Power Efficient Arithmetic Operand Encoding

Eduardo Costa, Sergio Bampi
UFRGS
P. Alegre, Brazil
ecosta,bampi@inf.ufrgs.br

José Monteiro
IST/INESC
Lisboa, Portugal
jcm@algos.inesc.pt

Abstract

This paper addresses the use of alternative codes for arithmetic operators. The objective is twofold. First, to investigate operand codes that yield simpler, *i.e.*, power efficient, arithmetic modules. Second, to investigate signal encodings that lead to the reduction of the switching activity in the data buses. Although signal correlation is more relevant for address buses, where signal encoding has received much attention, in many cases correlation in the data buses is still very significant. By using low-switching operand codes directly in the arithmetic modules, the process of encoding and decoding of the signals can be avoided.

We propose a Hybrid encoding for the operators, which is a compromise between the minimal input dependency presented by the Binary encoding and the low switching characteristic of the Gray encoding. We present a methodology for the generation of arithmetic operators, such as adders and multipliers, using Hybrid encoded operands. The overall area, delay and power consumption under different word size operators are evaluated for both the Hybrid and Binary modules. The results show that power savings of up to 30% in array multiplier modules are possible, with 33% reduction in the switched capacitance in the buses. Additionally, a 17% delay improvement is achieved, with an area penalty of 30%. The Hybrid encoding can also be as easily used in address buses where the same 33% savings can be obtained with low overhead transcoders.

1 Introduction

For most of the current IC designs in key high volume applications, meeting the targeted power dissipation budget is of paramount importance. Research in CMOS power modeling and optimization is thriving for more accurate predictions and power efficiency. Recent research [1], [2] has focused on the CMOS power consumption estimation. There is a need for precise estimation to guide the designers towards minimizing and meeting strict power budgets.

This paper focuses on the power optimization of arithmetic modules through the use of different encodings for the operands. The main goal of the coding schemes is to reduce the overall switched capacitance in the system. In fact, in several applications, switching activity on high-capacitance buses accounts for a large fraction of the total power consumption. Recent research work have dealt with the encoding of the signals on the address buses in order to lower their switching activity [3], [4], [5], [6], [7], [8]. A significant reduction in the switched capacitance is reported, with power savings between 33% and 75% using different coding schemes.

In this work, we explore signal encoding to obtain power efficient arithmetic modules that operate directly on these codes. The aim is not only to reduce the switching activity in the input and output buses, but also the minimization of the complexity of the combinational logic in the modules. Additionally, there is no power overhead due to code converters since these are not required.

In general, there is less signal correlation in data buses than in address buses. The methodology we propose optimizes power inside the arithmetic modules, but also leverages on the data correlation that in many cases can be significant to reduce the switched capacitance in data buses.

We propose an *Hybrid* operand encoding that is a compromise between the minimal input logic dependency presented by the Binary encoding and the low switching characteristic of the Gray encoding. We present arithmetic modules that use operands under this encoding and compare area, delay and power with the Binary counterparts. We show that up to 30% power savings and 17% delay reduction are possible for an array multiplier, with about 30% area penalty. Additionally, this encoding achieves 33% switched capacitance reduction in the data buses.

Another feature of the Hybrid encoding is that its conversion to and from Binary is very simple, making it applicable to address buses with low overhead transcoders.

This paper is organized as follows. Section 2 discusses the main aspects related to coding for low power. In Section 3, we present operand encoding schemes used in order to implement arithmetic operators and we introduce the Hybrid code definition. Basic operators built using the Hybrid encoding are shown in Section 4. Section 5 presents Binary and Hybrid array multiplier architectures. Performance comparisons between the Binary and Hybrid architectures for different word sizes are presented in Section 6. Finally, in Section 7 we discuss the main conclusions of this work.

2 Coding for Low Power

Coding has long been used in communication systems to control the statistics of the transmitted data symbols, or in other words, to control the spectrum of the transmitted signal [3].

Low-power techniques for global communication in CMOS VLSI using data encoding methods are overviewed in [4], where it is shown that such techniques can decrease the power consumed for transmitting information over heavy load communication paths (buses) by reducing the switching activity.

One technique that has been proposed in order to reduce the switching activity on buses is One-Hot Coding [3]. This technique is a redundant coding scheme with a one-to-one mapping between the n-bit data words to be sent and the m-bit data words that are transmitted. The main disadvantage of this technique is related to the wire quantity required, proportional to 2^n.

The Limited-Weight Codes is another technique proposed in order to obtain switching activity reduction on buses [5]. This technique requires transition signaling in order to reduce the switching activity, since with transition signaling only 1's generate transitions. According to [5], transition signaling is convenient for low-power as it offers a direct way of controlling the bus activity factor simply by reducing the number of logical 1's transmitted over the bus.

The Bus-Invert method as a means of encoding words for reducing I/O power, in which a word may be inverted and then transmitted if doing so reduces the number of transitions [6]. In this method an extra bus line, called *invert* is used. The method looks at two consecutive data words on the bus. If the Hamming distance between the next word and the current transmitted word is at most $k/2$, the next word is sent as it is, with *invert* set to 0. Otherwise, each bit of the next word is complemented and invert is set to 1, indicating that the word has been complemented. The Bus-Invert method is explored in [8] in order to sequence words under the Bus-Invert scheme for the minimum transitions, *i.e.*, words can be complemented, reordered and then transmitted.

The Transition Coding and Bit Prediction techniques were used in order to reducing the number of transitions observed in data and address buses [7]. The Transition Coding technique indicates that there is a transition on the bus every time the data to be transmitted is a 1, and there is no transition on the bus if the data to be transmitted is a 0. Bit Prediction technique is used in some buses that exhibit very regular bit patterns. The main idea consists by using the Bit Prediction technique in address buses that exhibit a very high percentage of addresses that are sequential, so that a factor can be used to predict the value of the next data word with reasonably high accuracy.

One of the most promising encodings that can be used to reduce switching activity is the Gray code since only one bit changes between consecutive values. Therefore, for highly correlated signals the switching activity can be reduced significantly. This code has been applied to code address lines for both instruction access and data access to reduce the number of transitions [3]. In [9] it is observed that the use of Gray code to cache memory addresses can produce about 33% power reduction in the instruction cache. Although there is no overhead in terms of the number of bit lines, translating between Gray and Binary requires complex, *i.e.*, power consuming, encoders/decoders.

As presented above, there is a large number of techniques that resort to signal encoding in order to reduce switching activity on buses. These techniques have all been applied to address buses where data is highly sequential. In our work we apply similar techniques to arithmetic operators that operate directly upon different coded inputs. Although data buses in general do not present the same degree of sequentiality as address buses, there are many cases where data signal correlation is significant and can be explored to reduce power consumption. Further, we do not rely solely on the savings of the switched capacitance on the buses. Signal encoding is used to optimize the arithmetic modules themselves.

Table 1. HYBRID CODE REPRESENTATION ($m = 2$)

Dec	Hyb	Dec	Hyb	Dec	Hyb	Dec	Hyb
0	0000	4	0100	8	1100	12	1000
1	0001	5	0101	9	1101	13	1001
2	0011	6	0111	10	1111	14	1011
3	0010	7	0110	11	1110	15	1010

3 Operand Encoding

In the process of implementing arithmetic operators that operate directly upon coded inputs some different coding schemes were investigated. For example, the Transition Coding technique transforms the code representation from the bus before the arithmetic operation is performed. It was investigated the viability to get arithmetic operators using the code directly from the bus. However, it was observed that the output data depends on previous and actual states and that this information at the output is not enough. Taking into account these values for each operand requires complex hardware in order to perform the arithmetic operation.

As discussed in the previous section, the Gray code is very appealing for low power due to its intrinsic low switching. Therefore our first attempt was use the Gray code for the operands, therefore avoiding the costly encoders/decoders. The problem that we faced immediately was that all the bits in the output of an arithmetic operation are a function of all the bits of all the inputs. This in stark contrast with the Binary code, where the least significant bits only depend on the least significant bits of the operands. Consequently, the combinational logic required by arithmetic modules using Gray encoded operands is much larger than Binary modules. Furthermore, it is not possible to develop a regular structure for the operators such that different word sizes can be accommodated.

We arrived at a compromise between the Binary and the Gray codes, which we call the Hybrid code, that we propose next. A similar but more limited code has been developed independently in [10].

3.1 Hybrid Code Definition

The idea of the Hybrid code is to split the operands in groups of m-bits, encode each group using the Gray code and use the Binary approach to propagate the carry between the groups. In this manner, the number of transitions inside each group is minimized and a regular structure can easily be built, where the least significant group of the result will also depend only on the least significant group of the operators. Table 1 exemplifies the Hybrid encoding for 4-bit numbers and groups of $m = 2$ bits.

Table 2 presents the number of transitions for a counting sequence for the Binary, Gray and Hybrid codes. The Hybrid code presents a number of transitions right between the Gray and Binary codes, with 33% less number of transitions than the Binary code. Hence, for systems where the switched capacitance in the data buses is significant and where the data presents a high degree of correlation, up to a third of power can be saved.

Although in this work we target data buses, the Hybrid code that we propose in this section can naturally be applied to address buses, where this one third factor in power savings can be more easily achieved due to the higher degree of correlation.

Table 2. Number of Transitions for Different Codes

Number of Bits	Number of Transitions			Diff	
	Binary	Gray	Hybrid($m = 2$)	H→B	H→G
4 bits (0→15→0)	30	16	20	-33%	+25%
8 bits (0→255→0)	510	256	340	-33%	+33%
16 bits (0→65535→0)	131070	65536	87380	-33%	+33%

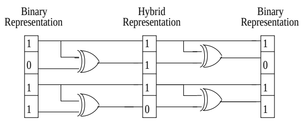

Figure 1. Conversion between the Binary and Hybrid codes.

An additional feature of the Hybrid code is that the conversion to and from Binary is very simple, as indicated in Figure 1. Translating words of W bits between Binary and Hybrid, in any direction, all that is required are $W/2$ EXOR gates. Therefore, the overhead in terms of transcoders is low, making this encoding very desirable even for the case of address buses.

4 Basic Arithmetic Elements

Using the Hybrid code it is possible to generate adders and multipliers with regular structures. In this section we present the elementary building blocks for these operators. We focus on the Hybrid code using groups of size $m = 2$, therefore we need 2-bit adders and multipliers. We are currently investigating different values for m.

The basic elements shown in this section were implemented in the BLIF format, mapped to the `mcnc` library, and the results presented were obtained with the SIS tool [11]. To obtain the power consumption values for the basic arithmetic elements, two different input signals were used. One is the *uniform* that represents the power consumption under an uniform input distribution. Another is the *vectors* that represent a sequence between all possible combinations between 0 and the maximum value for that number of bits.

4.1 Hybrid Adder Basic Element

We have found that adders operating directly with Hybrid encoded operands are more complex than Binary adders. Moreover, given the ease of conversion between Hybrid and Binary codes, the most efficient implementation for a Hybrid adder is in fact to use a Binary adder and make the conversion at the inputs. This is shown in Figure 2 for a 2-bit adder operator. Naturally, this implies that the Hybrid adders will consume more power than the Binary adders. However, depending on the data correlation and capacitance load of the data buses, an overall power reduction is still possible. In our work the structure showed in Figure 2 is used in the Hybrid array multipliers.

Table 3 shows area and power results for 2-bit Binary and Hybrid adders. As can be observed in Table 3, the Hybrid adder presents a little more area and power values when compared to the Binary circuit. This fact occurs due to EXOR gates necessary in Hybrid adder operator. According to [12] due to EXOR gates, the difference between Binary and Hybrid adder is propagated to higher structures. However, as will be shown, the Hybrid array multiplier can be more power efficient than the Binary circuit even using the basic element.

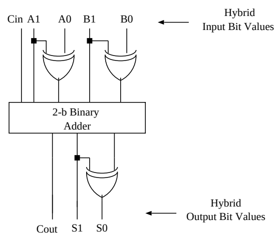

Figure 2. 2-bit Hybrid Adder.

Table 3. Area and Power for 2-bit Binary and Hybrid adders.

2-bit Adders	Area		Power (μW)	
	Nodes	Literals	Uniform	Vectors
Binary	10	28	83.7	78.4
Hybrid	13	40	121.2	113.9
Diff(Hyb→Bin)	+30%	+43%	+45%	+45%

4.2 Hybrid Multiplier Basic Element

For the Hybrid multiplier, it is possible to generate a simple 2-bit structure using just 8 logic gates, as depicted in Figure 3. This simple structure presents an advantage in terms of area and power compared to the Binary structure as shown by the results in Table 4.

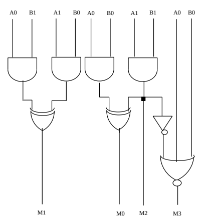

Figure 3. 2-bit Hybrid multiplier circuit.

The 2-bit Hybrid multiplier shows a better structure than shown by the Binary circuit. Contrary to the basic adder element, we were able to design a simpler basic Hybrid multiplier. Thus, no transformation is necessary in order to change the code representation. The Hybrid multiplier structure shown in Figure 3 is used in order to produce partial product terms in the Hybrid array multiplier presented in the next section.

5 Hybrid Array Multiplier

In an array multiplier, the operation is performed by adding partial products in sequence. Partial products are generated by intermediary multiplier cells and the carry is propagated to each group. The basic cell in the multiplier is a single-bit multiply ac-

Table 4. Area and Power for 2-bit Binary and Hybrid Multipliers.

2-bit Multipliers	Area		Power (μW)	
	Nodes	Literals	Uniform	Vectors
Binary	15	36	64.1	56
Hybrid	8	20	55	36.7
Diff(Hyb→Bin)	-47%	-44%	-14%	-34%

cumulate cell [13]. In this work, Binary and Hybrid array multipliers are implemented by considering multiplication of unsigned numbers. Binary architectures are implemented using the well-known array multiplier structures [13], [14] and Hybrid architectures are performed 2 bits at a time, in a base 4 representation. This representation allows us to produce a regular architecture as shown in Figure 7.

5.1 Binary Array Multiplier

In the operation of a Binary array multiplier, partial products are performed in a parallel form. Hand multiplication of unsigned or positive Binary numbers is done in much the same way as the familiar way in which ordinary decimal numbers are multiplied [13]. Figure 4 shows an example for a 4-bit Binary array multiplier.

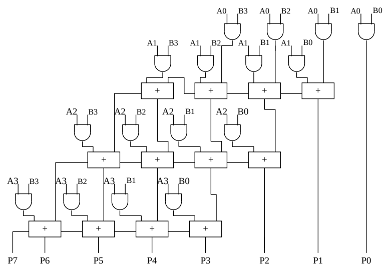

Figure 4. 4-bit Binary Array Multiplier.

As can be seen in the architecture of Figure 4, individual $1 \times W$-bit products are obtained with AND gates and are added together to obtain the complete $W \times W$ result. The W least-significant bits of the product are produced at the right hand side of the array.

```
    Binary Multiplier Procedure (W)
1.  /* First line */
2.  for j=0 to W-1 {
3.      write AND(in(A[0],B[j]); out(P[0,j]));
4.  }
5.  /* Remaining lines */
6.  for i=1 to W-1{
7.      for j=0 to W-1 {
8.          write AND(in(A[i],B[j]); out(R[i,j]));
9.          write adder(in(P[i-1,j+1],R[i,j],$c_{in}$[i,j-1]); out(P[i,j],$c_{out}$[i,j]));
10.     }
11.     P[i,j+1]=$c_{out}$[i,j];
12. }
```

Figure 5. Algorithm for the generation of different word-length Binary multipliers.

The pseudo-code presented in Figure 5 was implemented in order to generate any word size Binary array multiplier. The result is a circuit in BLIF format that allows us to estimate the area, delay and power for each word-length in the SIS environment. These values are then compared to the corresponding Hybrid architecture.

The Binary array multipliers are obtained by expressing partial product terms (AND gates) and adder operators in BLIF format. The amount of operators needed in each architecture are calculated according to the number of bits in the word, represented by the W variable, which defines the number of columns and lines of elements necessary to compose the complete multiplier.

5.2 Hybrid Multiplier Operation

In the Hybrid array multiplier, the operation is performed for a group of m bits at a time, where m is the size of the group for the Hybrid code. We will concentrate on the case for $m = 2$. Partial product terms are expressed by a base 4 representation, as shown in Figure 6, for an example of a 4-bit Hybrid array multiplying operation.

As shown in Figure 6, the partial product terms are performed by multiplying each 2-bit groups of the multiplier and multiplicand terms. Each partial line is performed by a $2 \times W$ multiplying operations as shown in Figure 6(b), where W represents the word size of the multiplicand term.

The final line for the Hybrid code multiplying operation is obtained by adding each 2-bit groups for the partial product terms, as shown in Figure 6(a). Decimal values are obtained by converting each group of 2 bits assuming a base 4 representation.

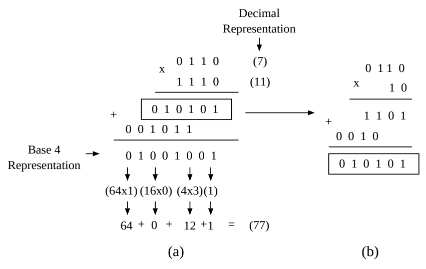

Figure 6. 4-bit Hybrid Multiplying Operation.

5.3 Hybrid Multiplier Architecture

In a similar manner as was discussed for the Binary array multiplier, the Hybrid array multiplying operation can be performed in a regular form, as in Figure 6. However, contrary to the Binary multiplier where the operation is performed bit by bit, the Hybrid operation is performed for a 2-bit group at a time. In order to implement an architecture that can consider these aspects, the basic elements shown in Figures 2 and 3 are used. The full Hybrid architecture can be seen in Figure 7 for a 4-bit Hybrid array multiplier example, where boxes $*$ and $+$ are the Figure 3 and Figure 2 circuits respectively.

As shown in Figure 7, the Hybrid array multiplier architecture presents a regular structure. Basic adders and multipliers elements are used in order to perform partial products. The 2-least significant bits of the product are produced at the right hand side of the array. The other bits are produced by adding partial product terms. As can be observed for the 4-bit Hybrid architecture example shown in Figure 7, it is necessary just one line com-

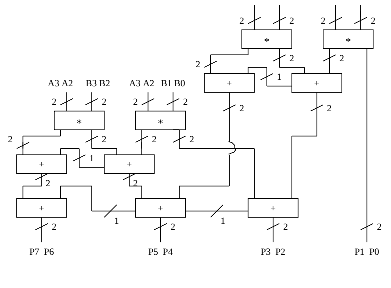

Figure 7. 4-bit Hybrid Array Multiplier Architecture.

posed by adders responsible for adding partial product terms. We show next that for a W-bit Hybrid architecture, ($W/2$)-1 of these adder lines will be used. Note that the regularity of the circuit's structure is the same as for the conventional array multiplier, the only difference is that the building block is larger.

5.4 Algorithm for Hybrid Multipliers

As mentioned previously, the Hybrid code allows to produce regular operations and therefore Hybrid array multiplier architectures can be obtained. In order to implement these regular architectures, a C program was written so that any word size Hybrid array multipliers can be implemented in the same form as was done for Binary multipliers. Figure 8 shows the algorithm for the generation of the Hybrid structure.

```
    Hybrid Multiplier Procedure (W)
1.  for i=0 to W-1, i=i+2{
2.      for j=0 to W-1, j=j+2{
3.          write multip(in(A[i+1]A[i],B[j+1]B[j]);
                    out(P[i/2,2j+3]P[i/2,2j+2]P[i/2,2j+1]P[i/2,2j]));
4.      }
5.      for j=0 to W-1, j=j+2{
6.          write adder(in(P[i/2,2j+5]P[i/2,2j+4],P[i/2,2j+3]P[i/2,2j],$c_{in}$[j/2]);
                    out(R[i/2,j+3]R[i/2,j+2],$c_{out}$[(j/2)+1]));
7.      }
8.  }
9.  for i=2 to W, i=i+2{
10.     R[i/2,1]R[i/2,0]=P[i/2,1]P[i/2,0];
11.     for j=0 to W+1, j=j+2{
12.         write adder(in(R[(i/2)-1,j+3]R[(i/2)-1,j+2],R[i/2,j+1]R[i/2,j],$c_{in}$[j/2]);
                    out(P[i/2,j+1+i]P[i/2,j+i],$c_{out}$[(j/2)+1]));
13.         R[i/2,j+1]R[i/2,j]=P[i/2,j+1+i]P[i/2,j+i];
14.     }
15. }
```

Figure 8. Algorithm for the generation of different word-length Hybrid multipliers.

The pseudo-code presented in Figure 8 shows that the Hybrid structures are obtained by expressing partial product terms and adders in BLIF format. Once again the number of columns and lines composed by basic elements that define the full architecture is calculated according to number of bits represented by the *W* variable.

Although the algorithm in Figure 8 has been written so that Hybrid architectures can perform operation in each 2-bit groups, it should be observed that it is easily adapted in order to perform operation in each m-bit group. In this case, we need basic Hybrid adders and multipliers in base 2^m representation.

6 Performance Comparisons

The algorithms shown in Figures 5 and 8 allow for the implementation of any word size Binary and Hybrid array multipliers. In this section, we present area, delay and power results for 4, 8, 12, 16 and 32 Binary and Hybrid multipliers architectures.

6.1 Area Results

As observed before, Binary and Hybrid multiplier architectures present partial product terms expressed in a different form. While the Binary architectures use $W \times W$ AND gates, the Hybrid architectures use $W/2 \times W/2$ basic multiplier elements (Figure 3), where W represents the word length. Although the Hybrid architecture uses half of the basic product elements than those used by the Binary architecture, it should be observed that each basic Hybrid multiplier element is composed by 8 logic gates and therefore the Hybrid multipliers present more gates for performing partial product than the Binary multipliers. Moreover, as was shown in Table 3, the basic 2-bit Hybrid adder presents higher area than the basic Binary element. Thus, as shown in Table V, the Hybrid array multipliers present higher area than Binary array multipliers for all word sizes.

Table 5. AREA FOR BINARY AND HYBRID ARRAY MULTIPLIERS.

Number of Bits	Area	Binary	Hybrid	Diff Hyb→Bin
4b	Nodes	64	82	+28.1%
	Literals	168	232	+38.1%
8b	Nodes	320	406	+26.8%
	Literals	848	1144	+34.9%
12b	Nodes	768	970	+26.3%
	Literals	2040	2728	+33.7%
16b	Nodes	1408	1744	+23.8%
	Literals	3744	4984	+33.1%
32b	Nodes	5888	7390	+25.5%
	Literals	15680	20728	+32.1%

6.2 Delay Results

Although Hybrid multipliers present higher area, these architectures can present less delay values than Binary multipliers, as shown in Table VI. This due to the fact that Hybrid multipliers present a smaller critical path delay than Binary multipliers. As can be seen in Figures 4 and 7, while the Hybrid architecture presents ($W/2$)-1 lines of adders responsible for adding partial product terms, the Binary architecture presents $W - 1$ lines. This fact become expressive in terms of delay values presented by different word size architectures.

Table 6. DELAY FOR BINARY AND HYBRID ARRAY MULTIPLIERS.

Architectures	Delay (ns)		Difference
	Binary	Hybrid	Hyb→Bin
4b	46.4	47.1	+1.51%
8b	117.6	104.7	-11%
12b	188.8	162.3	-14.03%
16b	260	219.9	-15.42%
32b	544.8	450.3	-17.34%

Table VI shows that, for a 4-bit multiplier, the Binary circuit presents less delay than the Hybrid circuit. However, in more complex circuits, from 8-bit to 32-bit architectures, where the critical path became expressive, the Hybrid structures present increasingly less delay value. The results shown in Table VI were obtained in SIS using the general delay model obtained from the `mcnc` library.

6.3 Power Consumption Results

Despite the higher area presented by Hybrid multipliers, these architectures consume less power than the Binary architectures, as shown in Table 7 (the power values were obtained in SLS tool [15] using general delay model and assume Vdd=5V and freq=20MHz). This occurs because the Hybrid multipliers cause internally a smaller number of transitions than the Binary multipliers. According to [16], [17], in a known signal flow most significant bits present a high temporal correlation and a small number of transitions. Thus, an input signal with high correlation applied to high word size Hybrid multipliers can reduce power consumption for these architectures. As shown in Table 7, we have applied to different word size Binary and Hybrid multipliers a *real trace* input signal. This signal represents the number of points of 2 sinusoidal signals with 90 degree phase difference. The number of points used in the real trace input signal was chosen between 2 and 10 times the maximum number representation 2^W. This interval reveals a good approximation where sinusoidal signals present less abrupt variation. However, as can be observed in Table 7 for the 32-b architecture, only part of the sinusoid was simulated due to the great amount of points required for the full simulation. In this architecture it was only possible to simulate in SIS tool using zero delay model.

Table 7. POWER FOR BINARY AND HYBRID ARRAY MULTIPLIERS.

Number of Bits	Number of Points	Binary P (mW)	Hybrid P (mW)	Diff. H→B
4b	100	2.58	3.27	+26.7%
	200	1.37	1.83	+25%
8b	1000	26.05	26.12	+0.27%
	2000	13.60	13.68	+0.61%
12b	20000	55.14	46.87	-15%
	40000	28.41	23.33	-22%
16b	150000	191.60	129.06	-32.64%
	300000	69.52	55.85	-24%
32b	4294964	14061.08	10905.45	-22.44%

Table 7 shows that for a given architecture the higher the number of points used in the real trace input signal, the less power consumption shown by the Binary and Hybrid architectures. This occurs because for a real trace input with high number of points the sinusoidal signal presents a better representation with less abrupt variation. In this case for each architecture shown in Table 7, the Hybrid circuit presents more power reduction than the Binary circuit. This occurs because in a signal with less abrupt variation, the Hybrid code presents less number of transitions than the Binary code (as observed in Table 2).

Another important aspect shown in Table 7 is related to the power consumption according to the word size. As can be observed, the higher the number of bits, the less power consumption presented by the Hybrid multipliers. This occurs because in more complex circuits the most significant bits become more important and when it is applied a real trace input with a less abrupt variation the Hybrid architectures take advantage of correlation aspects, where Hybrid code shows better results than Binary code. As can be seen in Table 7 from 12 bits to 32 bits architecture Hybrid multipliers present increasingly less power consumption than Binary circuits. As should be observed in Table 7 the best results shown by Hybrid multipliers are addressed to signals with high correlation aspects. We should note that while this is true for this input signal, this result may not hold for all types of input signals since we rely on input correlation.

7 Conclusions

The reduction of the number of transitions in order to minimize power consumption is the main goal of techniques that use different coding schemes on buses. In this work, we experimented implementing arithmetic operators that operate directly upon coded inputs.

We propose a Hybrid code as an alternative way to implement arithmetic operators, emphasizing on array multiplier structures. This code represents a compromise between the Binary and the Gray codes. The idea behind this encoding is, just like the Binary, to have the least significant bits of the output depend only on the least significant bits of the input operands, thus reducing logic complexity, but at the same time have consecutives codes follow the Gray code for groups of m bits. This code achieves 33% less switching than the Binary, with a similar level of complexity.

Performance comparisons from Binary and Hybrid array multipliers architectures were investigated and results showed that despite higher area shown by Hybrid multipliers, these architectures can present less delay and power consumption values.

As future work we hope to apply our Hybrid operators in concrete applications and explore different values for m, the size of the groups that work as Gray codes in the Hybrid encoding scheme. We also hope to experiment our Hybrid code in other multiplier circuits than array multipliers.

8 Acknowledgment

This research was supported in part by the CAPES (Brasilia/Brasil) Institute and by the portuguese FCT under program POCTI.

References

[1] F. Najm. A Survey of Power Estimation Techniques in VLSI Circuits. *IEEE Transactions on VLSI Systems*, 2(4):446–455, December 1994.

[2] C-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin. Power Estimation Methods for Sequential Logic Circuits. *IEEE Transactions on VLSI Systems*, 3(3):404–416, September 1995.

[3] A. Chandrakasan and R. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[4] M. Stan and W. Burleson. Low-Power Encodings for Global Communication in CMOS VLSI. *IEEE Trans. on VLSI Systems*, March 1997.

[5] M. Stan and W. Burleson. Limited-weight Codes for Low-Power I/O. *IEEE International Workshop on Low Power Design*, April 1997.

[6] M. Stan and W. Burleson. Bus-Invert Coding for Low-Power I/O. *IEEE Transactions on VLSI Systems*, March 1995.

[7] P. Ramos and A. Oliveira. Low Overhead Encodings for Reduced Activity in Data and Address Buses. *IEEE International Symposium on Signals, Circuits and Systems*, pages 21–24, July 1999.

[8] R. Murgai, M. Fujita, and M. Oliveira. Using Complementation and Resequencing to Minimize Transitions. In 35^{th} *DAC*, June 1998.

[9] C. Su and A. Despain. A Cache Design Tradeoffs for Power and Performance Optimization AQ Case Study. In *IEEE International Symposium on Low Power Design*, April 1995.

[10] R. Hakenes and Y. Manoli. A segmented gray code for low-power microcontroller address buses. In *EUROMICRO Conference*, volume 1, pages 240–243, 1999.

[11] E. Sentovich and et al. SIS: A System for Sequential Circuit Synthesis. Technical report, May 1992.

[12] E. da Costa, J. Monteiro, and S. Bampi. Power Optimization Using Coding Methods on Arithmetic Operators. *IEEE International Symposium on Signals Circuits and Systems (accept for publication)*, July 2001.

[13] R. Hartley and K. Parhi. *Digit-Serial Computation*. Kluwer Academic Publishers, 1995.

[14] K. Hwang. *Computer Arithmetic - Principles, Architecture and Design*. School of Electrical Engineering, 1979.

[15] A.J. Genderen. SLS: An Effi cient Switch-Level Timing Simulator Using Min-Max Voltage Waveforms. *Proceedings of VLSI Conference*, pages 79–88, 1989.

[16] P. Landman and J. Rabaey. Architectural Power Analysis: The Dual Bit Type Method. *IEEE Trans. on VLSI Systems*, 3(2):173–187, June 1995.

[17] S. Ramprasad, N. Shanbhag, and I. Hajj. Analytical Estimation of Signal Transition Activity from Word-Level Statistics. *IEEE Transactions on CAD*, 16(7):718–733, July 1997.