An Evolutionary Approach for Reducing the Energy in Address Buses

Giuseppe Ascia Vincenzo Catania Maurizio Palesi Antonio Parlato
Department of Computer Science and Telecommunications Engineering
University of Catania, Italy

Abstract

In this paper we present a genetic approach for the efficient generation of an encoder to minimize switching activity on the high-capacity lines of a communication bus. The approach is a static one in the sense that the encoder is realized ad hoc according to the traffic on the bus. This is not, however, a limiting hypothesis if the application scenario considered is that of embedded systems. An embedded system, in fact, executes the same application throughout its lifetime and so it is possible to have detailed knowledge of the trace of the patterns transmitted on a bus following execution of a specific application. The approach is compared with the most efficient encoding schemes proposed in the literature on both multiplexed and separate buses. The results obtained demonstrate the validity of the approach, which on average saves up to 50% of the transitions normally required.

1 Introduction

The spread of intelligent, portable electronic systems (digital cameras, cellular phones, PDAs, etc.) has made power consumption one of the main optimization targets in the design of embedded systems. A significant amount of the power dissipated in a system is known to be due to off-chip communication, for example between the microprocessor and the external memory. In the last few years the spread of systems integrated on a single chip, or systems-on-a-chip (SoC), and the use of increasingly advanced technology, has shifted the problem of power consumption on communication buses to the on-chip level as well — for example, in systems interconnecting the cores of a SoC. The ratio between wire and gate capacity has, in fact, gone from 3, in old technologies, to 100 in more recent ones [1] and is continuing to grow. For this reason, approaches aiming to reduce power consumption, which up to quite recently concentrated on the computing logic, are now tending to focus more and more on the communication system.

As the power dissipated by bus drivers is proportional to the product of the average number of transitions and the capacity of the lines of the bus, it may be a good idea to encode the information transiting on a communication bus so as to minimize the switching activity. Several approaches have been proposed and can be classified according to the type of bus to which the encoding schemes are applied (address/data/control bus), the class of architectures considered (general-purpose, special-purpose), the complexity of the encoder, which determines whether it is applicable on-chip and/or off-chip, etc..

The approach proposed in this paper refers to embedded applications, 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 trace is used to generate a truth table for an encoder that will minimize switching activity on the bus, which can thus be synthesized using any automatic logical synthesis tool. As the problem of searching for the encoder that will minimize switching activity can be viewed as an optimization problem, the exploration strategy adopted uses a heuristic method based on genetic algorithms (GAs).

A further contribution made by the paper is that of solving the problem of the considerable size of the trace files (used as input in static approaches) by means of a simple compression algorithm. The memory reference trace file is first compressed and then used to generate the encoder, thus drastically reducing processing times without affecting the efficiency of the approach.

The results obtained on a set of benchmarks confirm the validity of the approach: 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 therefore 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 $\mathcal{E} : U^{(w)} \longrightarrow U^{(w)}$, i.e. such that the following condition is met:

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

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. Figure 1, for example, shows three of the possible 24 encoders for 2-bit words. 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 optimal 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.

Encoder 1		Encoder 2		Encoder 3	
In	Out	In	Out	In	Out
00	00	00	11	00	10
01	01	01	10	01	11
10	10	10	00	10	01
11	11	11	01	11	00

Figure 1: Three of the possible 24 encoders for 2-bit words.

3 GA-based Bus Encoding

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 possible 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 [2]: 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.

The approach we propose uses genetic algorithms as the optimization tool. 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.

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. More precisely the fitness functions is the number of transitions saved when the encoder is used for the reference stream.

4 Experiments

In this section we will present the results obtained by applying our approach, 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 [3], 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.

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*		*Beach*		*Others*	
		trans	*saving*	*trans*	*saving*	*trans*	*saving*	*trans*	*saving*
`dashb`	619680	317622	48.74%	435516	29.71%	443115	28.40%	486200	21.50%
`dct`	48916	28651	41.40%	33179	32.17%	31472	35.60%	39327	19.60%
`fft`	138526	67468	51.30%	85405	38.30%	85653	38.10%	100127	27.70%
`mat_mul`	105950	50552	52.30%	60446	42.90%	60654	42.70%	77384	26.90%
Average saving		48.44%		35.77%		36.20%		23.93%	

Table 1: Transitions saving for the address multiplexed bus.

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 applied to partitioned buses of 4 and 8 bits respectively. *Beach* is the approach proposed in [3]. *Others* indicates the best result obtained by the encoding schemes *Gray* [4], *T0* [5], *Bus-invert* [6], *T0+Bus-invert*, *DualT0* and *DualT0+Bus-invert* [7]. As can be seen, *GEG4* is 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.

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

Table 2: Transitions saving for fetch only address bus.

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 *GEG+T0*.

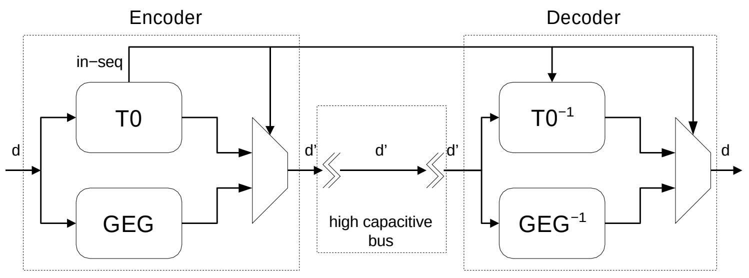

Figure 2: Block diagram of the *GEG+T0* encoder.

Figure 2 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 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 in inherits from *T0* the use of a signaling line that is not present in *GEG*.

5 Conclusions

In this paper we have presented a GA-based strategy for designing an encoder that will minimize switching activity on a bus. The strategy, called *Genetic Encoder Generator* (*GEG*) has been compared with the most effective techniques proposed in the literature. The results obtained on a set of specific applications for embedded systems have demonstrated the superiority of our approach, 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 [5] performs better than the approach 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%.

Of course, the analysis carried out in this paper does not consider the power overhead due to the encoder/decoder. The next step will therefore be logical synthesis of the encoder and decoder and power analysis. At any rate, the structure of the encoder/decoder system generated by *GEG* does not require power-hungry elements like correlators, decorrelators, registers etc., which are present in other approaches. It can reasonably be expected, therefore, that the power contribution made by the encoder/decoder will be sufficiently limited to allow the approach to be applied to on-chip buses as well.

References

[1] Neil H. West and Kamran Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, 1998.

[2] Pinaki Mazumder and Elizabeth M. Rudnick, *Genetic Algorithms for VLSI Design, Layout & Test Automation*, Prentice Hall, Inc., 1999.

[3] Luca Benini, Giovanni De Micheli, Enrico Macii, Massimo Poncino, and Stefano Quer, “Power optimization of core-based systems by address bus encoding,” *IEEE Transactions on Very Large Scale Integration*, vol. 6, no. 4, Dec. 1998.

[4] C. Su, C. Tsui, and A. Despain, “Saving power in the control path of embedded processors,” *IEEE Design and Test of computers*, vol. 11, no. 4, pp. 24–30, 1994.

[5] Luca Benini, Giovanni De Micheli, Enrico Macii, Donatella Sciuto, and Cristina Silvano, “Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems,” in *Great Lakes Symposium VLSI*, Urbana, IL, Mar. 1997, pp. 77–82.

[6] Mircea R. Stan and Wayne P. Burleson, “Bus invert coding for low power I/O,” *IEEE Transactions on VLSI Systems*, vol. 3, pp. 49–58, Mar. 1995.

[7] Luca Benini, Giovanni De Micheli, Enrico Macii, Donatella Sciuto, and Cristina Silvano, “Address bus encoding techniques for system-level power optimization,” in *IEEE Design Automation and Test Conference in Europe*, Paris, France, Feb. 1998, pp. 861–866.