Hardware Fault Compensation
Using Discriminative Learning

Farah Naz Taher    Joseph Callenes-Sloan
Department of Electrical Engineering
The University of Texas at Dallas
Email: {farah.taher, jcallenes.sloan}@utdallas.edu

Abstract—With process scaling and the adoption of post-CMOS technologies, permanent faults are becoming a fundamental problem. Circuits containing defects are either discarded (reducing yield) or partially disabled (reducing performance). In this paper, we propose a general approach using supervised and discriminative learning techniques to compensate for the effect of permanent faults on a circuit’s output. The insight for this approach is that many emerging systems and applications are able to tolerate some loss of quality in their computed results. Therefore, more scalable and lower overhead compensation techniques may be used to approximately correct for the effect of hardware faults on the circuit output. The proposed approach is shown to improve the output quality of complex accelerator and application-specific logic by 2-3 orders of magnitude while incurring <10% area overhead and <3% performance overhead.

Index Terms—Hardware Fault, Error Tolerance, Approximate Computing, Error Compensation, Supervised learning, Machine Learning.

I. INTRODUCTION

With process scaling, significant challenges are arising in the production of reliable transistors and circuits. Due to fundamental limitations in device physics and the manufacturing process, future silicon technology is likely to contain much less reliable logic and increased occurrences of permanent faults during and after fabrication [5], [6]. Spot and bridging faults due to process variations, lithography, and silicon impurities are the most well-known faults causing permanent faults in circuits [23].

The ITRS roadmap 2013 edition [1] predicts: “The ultimate nanoscale device will have high degree of variation and high percentage of non-functional devices right from the start” and “Circuits that can dynamically reconfigure itself to avoid failing and failed devices (or to change/improve functionality) will be needed”.

In order to guarantee proper functionality and overall reliability with current process technology, increasingly large amounts of resources are invested in testing circuits. After testing, depending on the severity of the defects found, faulty circuits are typically either discarded (providing reliability at the cost of reducing yield), or in some cases, partially disabled (providing reliability at the cost of performance) [12], [13]. Partially disabling defective portions of faulty chips allows manufacturers to do product binning and sell the lower performing part at a lower price (e.g. a processor or GPU with less parallel functional units). Much prior work has been done on reducing the cost of testing, e.g. reducing the time required for test by using built-in self test (BIST) hardware designs.

However, as the rate of permanent faults increases, due to increasing variability and device counts, there will be an increasingly significant cost associated with yield and performance loss. For this reason, testing for the occurrence of faults will alone likely not be sufficient to reduce the cost of providing reliable circuits. Instead, other disruptive approaches for tolerating faults will likely be needed.

The focus of this paper is on efficient correction or compensation for the effect of permanent faults on a circuit output. We first discuss one application-specific approach that exploits linearity in an algorithm, by translating the effect of an error to the output result (i.e. as a single correction factor). However, this approach is limited by requiring intermediate signal state that can be difficult to determine a priori. Therefore, we then focus on general approaches for compensating for the effect of permanent faults on approximate circuits.

Many emerging systems and applications are able to tolerate some loss of quality or optimality in their computed results (e.g. multimedia, DSP, data mining, synthesis, and recognition applications). Many researchers have already begun to explore energy/accuracy trade-offs with approximate applications [4], [19]. In the context of timing errors in DSP and multimedia systems, researchers have shown how statistical methods can be used to build compensation models and improve overall accuracy [20].

We investigate an approach for building compensation logic for approximate computing systems and applications using supervised learning algorithms (i.e. inferring a compensation function based on the set of circuit behavior training data acquired during testing). We call this approach discriminative error compensation (DEC). As opposed to using statistical methods, accurate low-complexity models can be developed directly without having to model error, input conditional, and joint probability distributions. For this reason, discriminative learning can scale better for many problems and produce more accurate results than a statistical (i.e. generative) learning approach. Much prior work has shown that the choice of learning algorithm is highly application dependent and likely an ensemble of different models will be required [16], [18].

This paper makes the following contributions:

- We introduce the idea of reducing the impact of permanent faults by using discriminative learning to build
compensation function and show it can provide up to a 2-3 order magnitude reduction in output error and in many cases scale better than generative models.

- We show an example implementation of a general purpose and adaptable error compensation block, which can be used for individual ASICs or System-on-Chip based architectures containing various faulty accelerator hardware.
- We perform gate-level fault injection experiments for a variety of circuits and provide experimental results showing that for complex accelerators and application-specific logic, significant error reduction can be provided with an error compensation module requiring <10% area and <3% online performance overhead.

The rest of the paper is organized as follows. We introduce hardware fault compensation model in Section II. Related works are stated in Section III. Next, we explain the methodology and simulation results in Section IV and V. Section VI concludes.

II. HARDWARE FAULT COMPENSATION

In this section, we present an approach for producing reliable circuits by compensating for the impact of permanent faults on the circuit output. Consider a faulty computation module, shown in Figure 1, with input (x) and ideal output (y0). Permanent faults in the computational module may result in an input dependent error (ε), in the actual output (yf), i.e. yf = y0 + ε. The fault compensation module, Figure 1, takes as input some subset of the faulty modules input (x), intermediate state (xi), and the faulty output (yf) and produces an output (ye), which is equal to y0.

\[ y_0 = \sum_{i=0}^{n} c_i x_{n-i} = c_0 x_n + c_1 x_{n-1} + c_2 x_{n-2} + \ldots + c_n x_0 \]

where x is the serial input stream and c is the set of coefficients. Assume a permanent fault exists in the \( k \)th bit, near the beginning of the FIR input stream (e.g. the initial input flip-flop). As the input is streamed into the filter, the fault may also corrupt every following input (i.e. if \( x_0^k = 0 \) and the fault is a stuck-at-1 fault, there will be an error of \( \varepsilon' = 2^k \) induced in the input otherwise the error is zero).

\[ y_f = c_0(x_n + x_k \varepsilon') + c_1(x_{n-1} + x_k \varepsilon') + \ldots + c_n(x_0 + x_k \varepsilon') \]

Now we can transform the equation to separate the error from the correct output by exploiting the linearity of the computation (i.e. associativity and distributivity properties) and calculate the correct output by subtracting the error term as follows:

\[ y_f = (c_0 x_n + c_1 x_{n-1} + \ldots + c_n x_0) + (c_0 x_k \varepsilon' + c_1 x_{k-1} \varepsilon' + \ldots + c_n x_0 \varepsilon') \]

\[ y_0 = y_f - (c_0 x_k \varepsilon' + c_1 x_{k-1} \varepsilon' + \ldots + c_n x_0 \varepsilon') \]

A. Compensation by Algebraic Transformations

In this section, we describe one approach for creating an error compensation module by using algebraic transformations. Specifically, we can exploit the linearity property of linear operations within an application to separate the constant error factor from the circuit’s correct functionality.

Consider a linear circuit, a FIR filter, which is represented as follows:

Fig. 1: Fault Compensation Overview.

This approach also has the advantage that it can be used for exactly compensating for the effect of hardware fault. However, in order to use this type of approach, the compensation logic also requires connections for several potential intermediate node states (e.g. \( x_k^i \)). The location of the intermediate nodes needed for the compensation are determined by the location of the permanent faults. The presence and location of permanent faults can only be determined post silicon, so the compensation logic would require extra signal connections out of any potentially faulty node in the original circuit design. Therefore, although algebraic-based compensation may be used for certain faults within circuits using linear operations, it is limited by requiring a more intrusive design with significant amounts of shared state (e.g. not only input and output...
B. Compensation by Supervised Learning

In the post-silicon validation phase, a large number of chips are discarded or disabled due to random fabrication defects. By compensating for the impact of the faults on the circuit output many of these faulty chips may be refurbished and utilized. In this section, we propose a general approach to build an error compensation module using supervised learning. The insight for this approach is that many emerging applications require only approximate results (e.g., results within a certain distance of the exact result). Therefore even though hardware faults may disrupt the original intended functionality of the design, a low overhead compensation logic that reduces the amount of error in the output, may still be acceptable for the application. Supervised learning allows for a fast, flexible, and scalable way for generating accurate compensation routines which are specific to the particular set of application inputs and permanent faults.

In supervised learning, a map between a set of input attributes and an output variable is predicted to the unseen data or in other words to build a brief model of the distribution of class labels according to predictor features [8], [15]. Before testing, a fault-free golden chip is first analyzed and the outputs of the circuit are generated according to a set of typical inputs. Chips containing permanent faults will be analyzed using the same input pattern and the faulty data output set will be measured. Each of these data sets (i.e., the input pattern ii. the faulty output set iii. the fault-free output set) will then be used for learning the best compensation function assuming only the input and output state.

A supervised learning algorithm can be used to determine the best model to compensate for the given set of permanent faults. In the post-silicon phase, if the circuit is found to be faulty \( y_f = y_0 + e \), the compensation block can be configured to compensate for faults and provide a corrected output near the actual correct output:

\[
|y_f - y_0|^2 >> |y_e - y_0|^2
\]

Both linear and non-linear classification/regression algorithms can be used for compensation block depending on the characteristics of the application and set of permanent faults. Linear regression algorithms model the relationship between multiple variables and fits the relationship in a linear equation. In non-linear regression, the data set is pruned into nodes which represent tests of classification features and each leaf represents either a model tree, a regression tree, or a simple linear regression. In this paper, we use both linear regression and decision/regression (i.e., Reptree) for building the discriminative error compensation routines.

As described in Section II, a subset of the faulty modules state may be used to build a compensation model. Depending on which subset is used, the hardware configuration may also change. We consider four possible feature subsets for compensation (and corresponding hardware changes):

1) Use the faulty output directly without any compensation (i.e., disable compensation module).
2) Use supervised learning to compensate the actual output based on the input and faulty output.
3) Use supervised learning to compensate the actual output based on the input only (i.e., disable the faulty module).
4) Choose the best model for compensation based on the fault and circuit.

The compensation block is designed and fabricated along with the actual module. Post-silicon, after testing phase, if the circuit is found faulty; the relevant coefficient set for compensation calculation is loaded to the compensation block.

C. The Compensation Architecture

In this section, the hardware architecture of the compensation based on supervised learning is described. The compensation modules described in Section II-B could be utilized in an ASIC that is used for approximate computation. Given the trend toward System-on-Chip (SoC) architectures (which may contain numerous approximate accelerators), another important usage scenario for the compensation module is as a shared component in a SoC (see Figure 2). By sharing the compensation module among the accelerators, the area costs would be amortized. The hardware architecture for the compensation module using linear regression and regression/classification tree are described below.

1) Linear Regression: Linear regression predicts the response to \( n \) data points by a regression model given by, \( y = a_0 + a_1.x_1 + a_2.x_2 + \ldots + a_n.x_n \) where \( a_0, a_1, \ldots, an \) are the constants of the regression model which reduces the difference between faulty and actual value. With a circuit that has \( n \) inputs and \( m \) outputs, there will be \( n+2 \) number of attributes where, \( n \) inputs will be of 1-bit, 1 will be \( m \) bits (representing the faulty output) and 1 more for the constant. Figure 3 shows the hardware architecture of the linear regression algorithm.

‘Weight Module’ contains the memory used to store the attribute values calculated from the machine learning. The size of this memory will depend on the number of attributes of the circuit.
Fig. 3: Hardware Architecture: Linear Regression.

All the input bits will have a corresponding weight deduced by the algorithm. If the bit value is 0, the weight is ignored. If the weight of the corresponding input bit that has a value of 1, a multiplexer-accumulator combination, ‘Multiplexer-Accumulate Module’, will accumulate the value in a register.

‘MAC Module’ performs multiply-accumulate operation where the output instance multiples with its weight values and accumulated in a register.

2) Decision/Regression Tree: In our approach, we have used two tree algorithms. J48 and RepTree. J48 is a classification tree for generating a pruned or unpruned C4.5 decision tree. RepTree is an algorithm which can build both decision and regression tree using information gain/variance back-fitting and pruning. J48 and RepTree has difference in there algorithms for tree generation but can be implemented with the same hardware.

Figure 4 shows the hardware architecture of the decision tree algorithm.

‘Node Module’ will consist of comparators and multiplexers of the number of leaf. The comparator will compare the faulty outputs in order to detect which leaf an instance falls into and the Multiplexer tree will compare the input bits based on value 1 or 0 to select the corresponding leaf module.

‘Weight Module’ contains the memory used to store the attribute values calculated from the supervised learning. The size of this memory will depend on the number of attributes used from the circuit. The output weight module consists of the output values set from supervised learning.

III. RELATED WORK

There has been much work on detecting and correcting errors in hardware. Many of the techniques focus on using structural or information redundancy (i.e. error correcting codes) to make a particular circuit or subsystem reliable [11], [22]. In test-related applications, parity trees have been used extensively in the past [3]. While much prior work focuses on the detection of permanent faults, this paper focuses on compensating for the effect of the permanent faults on the circuit output.

Some past work has addressed the use of error compensation in the context of digital signal processing systems (DSP) [2], [20], [25]. Authors of [2], [20] proposed algorithmic noise-tolerance (ANT) techniques for DSP systems where timing errors are permitted to occur and then corrected by a statistical error control block. In [21], [24], an approximate implementation of the main DSP block is used to estimate the output and provide reliability in presence of timing errors introduced by voltage over-scaling. In [2], errors are corrected by using composite error probability mass function to compute the a-posteriori probability (APP) ratio for each output bit. In this paper, we instead investigate an approach for building compensation logic for approximate computing systems and applications using supervised learning algorithms (i.e. inferring a compensation function based on the set of circuit behavior training data acquired during testing, discriminative error compensation (DEC)). We do not need to generate bit-wise results for each output which provides the approach more flexibility. As opposed to using statistical or Bayesian methods, accurate low-complexity models can be developed directly without having to model error, input conditional, and joint probability distributions using discriminative models. For this reason, discriminative learning may scale better for many problems and produce more accurate results than a statistical (i.e. generative) learning approach. In Section V, we compare the performance of the discriminative against generative models (e.g. Bayesian learned models).
IV. METHODOLOGY

To evaluate the performance of using supervised learning for building modules that compensate for the impact of permanent faults, we consider a set of different benchmark circuits shown in Table I. Our evaluation focuses on permanent faults. Spot and bridging faults due to process variations, lithography, and silicon impurities are the most well-known faults causing permanent faults in circuits [23]. We have injected random permanent faults spread across the whole circuit depth to observe and record the training data set using the Modelsim simulation tool [9]. Each independent permanent fault is modeled as a stuck-at-1 or stuck-at-0 fault during the circuit simulation.

Next, we built the compensation routines using learning algorithms (regression models, decision trees, Bayesian models etc) using the machine learning tool, WEKA [10]. Based on the parameters learned from these routines we then built and measured the area/performance overheads of the corresponding compensation modules. For each of circuits described in Table I, after observing 30 random faults for each circuit, the mean accuracy converged with <10,000 input patterns.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>ISCAS-85 benchmark circuit for 27-channel interrupt controller with 36 input and 7 outputs.</td>
</tr>
<tr>
<td>s832</td>
<td>ISCAS-89 benchmark circuit for programmable logic device (PLD) with 18 input and 19 outputs.</td>
</tr>
<tr>
<td>c6288</td>
<td>ISCAS-85 benchmark circuit 16-bit Multiplier circuit consists of 32 inputs and 32 outputs, 240 full and half adders.</td>
</tr>
<tr>
<td>Dot Product</td>
<td>16-bit Dot Product generator which takes 10 sets two 16-bit numbers total 320 inputs, returns 35-bit single output;1 c6288 multiplier and 1 Ripple carry adder used sequentially.</td>
</tr>
<tr>
<td>c7552</td>
<td>ISCAS-85 benchmark circuit 34-bit adder and magnitude comparator with input parity checking, 207 inputs and 108 outputs consisting 3512 gates.</td>
</tr>
<tr>
<td>FIR Filter</td>
<td>31-Tap Low Pass FIR filter with 8 inputs and 12 output, 30 adders and 31 multipliers.</td>
</tr>
</tbody>
</table>

The accuracy of the models are examined using the following error metrics:

**Correctly Classified Instances:** The percentage of test instances that were correctly classified.

\[
\text{Mean Absolute Error (MAE)} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|
\]

\[
\text{Relative Absolute Error (RAE)} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{\sum_{i=1}^{n} |y_i|}
\]

where \(y_i\) and \(\hat{y}_i\) are the compensated output and the ideal output respectively for a given input.

V. RESULTS

This section evaluates the accuracy and area/performance costs of compensation modules built using supervised learning.

We compare the accuracy (see Section IV for metrics) of the original faulty circuit outputs against the compensated outputs. For each benchmark circuit, there is a bar representing the accuracy of the 1) original faulty output, 2) compensation based solely on the input, 3) compensation based on the input and faulty output, and 4) the best compensation depending on the circuit and fault location. We assume that the compensation block is error-free.

The results for the Discriminative Error Compensation (DEC) routines are considered in two parts: Decision/Classification models and Regression Models.

A. Decision/Classification Models

Most prior work conducted for error compensation are highly application dependent and use probabilistic especially Bayesian models. Our approach is particularly well suited in the context of permanent fault models, as the activation of the fault is solely dependent on random data, and not the previous circuit state and environmental conditions. The DEC classification approach is useful for circuits that require exact outputs and produce a relatively small set of outputs.

1) Algorithm and Sample-Size Selection: We now show how discriminative error compensation (DEC) can be used in the context of these problems. We compare two discriminative machine learning algorithms (J48 and RepTree) with two generative Bayesian model algorithms (Naive Bayes and BayesNet).

From Fig: 5, we first observe that with increasing number of samples, the number of correctly classified instances also increases. Moreover, after only a several thousand samples, the accuracy saturates for each of the tested circuits. Finally, when comparing the discriminative models (J48 and RepTree) with the generative/statistical models (NaiveBayes and BayesNet), we confirm that the discriminative models tend to show higher accuracy than Bayesian models as the size/complexity of the circuit and its output increases (as discussed in Section III).

c432, s832 and FIR Filter have 632, 1442 and 167476 transistor respectively. Discriminative learning algorithms tend to scale better for these problems and produce more accurate results than the probabilistic (i.e. generative) learning approaches. Naive Bayesian classifier because of its independence assumptions between predictors, shows the lowest accuracy among all the algorithms. NaiveBayes is a high bias and low variance classifier which in many cases is considered a good option as it does not overfit. As the number of training sample increased decision trees start to provide more accuracy. The main drawback for decision trees can be overfitting which causes a larger tree. In the case of the these compensation routines, overfitting is less of a disadvantage as we want more accuracy and with larger circuits the area overhead is relatively very small.

The accuracy graph saturates at 10000, 9000 and 5000 number of training samples for c432, s832 and FIR Filter respectively.

2) Accuracy: We evaluate c432, a channel interrupt controller, using DEC where the supervised learning technique is
tuned to predict the exact output. From Fig. 6, we observe that DEC allows for an increase of 20% accuracy predicting the exact output nearly 92% accuracy when compensate using faulty output.

For s832, the accuracy increases to average of 98% when used compensation with faulty output from 71% without using any compensation. For FIR Filter, both compensation models with and without faulty models show 100% accuracy.

B. Regression Models

The circuit c432 has a total of 121 different outputs, whereas in the case of a adder or multiplier the total number of possible outputs is proportional to the number of input patterns and significantly larger than for c432 (e.g. for the 32-bit multiplier, there may be a total of $2^{64}$ possible output patterns).

Regression based algorithms (Linear regression, RepTree etc) can be used for circuits which are used in scenarios where approximate results can be used. In such cases when a fault is present, it is more important to produce results closer to actual result so that the relative absolute error is as low as possible, instead of exactly correcting individual outputs. For this reason, the compensation may add a small amount of error to some outputs which were otherwise correct, but the overall average error will be significantly smaller (Figures 7 and 8).

1) Accuracy: The accuracy of the different circuits (x-axis) against the relative absolute error and the mean absolute error are shown in Figure 7 and Figure 8 respectively. We observe that for all three circuits, the accuracy can be improved with the use of compensation logic and the error was reduced by up to 2-3 orders of magnitude. The best approach (e.g. compensation with or without fault output) depended on the circuit and fault location. For c6288, c7552 and Dot Product, the compensation model built using the faulty output provided the best accuracy.

Figures 7 and 8 show the standard deviation and max/min spread for the output error of each circuit. As expected, the compensation module reduced the mean output error but also significantly reduce the spread and variance of the error. The amount of reduction depended significantly on the complexity and structure of the circuit however. For c6288, the mean error was reduced by 27%, while the variance was reduced by over 50%. For c7552, the mean error was reduced by a few orders of magnitude. A 50% and 40% improvement in output error for a multiplier, and dot product respectively may be a sufficient reduction in the output error depending on the requirements of the particular approximate application being used.

C. Performance and Area Overhead

In this section, we consider the area overhead of using supervised learning based error compensation modules. We consider two potential compensation logic designs, one optimized for area and another optimized for performance (Note that this is the on-line performance overhead). For the compensation logic, a 10 Transistor 1-bit full adder [17], a magnitude comparator [7], a multiplexer and a EPROM are used. For the area optimized compensation, for linear
regression, only one adder and one multiplexer are needed. For RepTree, only one comparator and one multiplexer are needed. For the performance optimized design, full parallel adders, comparators and Multiplexers are used in order to reduce the overall compensation delay.

From Figure 10, we observe that, though for the smaller circuits (c432), the area overhead is 2-3 order magnitude but for the more complex circuits (c6882, c7552 and FIR Filter) area overhead is only 3% to 16%. For Dot Product, we observe 12% area overhead for area optimization.

Much like the area overhead, we observe, the latency overhead can be amortized and decreases as the size of the circuit increases. For the performance optimized implementation of the more complex benchmarks (FIR filter, c6288 and c7552) the overall performance delay was $< 3\%$. For performance optimization configuration, with a little extra area overhead, we can compensate with much less latency overhead.

D. Sensitivity Study

For a particular circuit, varying the algorithm, number of samples or size of the compensation logic will effect the compensation accuracy. Each of these parameters allows for trade-offs between the model accuracy and area/performance overhead. As discussed in section II-B, the supervised learning algorithm which provides the best accuracy also depends on the characteristics of the given circuit. Figure 11, shows the
Fig. 8: Box Plot for average of Mean Absolute Error for original output, compensation based on the input, compensation based on the input and faulty output, and the best compensation depending on fault.

TABLE II: Transistor needed for compensation logic

<table>
<thead>
<tr>
<th>Area optimized</th>
<th>Performance optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of Transistors in Actual circuit</td>
<td>Compensation w/o faulty output</td>
</tr>
<tr>
<td>c432</td>
<td>632</td>
</tr>
<tr>
<td>s832</td>
<td>1,442</td>
</tr>
<tr>
<td>c6288</td>
<td>10,112</td>
</tr>
<tr>
<td>Dot Product</td>
<td>11,268</td>
</tr>
<tr>
<td>c7552</td>
<td>15,400</td>
</tr>
<tr>
<td>FIR Filter</td>
<td>167,476</td>
</tr>
</tbody>
</table>

Fig. 9: Area Overhead (%) for Area and Performance optimization. Red Column: Compensation w/o Faulty Output, Blue Column: Compensation w Faulty Output.
accuracy for benchmark circuit s832 using various potential learning models. We observe that for 10000 training samples, J48 decision tree provides the best accuracy where the algorithm can correctly classify 99.32% instances. For c432 and FIR Filter circuits also J48 provided best accuracy correctly classifying 93% and 100% instances respectively.

The choice of the algorithm, number of training samples for learning and size of compensation blocks mainly depends on the accuracy needed for the approximation circuit the compensation is applied to.

VI. CONCLUSION AND FUTURE WORK

With process scaling and the adoption of post-CMOS technologies permanent faults are becoming a fundamental problem. Today, circuits containing defects are either partially disabled (sacrificing performance) or simply discarded (reducing yield). We propose a general approach using supervised learning to reduce the impact of permanent faults in a circuit. We also describe a low cost design for using the supervised learning based compensation model. For complex accelerator and application-specific logic, the approach is shown to reduce the mean output error by up to 2-3 orders of magnitude while incurring <10% area overhead and <3% performance overhead. Our future work will address our approach in the context of more complex ASICs and general purpose computing logic.

REFERENCES


