in terms of energy for all applications separately) and Figure 7 (percentage comparison) show the results compared to related work: bus invert with 4 groups (*BI4*)[2],[5], *Working-Zone Encoding* (WZE) [4] and *Transition Pattern Coding* (TPC) [8], where a positive sign means a decrease of energy consumption (i.e. an improvement) and a negative sign means an increase of energy consumption compared to a non-encoded ('Raw') data bus. The average energy per memory access is calculated as

$$Avg.\ Energy\ per\ Mem\ acc\ for\ an\ app. = \frac{\#\ Total\ energy}{\#total\ memory\ access}$$

$$Avg.\ Energy\ per\ Mem\ Acc = \frac{\sum Avg.\ Energy\ per\ Mem\ Acc\ for\ an\ App.}{\#\ Application}$$

The comparison shows that ADES with bus invert is outperforming all the other encoding schemes at a relatively small circuitry and a low delay. It also shows that bus invert can contribute 8% energy savings (comparing table 2 with the column of wo=6, wi=4 in Figure 8), if it is combined with ADES and the gain is greater than using bus invert alone. This is because bus invert encoding with four groups has four additional lines. While the scheme reduces the energy on the original bus

[i] We assume that a subtractor is not shared for reason of reducing delay.

[ii] We add one zero line to the original word and divide it to 11 groups with 3 lines per group. One additional line is added to each group for encoding (m=3,a=1).

[iii] The parameters used in ADES is wo=6 and wi=4.

[iv] One additional line is for ADES and the other is used bus invert.

lines, it also brings in extra energy consumption on these additional lines. When bus invert encoding is combined

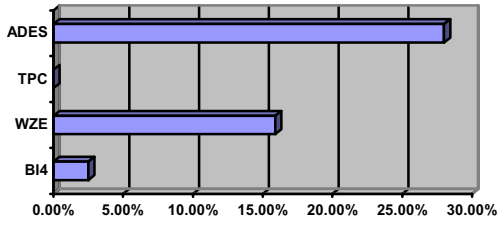
Table 3: Energy consumption for BI, WZE, TPC and ADES

	# mem_acc	Energy (Joule)				
		Orig	B4I	WZE	TPC	ADES w. BI
adpcm-dec	5.22e+5	1.16e-5	1.21e-5	9.75e-6	1.80e-5	8.49e-6
adpcm-enc	5.22e+5	1.09e-5	1.18e-5	9.13e-6	1.79e-5	8.22e-6
compress	2.22e+6	8.75e-6	8.34e-6	6.72e-6	2.07e-4	6.16e-6
gcc	5.13e+8	1.14e-2	1.13e-2	9.48e-3	1.38e-2	7.78e-3
go	4.86e+7	8.47e-4	7.80e-4	6.85e-4	8.31e-4	6.27e-4
ijpeg1	1.41e+8	2.51e-3	2.35e-3	2.22e-3	3.30e-3	1.99e-3
ijpeg2	2.78e+6	5.56e-5	5.47e-5	4.80e-5	6.80e-5	4.00e-5
li	4.56e+8	1.17e-2	1.07e-2	9.26e-3	1.23e-2	7.83e-3
m88ksim	1.07e+6	2.31e-5	2.17e-5	1.92e-5	2.18e-5	1.53e-5

with ADES only one additional line is added since the transitions are concentrated in the lower part of the bus. Thus, the extra energy is thus reduced to a minimum.

As for the Transition Pattern Coding (TPC) results, it seems TPC delivers good results only when the data is totally random. However, for the set of real-world applications we used it could not deliver improvements over the 'Raw' bus.

Figure 8: Average energy saved



6 Conclusions

We have presented the general bus (i.e. data bus) encoding technique 'ADES' that minimizes the energy/power consumption by taking into consideration inter-wire capacitances. Thereby, we do not exploit any specific characteristics of specialized buses (like address buses). We do though exploit the fact that there is a certain amount of correlation between adjacent bits of a word sent via the data bus when using real-world and freely available applications. By comparing our results to the non-encoded cases, we achieve 18% to 40% energy consumption (28% average). We also compared our 'ADES' scheme directly to other known data bus en/decoding techniques using the same set of applications and we achieve better results in all cases. Due to some additional en/decoder logic, the latency slightly increases but no additional clock cycles are needed. The limitation of our approach is that it might not be applicable to very area-sensitive SOCs since the en/decoders require (though relatively small, as shown) additional chip area.

References

[1] S. Ramprasad, N. R. Shanbhag, Ibrahim N. Hajj, "A Coding Framework for Low-Power Address and Data Busses", IEEE Tr. on VLSI Systems, Vol. 7, No. 2, pp. 212-221, Jun 1999.

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

[3] M. R. Stan, W. P. Burleson, "Low-Power Encodings for Global Communication in CMOS VLSI", IEEE Tr. on VLSI Systems, Vol. 5, No. 4, pp. 444-455, Dec 1997.

[4] E. Musoll, T. Lang, J. Cortadella, "Working-Zone Encoding for Reducing the Energy in Microprocessor Address Buses", IEEE Tr. on VLSI Systems, Vol. 6, No. 4, pp. 568-572, Dec 1998.

[5] T. Lang, E. Musoll, and J. Cortadella, "Extension of the Working-Zone-Encoding Method to Reduce the Energy on the Microprocessor Data Bus", International Conference on Computer Design, Oct 1998.

[6] J. Henkel, H. Lekatsas, **"**A2BC: adaptive address bus coding for low power deep sub-micron designs", Proc. of Design Automation Conference, pp. 744-749, 2001.

[7] P.R. Panda, N.D. Dutt, "Low-Power Memory Mapping Through Reducing Address Bus Activity", IEEE Tr. On VLSI Systems, Vol 7, No. 3, pp. 309-320, Sep 1999.

[8] P.P. Sotiriadis, A. Chandrakasan, "Bus energy minimization by transition pattern coding (TPC) in deep sub-micron technologies", IEEE/ACM International Conference on Computer Aided Design, pp. 322–327, 2000.

[9] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, 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.

[10] K. Kim, K. Baek, N. Shanbhag, C.L. Liu, S. Kang, "Coupling-driven signal encoding scheme for low-power interface design", IEEE/ACM International Conference on Computer Aided Design, pp. 318-321, 2000.

[11] Semiconductor Industry Association, " International technology roadmap for semiconductors", 1999.

[12] K. Sayood, "Introduction to Data Compression", Morgan Kaufmann Publishers, 1996.

[13] C. Kretzschmar, R. Siegmund, D. Müller, "Adaptive Bus Encoding Technique for Switching Activity Reduced Data Transfer over Wide System Buses", International Workshop - Power and Timing Modeling, Optimization and Simulation, Goettingen (D), Sep 2000.

[14] Z. Huang and M.D. Ercegovac, "Effect of Wire Delay on the Design of Prefix Adders in Deep-Submicron Technology", Proc. 34th Asilomar Conference on Signals, Systems and Computers, 2000.

[15] L. Macchiarulo, E. Macii, M. Poncino, "Low-Energy Encoding for Deep-Submicron Address Buses", IEEE/ACM Proc. of International Symposium on Low Power Electronics and Design (ISLPED'01), pp.176-181, 2001.