

# A Genetic Approach To Bus Encoding

Giuseppe Ascia Vincenzo Catania Maurizio Palesi Antonio Parlato  
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni  
University of Catania, Italy

## Abstract

*Increasingly rapid improvements in silicon process technologies have made it possible to integrate tens of millions of transistors operating at frequencies in the GHz range on a single silicon die. This has led to a great increase in the amount of power consumed per area unit in a silicon die, which makes power consumption optimisation a fundamental design objective, along with area and performance. In this paper we present two new approaches based on genetic algorithms (GAs) to reduce power consumption by address buses in an embedded system. The first approach (GEG) makes it possible to obtain the truth table of an encoder that minimizes switching activity on a bus, whereas the second (GNEG) outputs the netlist of the encoder using the lowest possible number of logic gates. The approaches are compared with the most effective ones already presented in literature, on both multiplexed and separate buses. The results obtained demonstrate the validity of the approaches, which on average save up to 50% of the transitions normally required, as well as their practical applicability, even in an on-chip environment.*

## 1. Introduction

Up to a few years ago VLSI design was mainly oriented towards optimisation in terms of area and performance. Levels of integration and operating frequencies were not so high as to cause power consumption problems. Nowadays, however, rapid improvements in silicon process technologies have made it possible to integrate tens of millions of transistors operating at frequencies in the GHz range on a single silicon die. This has led to a great increase in the amount of power consumed per area unit in a silicon die, which makes power consumption optimisation a fundamental design objective, along with area and performance. The increase in power dissipation has compounded the problems of packaging and ensuring reliable operations by these chips (every 10°C increase in power doubles the failure rate). Thus, for many applications, it is essential to minimize the amount of power dissipated in order to reduce the costs associated with cooling the chip. In addition, the increase in levels of integration has led to the concept of system-on-a-chip (SoC) and the proliferation of portable battery-driven applications such as mobile phones, PDAs, digital cameras, laptops, etc. The competitiveness of these products in the marketplace depends to a great extent on the functionality-to-weight ratio. Most of these portable applications run on batteries,

which are also the heaviest component of these systems. Without power minimization, these systems would need heavy batteries to ensure reasonable time of operation between battery recharges.

One of the main contributors to power consumption is switching activities on the high-capacity lines of an interconnection system. It is estimated that power dissipated on the I/O pads of an IC ranges from 10% to 80% of the total power dissipation with a typical value of 50% for circuits optimized for low power. Whereas up to a few years ago these problems were only due to switching activity on off-chip buses, today they are of increasing importance for on-chip buses as well. The wire-to-gate capacitance ratio has, in fact, gone from 3 for old technologies to 100 for the new ones and is continuing to rise [5], shifting the problem of power consumption from computing to communications. Various techniques have been proposed to reduce switching activity on the lines of a bus and thus the amount of power dissipated. The technique consists of encoding data prior to transmission and decoding it once it reaches its destination. Unfortunately, many of them introduce extra lines along the bus or are so complex that the great overhead in terms of power for the encoder/decoder means that they can only be used in an off-chip environment.

In this paper we propose two bus encoding techniques based on genetic algorithms (GA). Both can be applied to embedded systems, i.e. ones in which it is possible to know in advance the trace of the patterns transmitted on a communication bus following execution of a specific application. The first technique generates a truth table for an encoder which minimizes switching activity on a bus. The second, on the other hand, operates directly on the scheme of the encoder, modifying the connections and type of logic gates used so as to obtain a structure that will minimize the number of outgoing transmissions and at the same time the number of logic gates used.

The results obtained on a set of benchmarks confirm the validity of the approaches: the saving in terms of transitions is greater than that obtained by the most efficient techniques so far proposed in the literature.

## 2. Formulation of the Problem

Let us consider a binary alphabet to compose words with a fixed length of  $w$  bits. Let  $U^{(w)}$  be the universe of discourse for words of  $w$  bits (i.e. the set of words it is possible to form with  $w$  bits). The cardinality of  $U^{(w)}$  is there-

fore  $2^w$ .

An *encoder* associates each word in  $U^{(w)}$  with one and only one word in  $U^{(w)}$  in such a way that there is only one output coding for each input, thus making the *decoder* able to decode the word univocally. In formal terms, an encoder  $\mathcal{E}$  is an injective and surjective (and therefore invertible) function  $\mathcal{E} : U^{(w)} \rightarrow U^{(w)}$ , i.e. such that the following condition is met:

$$\forall \alpha, \beta \in U^{(w)}, \alpha \neq \beta \Rightarrow \mathcal{E}(\alpha) \neq \mathcal{E}(\beta) \quad (1)$$

We will call the inverse of  $\mathcal{E}$  *decoder* and indicate it with  $\mathcal{E}^{-1}$ .

As no redundancy is being considered, it is easy to calculate that  $2^w!$  different encoders are possible. Once the reference stream has been fixed, the ensuing number of transitions on the bus depends on the encoder used. The aim is therefore to find the best encoder that will minimize the number of transitions on a bus for a specific reference stream. Of course, as the space of possible encoders grows in size with the factorial of the size of the bus, exploration based on an exhaustive technique would be unfeasible.

### 3. Our Proposal

In this section we will present two approaches for generating an encoder that will minimize switching activity on a communication bus. Both are static, in the sense that the encoder is generated ad hoc on an address stream taken as input.

In Section 2. it was pointed out that designing an encoder that will minimize switching activity on a bus can be seen as a problem of optimization and dealt with using design space exploration techniques. The design space, which includes all the encoders that could possibly be realized, grows in a factorial fashion along with the number of words to be encoded, which in turn grows exponentially with the size of the bus.

In general, when the space of configurations is too large to be explored exhaustively, one solution is to use evolutionary techniques. Genetic algorithms have been used in several VLSI design fields [4]: in problems relating to layout such as partitioning, placement and routing; in design problems including power estimation, technology mapping and netlist partitioning and in reliable chip testing through efficient test vector generation. All these problems are untreatable in the sense that no polynomial time algorithm can guarantee an optimal solution and they actually belong to the NP-complete and NP-hard categories.

#### 3.1. GEG: Genetic Encoder Generator

Figure 1 shows the design flow called *GEG* (Genetic Encoder Generator).

The starting point is the specific application being executed (e.g. simulated), to obtain a memory reference trace file which will be the address stream used to generate the encoder. In order to facilitate generation of the encoder,

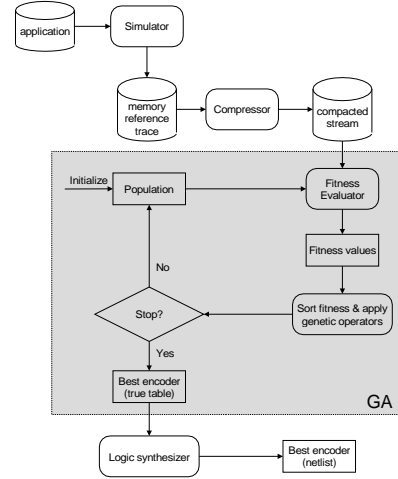


Figure 1: Encoder design flow.

the stream is compressed, as will be explained later. Initially a population of encoders is initialized with random encoders and evaluated on the compressed stream. Each encoder in the population has an associated fitness value which represents a measure of its capacity to reduce the number of transitions on the bus. Encoders with higher fitness values are therefore those which determine a lower number of transitions on the bus when stimulated with the compressed stream. The classical genetic operators, suitably redefined for this specific context, are applied to the population and the cycle is repeated until a stop criterion is met. At the end of the process, the individual with the highest fitness value is extracted from the population. This individual will be the optimal encoder being sought. As will be seen later on, the encoder is expressed in the form of a truth table. The last step in the flow is therefore logical synthesis of the optimal encoder, which can be done using any automatic logical synthesis tool. To obtain the encoder it will, of course, be sufficient to exchange the encoder input and output columns and perform the synthesis.

The memory reference trace file produced following execution of an application typically comprises hundreds of millions of references. It is therefore advisable to compress the stream so as to obtain a stream with an upper bound on the number of patterns. If  $S$  is the initial stream and  $S^*$  is the compressed one, the optimal encoder obtained using  $S^*$  has to be the same as the one that would have been obtained if we had used  $S$  as the input to the encoder design flow. The compression is therefore lossless for encoder generation purposes. Rather than compression, the technique used is based on a different representation of the reference stream. Let us consider a bus with a width of  $w$ . A reference stream is a sequence of patterns. Each pattern is an address of  $w$  bits. A compressed stream is also a sequence of patterns. A generic pattern is a 3-tuple  $\langle r_i, r_j, n_{ij} \rangle$  with  $i, j = 0, 1, 2, \dots, 2^w - 1$  and  $i > j$  that specifies the number of occurrences  $n_{ij}$  when the references  $r_i$  and  $r_j$  are consecutive in  $S$ . The meaning of the condition  $i > j$  can be explained by observing that, for

	Encode	Decode
0	5	4
1	1	1
2	3	7
3	6	2
4	0	6
5	7	0
6	4	3
7	2	5

**Figure 2: Representation of the chromosome.**

our purposes, it is only necessary to know what the consecutive addresses are and not their order. If inverted, in fact, the number of transitions does not change. Using this transformation, the maximum number of patterns in  $S^*$  will be  $2^{w-1} \times (2^w - 1)$  whatever the number of patterns in  $S$ . For example, if  $S = [12, 3, 12, 12, 5, 7, 12, 5, 7, 5]$  the compressed stream will be made up of the following four patterns  $S^* = [\langle 3, 12, 2 \rangle, \langle 5, 7, 3 \rangle, \langle 5, 12, 2 \rangle, \langle 7, 12, 1 \rangle]$

### 3.1.1. GA-based Bus Encoding

The approach we propose uses genetic algorithms as the optimization tool. Application of GAs to an optimization problem requires definition of the following three attributes: the chromosome, the fitness function, the genetic operators.

The *chromosome* is a representation of the format of the solution to the problem being investigated. In our case it is a representation of an encoder. The representation we chose consists of encoding the truth table of an encoder. In this way the chromosome will be made up of as many genes as there are rows in the truth table of an encoder. The gene in position  $i$  represents encoding of the word  $i$ . That is, for an encoder of  $w$  bits, we will have  $2^w$  genes. The  $i$ -th gene will represent encoding of the binary word that encodes  $i$  with  $w$  bits.

For reasons that will be explained when we deal with the definition of the genetic operators, the chromosome was enriched with further information. The chromosome can be represented as a table with  $2^w$  rows and 2 columns. Each row corresponds to a gene. Once the generic row  $i$  is fixed, the first column represents the encoding of  $i$ , while the second gives the position of the gene whose encoding is  $i$  (Figure 2).

The *fitness function* measures the fitness of an individual member of the population. In our case the individual is represented by an encoder, so the fitness function assigns each encoder a numerical value that measures its capacity to reduce switching activity on a bus. Naturally, the fitness function will depend not only on the encoder but also on the reference stream the encoder is stimulated by.

Let  $T_{woe}(S^*)$  be the number of transitions on the bus due to  $S^*$  without encoding and  $T_{we}(E, S^*)$  the number of transitions on the bus due to  $S^*$  when it is filtered by

the encoder  $E$ . The fitness function is defined as follows:

$$f(E, S^*) = \frac{T_{woe}(S^*) - T_{we}(E, S^*)}{T_{woe}(S^*)} \quad (2)$$

that is,  $f(E, S^*)$  returns the number of transitions saved when the encoder  $E$  is used for the compressed stream  $S^*$ .

The fitness function defined by Equation (2) is applied to each individual in the population (i.e. to each encoder). The aim of the GA is thus to make the population evolve so as to obtain individuals with increasingly higher fitness values.

The encoders are ordered by decreasing fitness values. The individual with the highest fitness value will always be inserted into the new population (elitism). The encoders making up the population are selected with a probability directly proportional to their fitness value. With a user-defined probability the genetic operators are applied to them and they are inserted into the new population. This selection process is repeated until the new population reaches the size established by the user.

The genetic operators were appropriately redefined so as to guarantee that application to an encoder (in the case of mutation) or a pair of encoders (in the case of cross-over) always gives rise to an encoder.

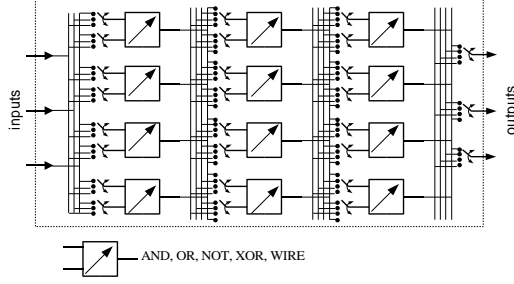
**Mutation** Mutation is a unary operator that is applied with a certain probability (which we will call *mutation probability*) to an encoder. Application of the mutation operator to an encoder consists of varying the coding of a word with a probability equal to the mutation probability.

```
Mutate(EncDec encoder, double prob)
begin
  for i=0 to encoder.rows do
    if (Event(prob))
      new_enc = randomInt(0, encoder.rows-1);
      Update(encoder, i, new_encoding);
    end if
  end for
end
```

where the function `Event(p)` returns `true` with a probability of  $p$ , and the function `randomInt(m, M)` returns a random integer ranging between  $m$  and  $M$ . The function `Update(...)` updates the coding of a word while maintaining consistency at the end of the decoding phase.

**Cross-over** Cross-over is a binary operator that is applied to two elements of the population with a certain probability that we will call *cross-over probability*. Given two encoders  $E_1$  and  $E_2$ , and having chosen two random indexes  $i$  and  $j$  where  $i < j$ , the coding of the words  $i, i+1, \dots, j-1, j$  is exchanged between  $E_1$  and  $E_2$ .

```
xOver(double prob, EncDec E1, EncDec E2)
begin
  if (Event(prob))
    int i = randomInt(0, E1.rows-1);
    int j = randomInt(0, E1.rows-1);
```



**Figure 3: A gate in a two-dimensional template, gets its second input from either one of two gates in the previous column.**

```

for r=i to j do
  int aux = E1[r].enc;
  Update(E1, r, E2[r].enc);
  Update(E2, r, aux);
end for
end if
end

```

### 3.2. GNEG: Genetic Netlist Encoder Generator

The second technique proposed uses GAs to directly obtain the scheme of the encoder. The first problem to solve is encoding (i.e. the representation) of the solutions as chromosomal strings that the GA can evolve. The representation we chose is a 2D matrix in which each element is a gate (there are 5 types of gates: AND, NOT, OR, XOR and WIRE) that receives its two inputs from any gate in the previous columns, as shown in Figure 3. A chromosomal string encodes the matrix shown in Figure 3 by using triplets in which the first two elements refer to each of the inputs used, and the third is the corresponding gate.

The aim is to generate logic circuits that are encoders (i.e. that meet the condition (1)), which will minimize the number of output transitions for a fixed amount of input traffic, and maximize the number of WIRES (or, equivalently, use the lowest possible number of logic gates). Let  $C$  be a logic circuit with an input of  $n$  bits and an output of  $n$  bits. Let us indicate as  $C(x) = y$  the output  $y$  of  $C$  when an input  $x$  is present. Let us also indicate as  $O(C)$  the number of pairs of input words that generate the same output:

$$O(C) = |\{(x_1, x_2) : x_1 \neq x_2 \wedge C(x_1) = C(x_2)\}|$$

Of course, if  $C$  is an encoder,  $O(C) = 0$ . We define the fitness function as follows:

$$f(C) = \begin{cases} 2^n - O(C) & \text{if } O(C) \neq 0 \\ 2^n + s(C) + w(C) & \text{otherwise} \end{cases}$$

where  $s(C)$  is the fraction of transitions saved by using the encoder  $C$  as compared with the amount required when no coding is used, and  $w(C)$  is the number of wires in the circuit  $C$ . In other words, we can say that our fitness function works in two stages. In the first stage the search space is explored to find the encoder. In the second, the

fitness function is modified and each valid design is evaluated according to the number of gates it contains and the output transitions required: the fewer the gates and output transitions, the more positive the evaluation.

## 4. Experiments

In this section we will present the results obtained by applying our approaches, comparing them with the most effective approaches proposed in the literature.

The application scenario referred to is encoding of the addresses transmitted on a 32-bit bus and generated by a processor during execution of a specific application. Two cases are considered: (i) the bus is multiplexed, (ii) it is a dedicated bus. In the former case the addresses travelling on the bus refer to both fetching instructions and accesses generated by load/store instructions. In the latter case, the bus considered is dedicated address bus for fetching instructions (e.g. the address bus between a processor and an instruction cache).

The 32-bit bus is partitioned with clusters containing the same number of bits and the approach was applied to each cluster separately. It is, in fact, computationally unfeasible to apply the approach to the whole bus, given that the data structure used would require tables of  $2^{32}$  rows to be handled. The cases studied referred to clusters of 4 and 8 bits. No particular criterion was followed in grouping the lines into clusters — they were grouped sequentially in a cluster. With  $c$  clusters, for example, each will include  $w = 32/c$  lines. The  $i$ -th cluster will contain the lines  $i \times c, i \times c + 1, \dots, i \times c + w - 1$ . A different way of clustering the lines of the bus (e.g. allocating lines with a higher correlation to the same cluster) may well enhance the performance of the approach and will be investigated in subsequent analyses.

We considered the same reference traces as are used in [1], generated following the execution of specific applications in the field of image processing, automotive control, DSP etc.. More specifically, `dashb` implements a car dashboard controller, `dct` is a discrete cosine transform, `fft` is a fast Fourier transform and `mat_mul` a matrix multiplication.

In all the experiments that will be discussed in the following subsections, we considered a population of 10 individuals, a mutation probability of 50%, and a cross-over probability of 25% for *GEG*. For *GNEG* we considered a population of 100 individuals, a cross-over probability of 90%, a mutation probability of 1%, and a 3x5 gate matrix. These parameters were set following an exhaustive series of simulations.

### 4.1. Address Bus (fetch + load/store)

In this subsection we will comment on the results obtained when encoding is applied to a multiplexed address bus (i.e. one on which addresses generated by both fetch and load/store instructions are travelling).

bench	trans	GEG8		GEG4		GNEG4		Beach		Others	
		trans	saving	trans	saving	trans	saving	trans	saving	trans	saving
dashb	619680	317622	48.74%	435516	29.71%	420353	32.17%	443115	28.40%	486200	21.50%
dct	48916	28651	41.40%	33179	32.17%	35464	27.50%	31472	35.60%	39327	19.60%
fft	138526	67468	51.30%	85405	38.30%	78525	43.31%	85653	38.10%	100127	27.70%
mat_mul	105950	50552	52.30%	60446	42.90%	62360	41.14%	60654	42.70%	77384	26.90%
Average saving		48.44%		35.77%		39.69%		36.20%		23.93%	

Table 1: Transitions saving for the address multiplexed bus.

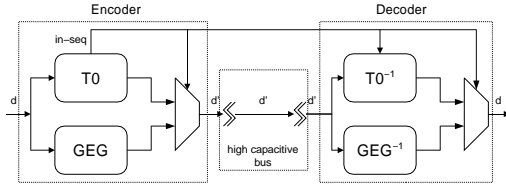


Figure 4: Block diagram of the *GEG+T0* encoder.

Table 1 summarizes the results obtained. The first column (*bench*) identifies the benchmark. The second (*trans*) gives the total number of transitions on the bus when no encoding scheme is applied. The remaining columns (in groups of two) give the number of transitions for each approach (*trans*) and the percent saving in transitions as compared with the case in which no encoding scheme is applied (*saving*). *GEG8* and *GEG4* represent the same implementation of the approach *GEG* applied to partitioned buses of 4 and 8 bits respectively. *GNEG4* represent the same implementation of the approach *GNEG* applied to partitioned buses of 4 bits. *Beach* is the approach proposed in [1]. *Others* indicates the best result obtained by the encoding schemes *Gray* [7], *T0* [2], *Bus-invert* [6], *T0+Bus-invert*, *DualT0* and *DualT0+Bus-invert* [3]. As can be seen, *GEG4* and *GNEG4* are on average equivalent to *Beach*. Increasing the size of the clusters to 8 bits increases the saving by about 13% as it is possible to exploit the temporal correlation between the references more fully.

#### 4.2. Address Bus (fetch only)

When the address bus is not multiplexed the percentage of addresses in sequence increases considerably. Table 2 gives the results obtained on an address bus carrying references to fetch operations alone.

In this case *T0* achieves much better savings than the other approaches by exploiting the high percentage of addresses in sequence which do not determine any transitions on the bus. *GEG8* maintains its efficiency with average savings of over 40%. The efficiency of *T0* can be further enhanced by using a hybrid approach *GEG8+T0*.

Figure 4 shows a scheme of how this can be achieved. The pattern to be transmitted is encoded with both *T0* and *GEG*. If it is in sequence with the previous one, the *T0* encoding is transmitted; otherwise the *GEG* encoding is transmitted. Even though *GEG+T0* is extremely efficient at reducing the amount of power dissipated on the bus, in

bench	Area ( $\mu\text{m}^2$ )		Delay (ns)		Power (mW)	
	Enc	Dec	Enc	Dec	Enc	Dec
<i>GEG</i>						
dashb	10806	10397	1.50	1.42	0.46	0.47
dct	10642	10352	1.67	1.38	0.45	0.44
fft	10753	10397	1.49	1.38	0.46	0.46
mat_mul	8916	8729	1.53	1.51	0.40	0.42
<i>GNEG</i>						
dashb	265	273	0.38	0.42	0.09	0.11
dct	255	275	0.39	0.43	0.09	0.11
fft	298	301	0.41	0.48	0.10	0.10
mat_mul	272	280	0.39	0.41	0.08	0.09

Table 3: Area, delay and power characteristics of the encoders and decoders.

calculating the saving account has to be taken of the overhead due to power consumption by the encoding/decoding logic. In *GEG+T0*, in fact, this contribution is certainly greater than that of both *T0* and *GEG*, as it contains them both, and both are active at the same time. Another point against *GEG+T0* is that it inherits from *T0* the use of a signaling line that is not present in *GEG*.

#### 4.3. Overall Power Analysis

Table 3 gives the area, delay and power characteristics of the encoders and decoders generated by *GEG* for 8-bit clusters and the benchmarks described previously. The results were obtained using Synopsys Design Compiler for the synthesis, and Synopsys Design Power for the power estimation. The circuits were mapped onto a  $0.13\mu\text{m}$ , 1.2V gate-library from Virtual Silicon. The clock was set to a conservative frequency of 100 MHz (i.e. a period of 10 ns) for *GEG* and 300MHz (i.e. a period of 3.3 ns) for *GNEG*. The average delay introduced by the encoder/decoder is, in fact, shorter than 4 ns for *GEG* and shorter than 1.3 ns for *GNEG* and so less than 40% of the clock cycle is dedicated to encoding and decoding information.

An encoding scheme is advantageous when the power saved on the bus (due to less activity) is greater than the power consumed by the encoding and decoding blocks. The power consumed by the bus can generally be expressed as  $P_B = \frac{1}{2} V_{dd}^2 \alpha f C_l$  where  $V_{dd}$  is the supply voltage,  $\alpha$  is the switching activity (i.e. the ratio between the total number of transitions on the bus and the number of patterns transmitted),  $f$  is the clock frequency and  $C_l$  is

bench	in-seq	trans	GEG8		GNEG4		Gray		T0		GEG8+T0	
			trans	saving	trans	saving	trans	saving	trans	saving	trans	saving
dashb	55.88%	111258	65694	40.96%	72528	34.81%	70588	36.55%	41731	62.49%	30182	72.87%
dct	60.31%	11675	6639	43.13%	7072	39.43%	6885	41.02%	2851	75.58%	1851	84.14%
fft	59.92%	25017	14486	42.09%	15743	37.07%	15969	36.16%	7021	71.93%	5063	79.76%
mat_mul	63.63%	26814	13802	46.08%	16212	39.54%	17095	36.24%	7850	70.72%	4345	83.79%
Average savings			43.06%		37.71%		37.49%		70.18%		80.14%	

**Table 2: Transitions saving for fetch only address bus.**

	$C_l$ (pF)				
	dashb	dct	fft	mat_mul	Average
GEG	1.61	2.74	1.70	1.78	1.93
GNEG	0.12	0.20	0.12	0.13	0.14

**Table 4: The minimum capacity a bus line has to have for the approach to be effective.**

the capacity of a bus line (assuming that all the lines have the same capacity). The overall percentage of power saved when an encoding scheme is used, as compared with when no encoding is used, can be calculated as follows:

$$P_{sav} = 100 \times \frac{P_{woe} - P_{we}}{P_{woe}}$$

where  $P_{woe}$  is the power consumed when no encoding strategy is used (which therefore corresponds to  $P_B$ ) and  $P_{we}$  is the power consumed when an encoding strategy is used (i.e. the sum of the power consumed by the encoder  $P_E$ , the decoder  $P_D$  and the bus  $P_B$ ). Solving the inequality  $P_{sav} > 0$  as a function of  $C_l$ , we find the minimum bus line capacity with which there is a positive net saving in power:

$$C_l > \frac{2(P_E + P_D)}{V_{dd}^2 f (\alpha_{woe} - \alpha_{we})} \quad (3)$$

Table 4 summarizes the minimum capacity a bus line has to have for the approach to be effective for each benchmark. It is not a large value even for an on-chip bus line. We can therefore conclude that the techniques proposed can effectively be used even with on-chip buses.

## 5. Conclusions

In this paper we have presented two GA-based strategies for designing an encoder that will minimize switching activity on a bus. The first, called *GEG* (Genetic Encoder Generator), draws up a truth table for the encoder that will minimize switching activity on the communication buses in an embedded system. The second, called *GNEG* (Genetic Netlist Encoder Generator), evolves a population of encoders with the dual aim of obtaining the least complex structure (i.e. comprising as few gates as possible) and minimizing switching activity. The results obtained on a set of specific applications for embedded systems have demonstrated the superiority of our approaches, with savings of around 50% on multiplexed address buses (instructions/data) and close to 45% on instruction address

buses. In the latter case the *T0* scheme [2] performs better than the approaches proposed here, with average savings of 70%. A mixed technique *GEG+T0* (in which a *GEG* and *T0* works concurrently) further enhances the efficiency of *T0*, achieving average savings of 80%. Finally, the low level of complexity of the encoder and decoder obtained make it possible to use them even in an on-chip environment.

## References

- [1] L. Benini, G. D. Micheli, E. Macii, M. Poncino, and S. Quer. Power optimization of core-based systems by address bus encoding. *IEEE Transactions on Very Large Scale Integration*, 6(4), Dec. 1998.
- [2] L. Benini, G. D. Micheli, E. Macii, D. Sciuto, and C. Silvano. Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems. In *Great Lakes Symposium VLSI*, pages 77–82, Urbana, IL, Mar. 1997.
- [3] L. Benini, G. D. Micheli, E. Macii, D. Sciuto, and C. Silvano. Address bus encoding techniques for system-level power optimization. In *IEEE Design Automation and Test Conference in Europe*, pages 861–866, Paris, France, Feb. 1998.
- [4] P. Mazumder and E. M. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall, Inc., 1999.
- [5] International technology roadmap for semiconductors. Semiconductor Industry Association, 1999.
- [6] M. R. Stan and W. P. Burleson. Bus invert coding for low power I/O. *IEEE Transactions on VLSI Systems*, 3:49–58, Mar. 1995.
- [7] C. Su, C. Tsui, and A. Despain. Saving power in the control path of embedded processors. *IEEE Design and Test of computers*, 11(4):24–30, 1994.