A Genetic Bus Encoding Technique for Power Optimization of Embedded Systems

Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi

University of Catania
Department of Computer Science and Telecommunications Engineering
V.le Andrea Doria, 6 — 95125 Catania, Italy
{gascia,vcatania,mpalesi}@diit.unict.it

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. The approach refers to embedded systems in which 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 [9] and is continuing to grow, as foreseen in [6]. 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 approach have been proposed and can be grouped into two categories: *static* and *dynamic*.

Static techniques are based on a priori knowledge of the stream of patterns that will travel on the bus to generate an ad hoc encoder which will minimize the switching activity for that stream. The best-known static technique proposed in the literature is the *Beach solution* proposed by Benini *et. al* [1], who analyze the correlation between blocks of bits in consecutive patterns to find an encoding strategy that will reduce transition activity on a bus. In the approach proposed by Henkel *et. al* [4], as the most internal lines of a bus have a greater capacity, the reference trace is used to find a static permutation of bits that will reduce the activity on the lines with the greatest capacity.

Dynamic techniques don't require an advance knowledge of the stream of patterns: encoding decisions are made on the sole basis of past history. In the *bus-invert* technique [7] for data buses whose patterns are typically distributed in a random fashion a word to be transferred is inverted if the Hamming distance between it and the previous word is greater than the size of the bus divided by two. In this way the maximum number of lines that switch will be limited to half the size of the bus. Of course, an additional signaling line is needed to signal the type of encoding applied (inversion or no encoding).

Most of the bus encoding strategies proposed in the literature were envisaged for address buses. They all exploit the strong time correlation between one pattern and the next that is typically observed in an address stream. The high frequency of consecutive addresses, for instance, can be exploited by using *Gray* rather than natural binary encoding [8]. In this way the number of transitions for consecutive patterns will be limited to 1. If additional signaling lines are introduced, even better results can be obtained by using *T0* encoding [2] or one of the several variations on it [3]. The basic idea is to freeze the bus if the address to be sent is consecutive to the one sent previously (the receiver will generate the address locally).

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).

Furthermore a simple compression algorithm has been proposed to reduce the considerable size of the trace files (used as input in static approaches). Using the compressed memory reference trace file processing times have been drastically reduced with the same the efficiency.

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.

The rest of the paper is organized as follows. In Section 2 we present our GA-based proposal for the generation of an optimal encoder. In Section 3 we present the result obtained. Finally, in Section 4 we draw our conclusions.

2 Our GA-based proposal

In this section we will propose a static technique based on GAs for efficient generation of an encoder to minimize switching activity on a bus. The design flows is shown in Figure 1. Henceforward reference will be made to address buses, i.e. buses on which addresses travel (e.g. the addresses of instructions to be executed or general addresses generated by load/store instructions).

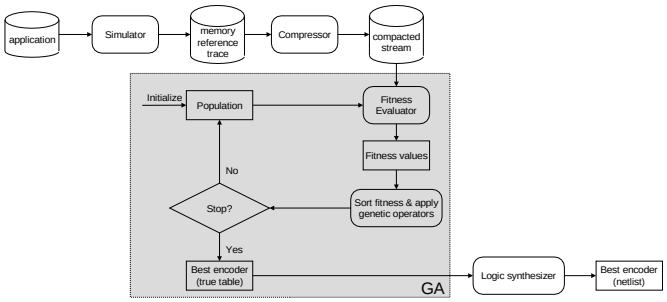

Fig. 1. Encoder design flow.

Starting from the specific application which is executed (simulated, emulated etc.), a memory reference trace file is obtained. In order to facilitate generation of the encoder, these trace file is compressed, as will be explained in the following subsections. The compressed stream is used to generate the optimal encoder which is expressed in the form of a truth table. As encoder generation can be viewed as an optimization problem, we used GAs to explore the design space. 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. 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 rest of this section is organized as follows. We will first classify encoder generation as a classical optimization problem that can be solved by using design space exploration techniques. We will then solve the problem of the size of the stream of references by means of a simple process of compression that will

exponentially increase the computational efficiency of the approach. Then we will deal with the problem of exploring the design space using GAs.

2.1 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. 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.

2.2 Compression of the Reference Stream

Evaluation of an encoder is a computationally onerous task. 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 $i, j = 0, 1, 2, \ldots, 2^w - 1$ and $i > j$ that specifies the number of occurrences n_{ij} when the references r_i ad r_j are consecutive in S. The meaning of the condition $i > j$ can be explained by observing that, for 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^* wil be:

$$L(w) = 1 + 2 + 3 + 4 + \ldots + (2^w - 1) = \frac{2^w \times (2^w - 1)}{2}$$

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 $S^* = [\langle 3, 12, 2\rangle, \langle 5, 7, 3\rangle, \langle 5, 12, 2\rangle, \langle 7, 12, 1\rangle]$

2.3 GA-based Bus Encoding

The design space of an encoder, 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 [5].

The approach we propose uses genetic algorithms as the optimization tool. Application of GAs to an optimization problem requires definition of the following four attributes: the chromosome, the fitness function, the genetic operators and the stop criterion. Each of these attributes will be defined with specific reference to the case study being investigated in the following subsections.

The Chromosome 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 chromosome is 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.

The Fitness Function The fitness function measures the fitness of an individual member of the population. In our case 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.

If E indicates the chromosome that maps the encoder and S^* the compressed reference stream, the number of transitions on the bus caused by S^* when no encoding scheme is applied (binary encoding) is:

$$T_{woe}(S^*) = \sum_{i=1}^{\text{rows}(S^*)} H(S^*[i,1], S^*[i,2]) \times S^*[i,3]$$

where H is the *Hamming distance* function and $S^*[i,j]$ is the j-th field of the i-th pattern in the stream S^*. Considering the i-th pattern, $S^*[i,1]$ and $S^*[i,2]$

are the pair of successive references and $S^*[i,3]$ is the number of occurrences of this pair. The number of transitions on the bus due to S^* when it is filtered by the encoder E is:

$$T_{we}(E,S^*) = \sum_{i=1}^{\text{rows}(S^*)} H(E[S^*[i,1],1], E[S^*[i,2],1]) \times S^*[i,3]$$

In short, the fitness function is defined as follows:

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

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

The Genetic Operators 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 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. Three genetic operators are used: Mutation, Permutation and Cross-over.

They were appropriately redefined so as to guarantee that application to an encoder (in the case of mutation and permutation) 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 E. Let E be an encoder of w bits. Application of the mutation operator to E consists of varying a single bit of an encoding chosen at random.

More specifically, a random number i between 0 and $2^w - 1$ is extracted and one of the w bits (also chosen at random) of the encoding of i is inverted.

Permutation Permutation is a unary operator applied with a certain probability (which we will call *permutation probability*) to an encoder E. Let E be an encoder of w bits. Application of the permutation operator to E consists of exchanging the encoding of two randomly chosen words.

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*. Let E_1 and E_2 be two encoders of w bits. When applied to the two encoders E_1 and E_2 the cross-over operator swaps the coding of a word chosen at random from

the 2^w that are possible. That is, it randomly extracts the word i, and $E_1[i,1]$ are swapped with $E_2[i,1]$. Of course, to guarantee that the condition (1) is met by the new encoders generated, $E_1[i,2]$ and $E_2[i,2]$ have to be updated.

The Stop Criterion Various stop criteria can be used, for example a user-managed parameter or an automatic criterion (e.g. convergence-based) that stops the process of evolution when no appreciable improvements in the species have been observed for a few generations. The stop criterion we will use is based on a user-defined maximum number of generations.

3 Experimental results

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.

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%, a permutation probability of 25%, and a cross-over probability of 25%.

The stop criterion used consists of blocking iterations when no appreciable improvements in the fitness value of the best encoder in the population have been observed for a certain number of generations.

3.1 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).

Table 1 summarizes the results obtained. The first column (*bench*) identifies the benchmark. The second (*trans.*) gives the total number of transitions on the

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.

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 [1]. *Others* indicates the best result obtained by the encoding schemes *Gray* [8], *T0* [2], *Bus-invert* [7], *T0+Bus-invert*, *DualT0* and *DualT0+Bus-invert* [3]. 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.

3.2 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 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

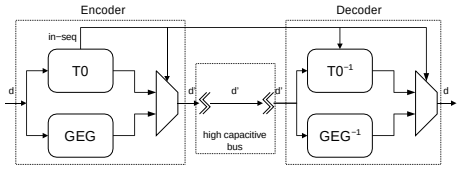

Fig. 2. Block diagram of the *GEG+T0* encoder.

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*.

3.3 Overal 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 DesignCompiler for the synthesis, and Synopsys DesignPower for the power estimation. The circuits were mapped onto a $0.18\mu m$, $1.8V$ gate-library from STMicroelectronics. The clock was set to a conservative frequency of 50 Mhz (i.e. a period of 20 ns). The average delay introduced by the encoder/decoder is, in fact, shorter than 4 ns and so less than 20% of the clock cycle is dedicated to encoding and decoding information.

bench	*Area* (μm^2)		*Delay (ns)*		*Power (mW)*	
	Encoder	*Decoder*	*Encoder*	*Decoder*	*Encoder*	*Decoder*
`dashb`	25456.64	25477.12	1.93	1.89	0.397	0.511
`dct`	25989.12	24985.60	1.90	1.97	0.282	0.398
`fft`	25325.57	25804.80	1.79	2.00	0.352	0.515
`mat_mul`	21733.38	21729.28	1.87	1.96	0.244	0.462
Average	24626.18	24499.20	1.87	1.96	0.319	0.472

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

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. Table 4 summarises the minimum capacity a bus line has to have for the approach to be effective for each benchmark.

bench	dashb	dct	fft	mat_mul	Average
C_l *(pF)*	3.14	4.14	3.01	3.15	3.36

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

4 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 [2] 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%.

References

1. 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*, 6(4), December 1998.
2. 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*, pages 77–82, Urbana, IL, March 1997.
3. 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*, pages 861–866, Paris, France, February 1998.
4. Jörg Henkel, Haris Lekatsas, and Venkata Jakkula. Encoding schemes for address busses in energy efficient SOC design. In *VLSI-SOC 2001 11th International Conference of Very Large Scale Integration*, Montpellier, France, December 2001.
5. Pinaki Mazumder and Elizabeth M. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall, Inc., 1999.
6. International thechnology roadmap for semiconductors. Semiconductor Industry Association, 1999.
7. Mircea R. Stan and Wayne P. Burleson. Bus invert coding for low power I/O. *IEEE Transactions on VLSI Systems*, 3:49–58, March 1995.
8. 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.
9. Neil H. West and Kamran Eshraghian. *Principles of CMOS VLSI Design*. Addison Wesley, 1998.