Selecting Error Correcting Codes to Minimize Power in Memory Checker Circuits

Shalini Ghosh[1], Sugato Basu[2], and Nur A. Touba[1]

[1]Computer Engineering Research Center,
Dept. of Electrical and Computer Engineering,
University of Texas, Austin, TX 78712-1084.
{shalini,touba}@ece.utexas.edu

[2]Machine Learning Research Group,
Dept. of Computer Sciences,
University of Texas, Austin, TX 78712-1188.
sugato@cs.utexas.edu

Abstract

The approach proposed in this paper reduces power consumption in single-error correcting, double error-detecting checker circuits that perform memory ECC. Power is minimized with little or no impact on area and delay, using the degrees of freedom in selecting the parity check matrix of the error correcting code. The non-linear power optimization problem is solved using two methods, genetic algorithms and simulated annealing. Both the methods are applied to two SEC-DED codes: standard Hamming codes and odd-column-weight Hsiao codes. Experiments on actual memory traces of Spec and MediaBench benchmarks indicate that considering power along with area and delay when selecting the parity check matrix can result in power reductions of up to 27% for Hsiao codes and up to 41% for Hamming codes. Experiments are also performed to motivate the choice of parameters of the non-linear optimization algorithms, using sensitivity analysis of the low-power solutions to the choice of the different parameters of each algorithm.

Keywords: Memory error correction, Hsiao and Hamming codes, low power, non-linear optimization.

1. Introduction

As technology continues to scale with smaller features sizes, lower power supply voltages, and higher operating frequencies, the soft error rate in logic circuits is rapidly increasing [1]. Concurrent error detection using error correcting codes at the outputs of a circuit provides a means to detect soft errors quickly before they have a chance to propagate and compromise the data integrity of a system. Error correcting codes (ECCs) are commonly used to protect against soft errors and thereby enhance system reliability and data integrity [2,3]. S*ingle-error-correcting* and *double-error-detecting* (SEC-DED) codes are generally used for this purpose. These codes are able to correct single-bit errors and detect double-bit errors in a codeword. In this paper, we focus our attention to the ECC checker circuit of a specific type of concurrent error detection: memory ECC, where error correcting codes are used to ensure data integrity during each memory read and write cycle.

A design criterion that has become very important in recent times is power reduction. With increasing miniaturization of devices, power has become a first-order design consideration motivating researchers to look at techniques of reducing power consumption in all components of system design. For memory ECC, power reduction is also an important consideration, since the ECC checker circuit is activated during each read and write access to the memory. In this work, we focus on reducing power in memory ECC checkers. There are many ways to construct SEC-DED codes and implement the corresponding ECC circuitry. While previous research has focused on minimizing area and delay in ECC circuitry, this paper looks at minimizing power in addition to minimizing area and delay. By considering power during the design of ECC circuitry, significant reductions can be achieved at little or no cost in terms of area and delay. For portable electronics this will help extend battery life.

While conventional low power design methodologies that have been developed for general circuits can be applied to the design of error detection circuitry in a straightforward manner, there are some special properties of error detection circuitry that can be exploited to further reduce power consumption. One such property is the fact that error detection circuitry typically contains large amounts of symmetry. For example, parity trees and two-rail checker trees are totally symmetric with respect to their inputs and thus allow complete freedom in the ordering of the inputs. The inputs can be ordered in any way with no change in the function of the circuit and no real impact on the area or delay. This property was first exploited to minimize power in [4]. Favalli and Metra considered signal probability on a level-by-level basis to order the inputs in two-rail checkers to minimize power (the method can also be used for parity trees). In [5], spatial correlation among signals was used for input ordering in parity trees and Berger code checkers. A nice feature of both of these methods is that power is reduced essentially for free as there is no impact in terms of area or delay. The only cost is the time for computing the input ordering.

In [6,7], the problem of reducing power consumption for fault tolerant buses with SEC codes was studied. The bus model that was used considers mutual capacitance effects and assumes transitions between all pairs of vectors are equally likely. The properties of both Hamming codes and dual rail codes with respect to power consumption were analyzed. Results in [7] indicate that for small bus word sizes dual rail codes require less power, while for larger word sizes Hamming codes are better. Another related work is [8], where Hamming codes were designed with the goal of area minimization of the ECC checker in mind.

The goal of this paper is to minimize a joint function of area, delay and power while designing Hamming [9] and Hsiao [10] codes (preliminary results were published in [11]). The proposed approach selects an ***H***-matrix that simultaneously minimizes power, area and delay, using non-linear optimization techniques. The paper is organized as follows: Section 2 gives an overview of the proposed method; Section 3 discusses the ECC memory hardware details; Section 4 gives the details of the optimization algorithms used; Section 5 explains the experimental methodology, the results of which are discussed in Section 6; Section 7 describes experiments performing sensitivity analysis of the results based on variation of parameters of the non-linear optimization algorithms, while Section 8 concludes our discussion and outlines promising areas of future work.

2. Overview of Proposed Method

The focus of our work is reducing power consumption in memory ECC circuitry that provides SEC-DED. Such circuits are widely used in industry in all types of memories including caches and embedded memories. The key design issue is selecting the code that is used. A *(n,k)* linear SEC-DED block code has *n* bits in each codeword consisting of *k* data bits and *n-k* check bits. The code can be represented by a parity-check matrix, ***H***, having *n-k* rows, one for each check bit, and *n* columns, one for each bit in the codeword. In order for the code to be SEC-DED, the ***H***-matrix must be formed in a way that the minimum distance between any codewords is 4. Two well-known methods for constructing a SEC-DED ***H***-matrix were described by Hamming and Hsiao. Different ***H***-matrices result in different area, delay, and power.

2.1 Key Idea

The main idea in this paper is to select the ***H***-matrix in a way that minimizes power, area and delay in the ECC checker. Once the ***H***-matrix has been selected, the corresponding ECC circuitry for implementing the code can be synthesized. The space of ***H***-matrices that provide SEC-DED capability is large. In [10], Hsiao showed that an ***H***-matrix that satisfies the following three constraints provides SEC-DED capability:

1. There are no all-0 columns.
2. Every column is distinct.
3. Every column contains an odd number of 1's (i.e., has odd *weight*).

Hsiao showed that by using minimum odd weight columns, the number of 1's in the ***H***-matrix could be minimized (and made less than a Hamming SEC-DED code). This translates to less hardware area in the corresponding ECC circuitry. Furthermore, by selecting the odd weight columns in a way that balances the number of 1's in each row of the ***H***-matrix, the delay of the checker can be minimized (as the delay is constrained by the maximum weight row).

In this paper, power consumption is considered as an additional factor in selecting the ***H***-matrix. For odd-weight-column codes, there are two degrees of freedom in selecting the ***H***-matrix that can be used to reduce power with little to no impact on area and delay. The first degree of freedom is simply permuting the columns. This has no impact on area or delay as it does not change either the total number of 1's in the ***H***-matrix or the balancing of 1's among the rows. The second degree of freedom is in selecting the odd-weight-columns that are included in the matrix. To minimize area and delay, the smallest odd weight columns should be used first (i.e., weight-1, then weight-3, then weight-5, etc.). However, note that in general, only a subset of the largest odd weight columns will be used. For example, for a (72,64) odd-weight-column code, all $C_1^8 = 8$ of the weight-1 and all $C_3^8 = 56$ of the weight-3 columns will be used, but only 8 of the $C_5^8 = 56$ possible weight-5 columns will be used. Selecting which 8 of the 56 possible weight-5 columns are used in the ***H***-matrix is a degree of freedom that can be used for minimizing power with little to no impact on area or delay.

How much power can be reduced using the degrees of freedom in selecting the ***H***-matrix will depend on the characteristics of the data stored in the memory. The more correlated the data in successive memory reads and writes is, the more power can be reduced through careful selection of the ***H***-matrix. The switching activity (and hence power consumption) in the encoding and decoding logic corresponding to a particular ***H***-matrix depends on which bit transitions occur in the data between successive memory reads and writes.

2.2 Useful Characteristics of Memory Data

In memory data, typically the high order data bit is more likely to be a 0 than a 1 whereas the low order data bit is more likely to have an even distribution between 0 and 1. Sparsity in higher order bits is a very common phenomenon for multimedia applications. In fact, special purpose compilers and architectures with support for variable bitwidth have been studied in order to exploit this characteristic of the multimedia applications [12]. Thus, since the low order bit is more likely to transition in successive memory accesses than the high order bit, it would be better that the low order bit correspond to a lower weight column in the ***H***-matrix and the high order bit correspond to a higher weight column in the ***H***-matrix. This would reduce the switching activity that occurs in the encoding/decoding logic and thereby reduce power. This is a simplistic example to show how selection of the ***H***-matrix can be used to exploit correlations in the data stored in the memory. More elaborate forms of spatial and temporal correlations in the data can be exploited with the proposed methodology.

How much correlation exists in the data stored in a memory will depend on the purpose and function of the memory. Some embedded memories for certain applications may have very correlated data and thus the proposed method for selecting the ***H***-matrix can be very effective in reducing power. Others may have less correlation. The types of data that are stored in different memories ranges broadly. Instruction caches and other memories that primarily contain instructions will tend to have a lot of correlation, as the frequency of execution of different instructions tends to be very skewed. Memories that contain a lot of numerical data will tend to have a lot of spatial correlations among higher order bit positions as the range of the numerical values may be limited and/or skewed. Some embedded controllers and sensors may spend a lot of time executing in a loop and thus have a lot of temporal correlations. No matter what the nature of the memory is, not all transitions will be equally likely, so there will be some scope for power reduction using the proposed method. The actual amount of power reduction will however depend on the extent of the correlation.

2.3 Proposed Method

The proposed technique consists of two steps. The first is to acquire information about the spatial and temporal correlations of the data in memory accesses. The second step is to use that information to select the ***H***-matrix for the odd-weight-column SEC-DED code. Information about the spatial and temporal correlations is acquired by analyzing a sample trace of the memory accesses for a typical workload. The application that will use the memory, or a representative sample of the applications if there are multiple applications, is simulated and a sample trace of memory accesses is obtained. The size of the sample should be chosen so that it is sufficiently representative of the typical workload. The spatial and temporal correlations among the data from the sample traces are then extracted so that the ***H***-matrix can be optimized for the typical workload of the memory. The resulting design of the ECC circuitry will then minimize the average power across the typical workload.

The second step of the proposed method involves selecting the ***H***-matrix once the correlation information has been extracted. This problem is a non-linear optimization problem. In this paper, two optimization techniques are investigated: simulated annealing (SA) and genetic algorithms (GA). Both techniques are described in detail in Sec. 4. Experimental results showed that genetic algorithms outperformed simulated annealing for this problem on the benchmark circuits that were studied, as explained in Sec. 6.

3. ECC Memory Checkers

The goal of this work is to reduce the switching activity in the part of the ECC circuitry that is used most frequently, namely the parity generator block which is used on every memory access (both read and write). Figure 1 illustrates the block-level design of a generic SEC-DED encoder/decoder for ECC memory. The left side of the figure is the processor interface where the relevant signals are *u_data*[63:0], representing the 64 bits of the processor data bus; *rw_n*, representing the memory read/write control signal; and *error-out*[1:0], the 2-bit error flag signal that is required to signal one of possible four error states: (1) no error, (2) correctable data error, (3) correctable parity error, and (4) detectable double error. The right side of the figure is the memory interface consisting of the 72-bit memory data bus *mem_data*[71:0]. The "Generate Parity Bits" block generates the parity bits to store with the processor data during a write cycle. During a read cycle, this block also generates the parity bits for the 64 data bits stored in memory. These generated parity bits are then compared with the stored parity bits to generate the syndrome. In this paper, the focus is on selecting an SEC-DED code that minimizes power consumption in the parity generator block since that is the part of the circuit most heavily used.

Hamming codes and Hsiao codes are commonly used in ECC circuitry. The proposed optimization method is applicable to both of these kinds of SEC-DED codes. Note that the proposed method can be used for any memory word size.

Figure 1. SEC-DEC Block Diagram (modified from [13])

4. Optimization Algorithms

In this paper, two optimization algorithms that are known to give good performance for highly non-linear optimization problems such as the one here are investigated. One is genetic algorithm (GA) and the other is simulated annealing (SA). In this section, we give a brief description of both these techniques and how they were adapted to this domain.

4.1 Cost function

Both the optimization techniques minimize a cost function modeled as a combination of the delay in the circuit, the size of the circuit, and the power dissipation in the circuit. It is a weighted linear combination of the following 3 components, which represent the different design objectives mentioned in Section 2:

1. *Power dissipation:* The power dissipated during ECC checking, which is found by doing power simulation of the permuted inputs through the parity checker circuit. The *power goal* is minimization of this dissipated power.
2. *Size of circuit:* The number of total gates in the circuit, obtained by performing multiple-output logic minimization of the ***H***-matrix equations, using 2-input XOR gates. The *circuit-size goal* is reduction of the number of XOR gates in the total parity checker circuit.
3. *Delay in circuit:* The balance of delay in the circuit, measured by the variance between the depths of the XOR circuits corresponding to different parity equations. The *timing goal is* to minimize the variance in delay between the XOR networks corresponding to the different parity bits to minimize the worst-case delay.

The overall cost function is:

$$(weight_{power} * \text{Power dissipation}) + (weight_{circuit} * \text{Circuit size}) + (weight_{delay} * \text{Circuit delay})$$

$$\text{where: } weight_{power} + weight_{circuit} + weight_{delay} = 1$$

4.2 Genetic Algorithm (GA)

Genetic algorithm (GA) is another popular non-linear optimization tool, useful for such large-scale non-linear optimization problems [14]. In GA, each possible solution to the problem is encoded as a gene. An initial population of *P* random genes is considered, from which a next generation of *P* genes is created by *crossover* and *mutation* operations. We considered a variant of GA where the top *E* best genes (*elites*) at each generation are directly copied into the next generation, thus preserving the best *E* solutions found so far: this GA principle is called *elitism.*

For the purpose of illustration, we will consider $n = 64$ in this section. Note that all these methods can be generalized to work for architectures of other sizes.

4.2.1 Overall GA algorithm

Figure 2 outlines the overall GA algorithm that we use.

Figure 2. Outline of Genetic Algorithm

4.2.2 Gene representation

For Hamming codes, each solution corresponded to a particular input permutation. The *inputGene* corresponding to this is encoded as a string of the mapping for the input memory bits positions. For example, for $n = 64$, one possible permutation could be represented in the *inputGene* by the string “2,3,1,4,5…63,64”, representing the permutation where the 1st memory bit position is mapped to the 2nd input in the checker circuit, the 2nd bit is mapped to the 3rd input, the 3rd bit is mapped to the 1st input, and the other memory bits are mapped to their corresponding circuit inputs.

For Hsiao codes, each solution also contains an additional component that we call the *matrixGene*, representing the design of the $\boldsymbol{H}$-matrix. In our 64-bit architecture example, we first index the 56 possible weight-5 columns in increasing order of their binary representation. The *matrixGene* is represented by the indices of the 8 weight-5 columns out of the 56 possible ones that are selected to fill up the last 8 positions of the $\boldsymbol{H}$-matrix (after filling up the first 64 with all possible weight-1 and weight-3 columns). So, a possible *matrixGene* would be “1,4,6,9,11,34,53,55”, representing the indices of the particular weight-5 columns selected while creating the $\boldsymbol{H}$-matrix. In the general case, for architectures of other sizes, the *matrixGene* would have a representation of a similar design choice of selecting some columns from a total set of possible odd weight columns.

In the case of Hsiao codes, the GA algorithm performed simultaneous *crossovers* and *mutations* of both the *inputGene* and the *matrixGene*. The *fitness* of a composite gene, comprised of the *inputGene* and the *matrixGene*, was considered to be the inverse of the total cost calculated as shown in Section 4.1.1, so that genes with less cost ended up being “more fit”.

4.2.3 Mutation operation

The *mutation* operation for the *inputGene* creates a child from a single parent, by choosing two input index mappings at random in the parent gene and swapping them. For example, swapping the 1st and 4th positions in the example gene considered above would generate the mutant *inputGene* "4,3,1,2,5…63,64" from the parent "2,3,1,4,5…63,64".

In the case of Hsiao codes, the *mutation* operation for the *matrixGene* creates a child from a single parent by selecting a column index at random and removing it from the selected set, bringing in a column from the unselected set. For example, swapping out the weight-5 column having index 3 and swapping in the column with index 4 in the example gene considered above would generate the mutant child "1,3,6,9,11,34,53,55" from the parent *matrixGene* "1,4,6,9,11,34,53,55".

4.2.4 Crossover operation

The *crossover* operation for the *inputGene* creates a child from two parents, trying to incorporate good features of both. We chose a *crossover* function where the mean of the positions in the two *inputGenes* was first computed, and the child was created by considering the sorted indices of the computed mean. In our example, the *crossover* between "3,1,2,4,5…63,64" and "1,2,3,4,5…63,64" would generate an intermediate mean [2.0,1.5,2.5,4.0,5.0…63.0, 64.0], for which the corresponding sorted indices would be "2,1,3,4,5…63,64" (where the ties between same values are broken arbitrarily).

Notice that in this crossover function, the child has the common feature of both the parents, i.e., matrix bit positions 4-64 are mapped to circuit input 4-64. If both the parents had low cost and this was a feature responsible for it, then the child would also inherit this feature.

In the case of the *matrixGene*, the *crossover* function appends the *matrixGenes* of both the parents (representing the selected columns) with the indices of the unselected columns. Then, an average and index-sorting operation similar to the *inputGene* is performed, after which the first 8 positions of the result are selected to get the *matrixGene* of the child. It can be easily shown that this operation is a valid *crossover* function for the subset-selection problem that underlies the design of the ***H***-matrix for Hsiao codes.

4.3 Simulated Annealing (SA)

For Hamming codes, we consider the ***H***-matrix in the standard form and the only thing that can be varied to reduce power is the input ordering. For 64 inputs, there are 64! possible input permutations, which makes an exhaustive search of the input ordering with the lowest power dissipation intractable.

For Hsiao codes, along with input ordering, we have the additional flexibility of designing the ***H***-matrix. As explained in Sec. 2, selecting which 8 of the 56 possible weight-5 columns are used in the ***H***-matrix for a (72,64) code is a degree of freedom that can be used for minimizing the dissipated power. So, the search space is even larger in this case, having (64! * C_8^{56}) possible solutions.

To solve this large non-linear optimization problem, we applied simulated annealing [15] to find a (local) optimum of the cost function.

5. Experimental Methodology

We ran experiments on 5 sample programs from the Spec 1995 and 2000 architecture benchmark suites: compress, perl, go, gcc and anagram, and 5 benchmarks from the MediaBench multimedia benchmark suite [16]: decode, encode, epic, cjpeg and rawcaudio. We used the architecture tool *SimpleScalar* [17] to simulate a 64-bit architecture, and for each program all the memory read and write accesses were recorded. These memory traces are the inputs to the "Generate Pariy Bits" block of the ECC checker circuit, which generates 8 parity-check bits corresponding to each 64 bit-wide memory word.

During estimation of the cost of each solution in the SA and the GA algorithms, the circuit corresponding to each ***H***-matrix was synthesized by multiple-output logic minimization with 2-input XOR gates as basic components using *sis* [18]. For each benchmark, the best input permutation and ***H***-matrix was obtained for both the Hsiao code and the Hamming code. The SA algorithm was initialized from a random solution, and the temperature was increased until the system "melted" [19]. Subsequently, the temperature was reduced in a Cauchy schedule and annealing was performed till convergence or for 500 time-steps, whichever occured earlier. The GA algorithm was run with the following parameters: size of population K = 250, number of elites E = 5, number of mutant children M = 50, number of filtered unfit parents = bottom 100, number of generations G = 200. The GA parameters were chosen after performing pilot studies of the sensitivity of the results to the algorithm parameters for a representative benchmark, details of which are analyzed in Sec. 8.

For both SA and GA, the performance of the final best solution of minimum cost was compared to 100 random solutions. For Hamming code, this corresponded to 100 random input orderings of the standard form Hamming code, whereas for the Hsiao code this corresponded to 100 random minimum odd-weight-column ***H***-matrices having a random input ordering. These random solutions emulate the convention design procedure that arbitrarily selects a code with minimum area and delay, but with no consideration of power.

The combined cost function was used in the experiments so that circuit size, delay and power were simultaneously minimized. Power was estimated by the number of transitions in the outputs of the XOR gates of the checker circuit (this corresponds to the dynamic power, leakage power was not considered), size of the circuit by the number of XOR gates in it, and maximum circuit delay by the maximum level among the XOR networks implementing each parity equation. Note that it was assumed that the inputs to the checker are synchronized coming from a register and glitch power was not considered. Hsiao and Hamming codes were studied as the two underlying SEC-DED codes of the ECC checker.

6. Discussion of Results

In this section we discuss the different experiments that were performed and their results.

6.1 Benchmark Characteristics

Figure 3 shows the characteristics of the 4 representative benchmarks from the set of 10 that we have considered, two each from Spec and MediaBench. The plots on the left show the signal probabilities in the columns of the memory trace matrix of the benchmark programs, sorted in increasing order. To generate each graph, the signal probability (i.e., probability of 1's) is computed for every column, and then the columns are sorted in increasing order of signal probability from left to right in the plot. For perfectly random inputs, each column in the input trace matrix would have 0.5 fraction of 1's, since 1's and 0's would be equally probable. The skewness of this distribution demonstrates that there is an uneven distribution of the number of 1's in different columns of the input memory trace matrix. The power optimization algorithms exploit this during re-ordering.

The plots on the right in Figure 3 are histograms that show the pairwise correlations between the columns. The histograms were constructed as follows: for each pair of columns in the input memory trace, we counted how many transitions between 1 and 0 (and vice versa) would occur if we placed an XOR gate between the two columns. The histogram counts how many column pairs have their fraction of transitions in each bin range. If any two columns in the input trace matrix were independent, then the proportion of transitions would be 0.5 x 0.5 = 0.25. The corresponding histogram would have all the frequency concentrated at the 0.25 bin. In this case, the distribution of histogram frequencies in multiple bins for different benchmarks, ranging from 0.04 to 0.34, demonstrates that there is significant useful correlation between the input columns, which is useful for power optimization.

Figure 3. Bit-wise profiles and pair-wise histograms for representative benchmarks

6.2 Comparison of GA and SA

For both GA and SA, the number of transitions in the ECC checker circuit corresponding to the best solutions was compared with the average (of 100) random number of transitions and the worst (out of 100) random number of transitions. As can be seen from the results in Table 1, GA gave 12% to 27% power reduction on the different benchmarks with respect to the average random transitions, and 14% to 34% reduction with respect to the worst-case random transitions. For this experiment, GA has much better performance than SA, which gave 1% to 14% power reduction with respect to average random, and 2% to 22% improvement compared to worst-case random. One possible reason for the better performance of GA over SA in this case could be that for Hsiao codes, the total power in the circuit is a highly non-linear and discontinuous function of the input ordering and choice of $\boldsymbol{H}$-matrix. Due to this, the gradient may not be well defined at every point on the cost function. So SA, which essentially performs a non-greedy gradient descent, does not perform very well. In comparison, GA's are known to perform well for cost functions with such characteristics. Moreover, the best E solutions found at every step of GA (E =5 in our experiments) are deterministically remembered by using *elitism*, which is an added advantage of GA over SA in this case.

In general, the results show that power savings in GA increase with increasing size of the benchmark traces. A possible reason for this could be that the inputs get more correlated as the size of the benchmark traces increases, thereby giving more scope to the GA algorithm to perform better optimization.

Table 1. Results of GA and SA on Hsiao code with overall cost function

6.3 Ablation Experiments

To study the effectiveness of the combined cost function to reduce power, area and delay, we ran ablation experiments on the (72,64) Hsiao code where we individually considered only the power, circuit size, and circuit delay components of the GA cost function. The results of these experiments are shown in Table 2. Note that in each individual optimization, we obtain values that are generally better than the corresponding values obtained using the overall cost. For example, for the benchmark program encode, the overall cost minimization gave power savings of 14%, a circuit size of 172 gates and the maximum number of levels to the output was 7. In comparison, individual minimizations of power, circuit size and maximum number of levels gave 15% power savings, 171 gates and 6 maximum number of levels respectively, which are individually better than their corresponding results for the combined cost function. However, when one component in the cost function is individually minimized, the other two components can have highly non-optimal values since the cost function does not consider them at all during the optimization process (as shown by the negative power savings, i.e., *increase* in power dissipated with respect to random, in many cases, if only delay is minimized). The overall cost function gives a good tradeoff between minimization of power and satisfaction of the other design requirements. Note that the weights of the 3 components of the cost function gives the designer the flexibility to incorporate specific design choices, e.g., more importance to power minimization over circuit size or delay minimization.

An interesting observation in the ablation study is that for compress and go, the power reduction for the combined cost function is better than the power reduction for individual power minimization. This apparently seems like an anomaly, but it can be explained as follows. Since power is a highly non-linear function of the input permutation and the choice of ***H***-matrix, there are many local minima in this function. The reduction of only the power component limits the search of the GA, and can cause the GA to sometimes get stuck in local minima. In these cases, using the combined cost function can help the optimization algorithm to get out of such local minima and get to a better optimum, as we see in the case of the compress and go benchmarks. So, apart from finding a good overall minimum of the various design components (power, circuit size, maximum delay), using the combined objective function also facilitates the GA algorithm to get do a better exploration of the search space, which enables it to avoid bad local minima more effectively in some cases.

Table 2. Results of GA on Hsiao code with individual cost functions

6.4 Performance on Hamming Code

In the next set of experiments, we ran the GA algorithm on the (72,64) standard Hamming code, for each of the benchmark circuits. Table 3 shows the power savings on the Hamming code, which are between 5% and 41% for the different benchmark circuits, are better than the corresponding power savings for the Hsiao code in most cases. One possible explanation for that is that the Hsiao code is well optimized in terms of balance of gates between the different parity circuits. In comparison, Hamming codes have a large skew in the number of XOR gates in different parity equations, which can be exploited by the optimization algorithm more effectively. Moreover the standard Hamming code typically produced larger circuits when synthesized, which also gave the GA algorithm a larger search space where it could produce solutions significantly better than random.

Table 3. Results of GA on Hamming code with overall cost function

7. Sensitivity of Solution to Algorithm Parameters

In order to study the sensitivity of the GA solution to different values of the algorithm parameters, we varied each parameter while keeping the other parameters fixed. Table 4 shows the results of these experiments on a sample Spec benchmark circuit *compress*. We successively varied the number of generations, the size of the population, the number of elites and the number of mutants, each time keeping the other parameters fixed. From this pilot study, we selected the values of the parameters that were used in the other experiments. This study also shows the sensitivity of the GA solution to the different parameters of the algorithm. For example, the GA solution converges to the best value after a particular number of times steps, is not very sensitive to the number of elites, works best with an optimum size of the population and number of mutants, etc.

For SA, varying the number of time steps gave increasing power savings with respect to the average random solution, and the SA solution reached convergence near 500 time steps for the benchmark *compress*. So we used 500 as the maximum number of time steps for the other benchmarks.

Table 4. Variance of GA power reduction with different parameter values, for representative Spec benchmark *compress*

Table 5. Variance of SA power reduction with different parameter values, for representative Spec benchmark *compress*

8. Conclusions

Overall, our experiments demonstrate that there is significant correlation among memory traces for the benchmark applications we studied, and that optimizing the input permutation and the design of the ***H***-matrix of the memory ECC checker using GA with a combined cost function gives us significant power reduction, while simultaneously minimizing the overall size of the circuit and the circuit delay. Note that both SA and GA are relatively slow optimization procedures, but the optimization is performed offline and it is the optimized ***H***-matrix that is deployed in the online error correction phase. So, speed of the optimization algorithm is not a major issue for the problem we are studying, implying that more sophisticated search or optimization techniques could be employed if necessary.

An area for future work is to extend the technique described here to handle memory ECC for the ChipKill server architecture [20] and for other error-correcting codes, e.g., Reed-Solomon codes, Fire codes, etc. For some of these codes, the proposed scheme will have be modified to handle certain characteristics of the codes, e.g., for byte error-correcting codes, *b*-byte column groups of the ***H***-matrix would have to be permuted instead of the columns being permuted directly. However, it would be relatively straightforward to frame the problem of finding the "best" ***H***-matrix in these cases as an optimization problem, which can be solved by simulated annealing (SA) or genetic algorithms (GA).

Another interesting area of future research is the study of how the presence of caches would affect the correlation in the data input to the ECC memory, and whether there is any systematic pattern there that can be exploited by the optimization algorithms.

Note that our method considers only dynamic power and does not factor in glitch and leakage power. Leakage power is becoming an increasingly important factor as technology is scaling down and future work is needed to address it.

Acknowledgements

This research was supported in part by the Hewlett-Packard Corporation, and in part by the National Science Foundation under Grant No. CCR-0306238.

References

[1] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. Proceedings of the International Conference on Dependable Systems and Networks, pp. 389 – 398, (2002)

[2] C. L. Chen, and M.Y. Hsiao. Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. IBM J. of Res. and Develop., vol. 28, no. 2, pp. 124-134 (1984)

[3] K. Gray. Adding Error-Correcting Circuitry to ASIC Memory. IEEE Spectrum, pp. 55-60 (2000)

[4] K. Favalli, and C. Metra. Design of Low-Power CMOS Two-Rail Checkers. Journal of Microelectronics Systems Integration, vol. 5, no. 2, pp. 101-110 (1997)

[5] K. Mohanram, and N.A. Touba. Input Ordering in Concurrent Checkers to Reduce Power Consumption. Proc. of IEEE Symposium on Defect and Fault Tolerance, pp. 87-95 (2002)

[6] D. Rossi, V.E.S. van Dijk, R.P Kleihorst, A.K. Nieuwland, and C. Metra. Coding Scheme for Low Energy Consumption Fault-Tolerant Bus. Proc. of International On-Line Testing Workshop, pp. 8-12 (2002)

[7] D. Rossi, V.E.S. van Dijk, R.P Kleihorst, A.K. Nieuwland, and C. Metra. Power Consumption of Fault Tolerant Codes: the Active Elements. Proc. of International On-Line Testing Symposium, pp. 61-67 (2003)

[8] R. Kleihorst, and N. Benschop. Fault Tolerant ICs by Area-Optimized Error Correction Codes. Proc. of International On-Line Testing Workshop, pp. 143 (2001)

[9] R.W. Hamming. Error Detecting and Error Correcting Codes. Bell System Tech. J., 29, 147 (1950)

[10] M.Y. Hsiao. A Class of Optimal Minimum Odd-weight-column SEC-DED Codes. IBM J. of Res. and Develop., vol. 14, no. 4, pp. 395-401 (1970)

[11] S. Ghosh, S. Basu and N.A. Touba. Reducing Power Consumption in Memory ECC Checkers. Proc. of the International Test Conference, pp. 1322-1331 (2004)

[12] M. Stephenson, J. Babb, and S.P. Amarasinghe. Bidwidth analysis with application to silicon compilation. PLDI, pp. 108-120 (2000)

[13] Xilinx Applications Note: CoolRunner-II CPLD. Single Error Correction and Double Error Detection (SECDED) with CoolRunner-II CPLDs. XAPP383 (v1.1) (August 1, 2003)

[14] J.H. Holland. Adaption in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, USA (1975)

[15] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi Jr. Optimization by Simulated Annealing. Science, pp. 671-680 (May 1983)

[16] C. Lee, M. Potkonjak, and W.H. Mangione-Smith. MediaBench: A Tool for Evaluating Multimedia and Communications Systems.

Proc. Of Micro 30 (1997)

[17] D. Burger, T.M. Austin, and S. Bennett. Evaluating Future Microprocessors: the SimpleScalar Tool Set, TR-1308, Univ. of Wisconsin-Madison, CS Dept. (1996)

[18] E. M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, P.R. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report Memorandum No. UCB/ERL M92/41, University of California, Berkeley (1992)

[19] H. Szu, and R. Hartley. Fast Simulated Annealing. Physics Letters A, vol. 122, no. 3,4, pp. 157 – 162 (1987)

[20] T.J. Dell. A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. IBM Microelectronics Division (1997)

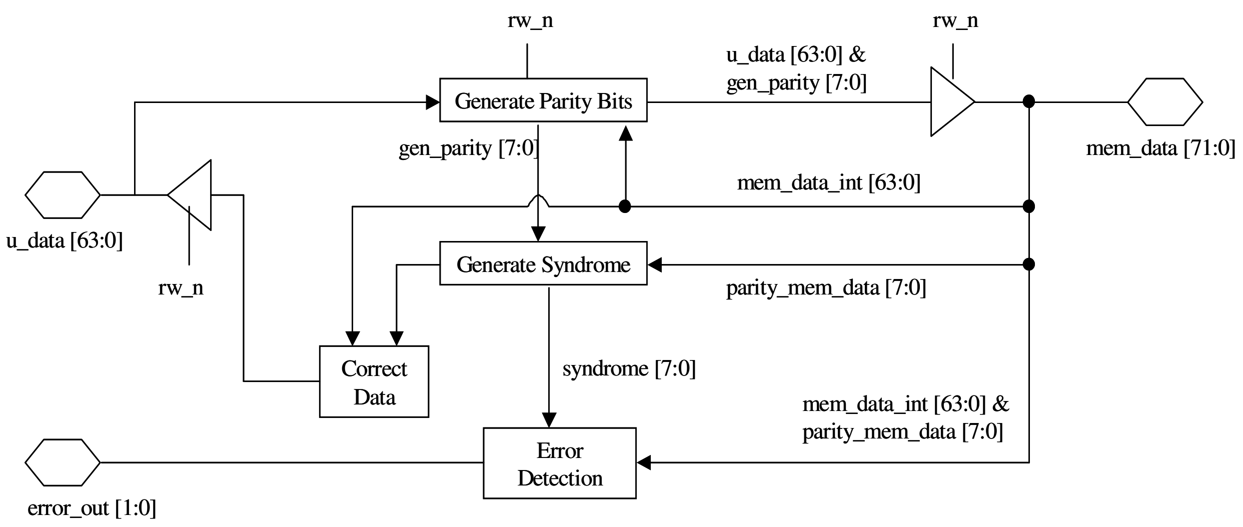

Figure 1. SEC-DEC Block Diagram (modified from [13])

Input: Initial population of *K* random genes, the number of elites *E*, the number *mutant* children *M*, the number of *unfit* parents *U*, the number of generations *G*
Output: gene with maximum fitness (minimum cost)
Algorithm:

```
1. for i = 1 to G
2.      Sort the K parent genes in decreasing order of fitness (increasing order of their cost)
3.      Copy the top E elite parents into the next generation
4.      Create M children by direct mutation of the E elites
5.      Reject bottom U parent genes as unfit
6.      Create remaining K-E-M children by crossover between any 2 parents that are not elites and not unfit
7. Return elite with maximum fitness (minimum cost)
```

Figure 2. Outline of Genetic Algorithm

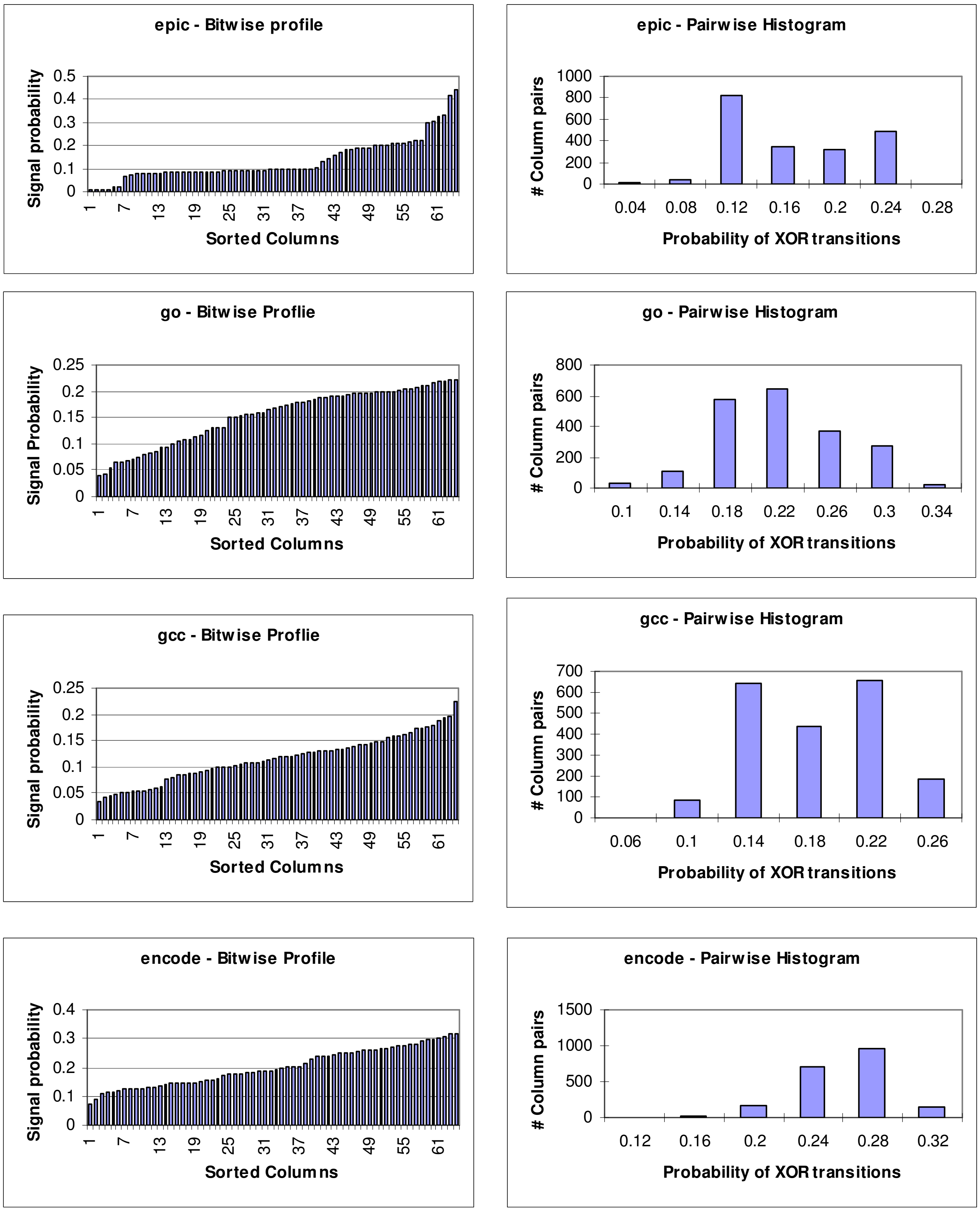

Figure 3. Bit-wise profiles and pair-wise histograms for representative benchmarks

Table 1. Results of GA and SA on Hsiao code with overall cost function

Benchmark Details		Number of transitions for Random Solution		Number of transitions for Optimized Solution		SA power reduction		GA power reduction	
Name	Memory trace size	Average	Worst	SA	GA	w.r.t Average Random	w.r.t Worst Random	w.r.t Average Random	w.r.t Worst Random
gcc	187089	6760149	7353330	6414782	5042942	5.1%	12.8%	25.4%	31.4%
go	118897	5449560	5852080	5156709	4253540	5.4%	11.9%	22.0%	27.3%
anagram	94041	4953000	5097104	4589887	4358206	7.3%	10.0%	12.0%	14.5%
compress	72193	1518005	1622810	1383552	1222844	8.9%	14.7%	19.4%	24.7%
perl	27657	1186947	1228706	1159770	1014818	2.3%	5.6%	14.5%	17.4%
epic	470633	17111347	18898620	14700707	12445636	14.1%	22.2%	27.3%	34.2%
cjpeg	18273	935956	975516	920797	759018	1.6%	5.6%	18.9%	22.2%
encode	7569	399527	417434	394907	329548	1.2%	2.3%	17.5%	21.1%
decode	4745	237972	251488	234402	195610	1.5%	6.8%	17.8%	22.2%
rawc-audio	2233	141921	147616	121390	118396	14.5%	17.8%	15.6%	19.8%

Table 2. Results of GA on Hsiao code with individual cost functions

Benchmark Name	Minimize power only			Minimize delay only			Minimize circuit size only			Minimize combined cost function		
	Power reduction	Ckt. size	Max levels	Power reduction	Ckt. size	Max levels	Power reduction	Ckt. size	Max levels	Power reduction	Ckt. size	Max levels
gcc	27.5%	172	7	-4.0%	172	7	0.9%	171	8	25.4%	171	7
go	19.8%	172	8	-1.9%	173	7	4.6%	171	8	21.9%	172	7
anagram	14.1%	172	8	1.9%	172	7	1.7%	171	7	12.0%	174	7
compress	18.6%	174	7	3.3%	174	7	1.0%	171	7	19.4%	175	7
perl	14.6%	172	7	0.4%	173	7	2.7%	171	8	14.6%	172	7
epic	30.5%	172	8	-1.2%	172	7	3.3%	171	7	27.3%	173	7
cjpeg	18.1%	172	8	-1.4%	172	7	1.7%	171	8	18.1%	172	8
encode	15.2%	173	7	-2.7%	173	6	4.2%	171	7	14.2%	172	7
decode	18.1%	172	7	-1.4%	173	6	2.5%	171	8	17.8%	171	7
rawcaudio	17.7%	173	7	1.4%	173	7	3.7%	170	7	15.6%	172	7

Table 3. Results of GA on Hamming code with overall cost function

Benchmark Name	Average random solution #transition	GA solution #transition	Hamming code power reduction
gcc	5793700	3408358	41.2%
go	4663511	2764384	40.7%
anagram	4120663	2810202	31.8%
compress	1161115	1097838	5.4%
perl	983159	710168	27.8%
epic	14792749	8628410	41.7%
cjpeg	783432	543062	30.7%
encode	336843	223354	33.7%
decode	201138	127272	36.7%
rawcaudio	120058	79654	33.6%

Table 4. Variance of GA power reduction with different parameter values, for representative Spec benchmark *compress*

Parameter	Values of the parameter	Circuit Size	Number of levels	GA power reduction with respect to Average Random (%)
Number of Generations	Number of Generations = 50	174	7	16.9
	Number of Generations = 100	174	7	19.3
	Number of Generations = 150	174	7	19.3
	Number of Generations = 200	175	7	19.4
Number of Elites	Number of Elites = 3	174	7	19.3
	Number of Elites = 5	175	7	19.4
	Number of Elites = 7	175	7	19.4
	Number of Elites = 9	175	7	19.5
Size of Population	Size of Population = 50	174	7	16.8
	Size of Population = 150	174	7	17.4
	Size of Population = 250	175	7	19.4
	Size of Population = 350	175	7	19.3
Number of Mutants	Number of Mutants = 10	174	7	16.4
	Number of Mutants = 50	175	7	19.4
	Number of Mutants = 75	175	7	18.6
	Number of Mutants = 100	174	7	18.6

Table 5. Variance of SA power reduction with different parameter values, for representative Spec benchmark *compress*

Parameter	Values of the parameter	Circuit Size	Number of levels	SA power reduction with respect to Average Random (%)
Number of time steps	Number of time steps = 100	174	8	7.7
	Number of time steps = 200	173	8	8.1
	Number of time steps = 300	172	8	8.7
	Number of time steps = 400	173	8	8.8
	Number of time steps = 500	173	7	8.9
	Number of time steps = 600	174	7	8.9