Resilience Evaluation for Approximating SystemC Designs Using Machine Learning Techniques

Mehran Goli[1] Jannis Stoppe[2,3] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[3]DLR Institute for the Protection of Maritime Infrastructures, 27572 Bremerhaven, Germany
{mehran jstoppe drechsle}@informatik.uni-bremen.de

***Abstract*—As digital circuits have become more complicated than ever, abstract description languages such as SystemC have been introduced, allowing designers to work on more abstract levels during the design process. Design metrics such as performance and energy consumption are a central concern for designers at all levels of abstraction. Approximate computing is a promising way to optimize these criteria, sacrificing accuracy. Defining which parts of a design can be approximated (and to what degree) is a crucial and non-trivial design decision, which is usually connected to a larger programming effort, especially when exploring the design space manually.**

In this paper, we propose an automated approach based on machine learning techniques in order to detect the resilience of a given SystemC design's modules. This is used to identify components of the design that can be approximated. The effectiveness of the proposed method is evaluated using several SystemC benchmarks from various domains.

I. Introduction

The ever-increasing complexity of *System-on-Chip*s (SoCs) and tight time-to-market constraints shifts the designers' focus to abstract levels in order to rapidly implement working results. System design at the *Electronic System Level* [1] (ESL) is one way to work on these more abstract layers, allowing designers to implement executable mixed hardware-software systems in high-level programming languages. The system-level language SystemC [2] has become the de-facto standard [3] to specify *Virtual Prototype* (VP) models at the ESL, enabling designers to rapidly prototype and consecutively simulate complex systems. Performance, cost (i.e. size) and energy consumption of the systems have always been and remain central concerns during the development of electronic systems [4]. As the cost of applying significant structural changes to a given design increases with the stage of development, these optimizations should be incorporated into the model as early as possible – i.e. at the ESL, if possible.

Approximate computing is one promising solution [5], [6] to improve the performance or reduce the required area and energy consumption of embedded systems – at the cost of output accuracy. Based on the idea that designs often include some specific parts that contribute less to the quality of output than others, modifying these parts of the model with respect to the quality of its output can lead to enhanced design metrics such as energy consumption and performance. The output quality is measured using its premise boundary that is specified as a Quality of Service (QoS) range. A part of the design is considered as resilient or approximable if its modification has a low impact on the output's QoS.

Specifying which parts of a given ESL design are approximable is the critical starting point of approximate computing techniques as the incorrect detection of critical parts as approximation candidates can be expensive in terms of the output quality of the system. For a given ESL design answering the following questions can help the designer to make better decision on design optimization.

1) Which modules of the design may be approximated?
2) What is the degree of approximation (i.e. the error rate that a module can accept w.r.t the reference QoS of the design) for an approximable module?
3) How much improvement on design metrics (e.g. performance) can be achieved by approximating a module?
4) How many modules of the design can be approximated at the same time?

According to [5], a solution that locates the error-resilience portion of a design (which identifies the most promising approximation candidates) should be as automated as possible. Existing solution are mostly based on developing new programming languages to provide designers with frameworks to manually specify approximable data [7] or source code annotations to determine whether or not a part of the code is resilient [8], [9]. Moreover, the methods focus on either algorithmic level [10], [4] or lower levels of abstraction, that is the *Register Transfer Level* (RTL) [11] and below [12], [13].

In this paper, a statistical analysis is proposed on the simulation behavior of a given ESL design in order to detect which parts of the model are candidates for approximation. The main contributions of the paper are

- proposing an automated resilience portion detection method using machine learning techniques to identify approximable components of a given ESL design,
- translating the simulation behavior of an ESL design into sets of observation to be used as the input of learning algorithms,
- performing a significance analysis on the potentially approximable components by ranking their inputs according to their contribution to the quality of their outputs, and
- applying the proposed method to several ESL benchmarks in various domains to evaluate its effectiveness.

II. Related Work

Approximate computing has been applied to the design of hardware domain modeled at RTL and below such as arithmetic units [14], data paths [15] or voltage scaling [16]. The leverage of machine learning algorithms to improve the energy consumption has been introduced in [17], [18] where the computational parts of the design are replaced with hardware neural networks. However, their application are limited to some particular designs. Furthermore, the impact of applying such techniques in order to achieve the desired improvement is significant at higher levels of abstraction.

978-1-5386-7557-1/18/$31.00 ©2018 IEEE

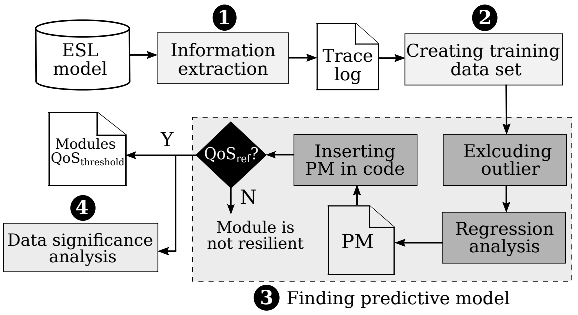

Fig. 1: The architecture of proposed method.

Several approaches [12], [19], [20] have been introduced related to classifying program code (or its underlying structures) as either critical or resilient. They mostly analyze the sensitivity (the degree of an output's volatility in relation to distortions in each variable in a program) based on statistical techniques [10], [4] in order to specify the approximable variables at the algorithmic level.

In [10] a statistics-based method is presented to automatically locate approximable structures in a program. First the variables of the program and the range of values are extracted. Then, during the "error injecting phase" the value of each variable is perturbed to measure the effect of the modification on the output. If the new output lies within the specified QoS threshold, the variable is marked as approximable. In [4] the same technique is used to analyze the sensitivity of each variable in a program. The difference is that it does not perform a range analysis of variables as it take advantages of memory bit flips for its error injection phase. Moreover, the number of program executions in the presence of perturbations to perform sensitivity analysis is selected using probabilistic computations while in [10] it is set manually.

However, these methods cannot be applied to hardware systems easily as they disregard any timing and architectural information. Moreover, the performance of all methods that identify local data as the resilient portion of a program is related to the number of variables and the number of samples (which corresponds to the required program executions). Therefore, execution time increases with the amount of variables, leading to significant issues with larger programs that rely on local data.

As ESL systems are usually structured using modules that communicate via signals, analyzing each of these modules separately with regard to their inputs and outputs should reduce the overall complexity of the process significantly while at the same time providing an approximation solution early in the design process. However, no existing solution supports a sensitivity analysis on ESL designs in order to detect their approximable components.

III. Proposed Methodology

As illustrated in Fig. 1, the process of finding resilient parts of a given ESL design is divided into four phases:

1) data retrieval and storage of the extracted information,
2) training data setup (using the extracted data as the input of learning phase),
3) predictive model creation for each module of the design and
4) a data significance analysis on input signals of each module which was marked as approximable in the previous phase.

A. Data Retrieval and Mapping

In order to analyze the simulation behavior of a given SystemC model, the run-time information needs to be extracted. It includes both static (describing the model's structure) and dynamic information (describing its behavior). The former refers (among other information) to the modules' names and the corresponding member functions and attributes (including input and output variables) while the latter refers to the values of each module's variables during simulation time.

The data extraction process is performed by running the design in debug mode under control of the GNU debugger (GDB). The execution is controlled by programming GDB to automatically extract the run-time information of the design. From the execution trace, a log file is generated including the design's structure and simulation behavior. As the piece of information related to each variable of a module scatters in the log file, this information is extracted from it and specified by following definition.

Definition 1: each SystemC design $D = \{M_i : 1 \leq i \leq n_m\}$, where module M_i is a tuple (I, S_{in}, S_{out}) and n_m is the number of modules. I is the instance name of the module. $S_{in} = \{s_{in_i} | s_{in_i} = (in_i, V_{in_i}) : 1 \leq i \leq n_{in};\ V_{in_i} = v_{in_j} | 1 \leq j \leq k\}$ is a set of input variables, where n_{in} is the number of inputs and V_{in_i} is a set of values that is assigned to input variable in_i during execution. $S_{out} = \{s_{out_i} | s_{out_i} = (out_i, V_{out_i}) : 1 \leq i \leq n_{out};\ V_{out_i} = v_{out_j} | 1 \leq j \leq k\}$ is a set of output variables, where n_{out} is the number of outputs and V_{out_i} is a set of values that is assigned to output variable out_i during execution. k is a positive integer indicating the size of V_{in_i} and V_{out_i}.

For a given ESL design we consider two assumptions. First, the execution trace for each output out of a module can be defined as a function $f : \mathbb{R}^n \to \mathbb{R}$ that calculates a scalar output $out \in \mathbb{R}$ by evaluating $out = f(S_{in})$. This assumption allows it to solve the module's behavior approximation with machine learning algorithms that require such a dependency between the input and output of a system.

Second, the test vectors for each design should provide a high code (i.e. statement) coverage for all modules of the design. We believe this assumption is fair as it is common practice to keep a design's code coverage monitored and high to ensure that it works as intended. Thus a design mostly contains a test bench with a high code coverage provided by designers either manually or by employing an automated test generator. Note that a higher coverage of the input signals generally results in an improved estimation of the modules' behavior.

B. Training Data Set Creation

The core idea is to use the data that is retrieved from a simulation run as a training set for a machine learning algorithm to build a new (estimated) model of each module present in the design. In order to do this, the first step is to filter and transfer the required information (from Section III-A) of each module into a valid training data set. The training data set includes two elements which are a) one or more inputs (the *predictors*) and b) the corresponding output (the *response*). Each pair of predictors and response is considered one *observation*. The set of input variables of each module S_{in} is defined as a set of predictors while each output of the module (out_i, V_{out_i}) from the output variables set S_{out} is specified as a response. Formally, the definition w.r.t the definition 1 is:

Definition2: for each module M a set of training data set

$$T = \{t_i \mid t_i = \{S_{in}, (out_i, V_{out_i})\} \ ; \ 1 \leq i \leq n_{out}\}$$

where each training data set t is an $(n_{in} + 1)$-dimensional vector and the size of T is equal to the number of output variables n_{out}.

The learning algorithm accuracy is related to two main factors: the dimension of the training data set and the features of each predictor (i.e. the inputs) within the training data set. While more information is generally beneficial for the correctness of machine learning results, irrelevant information may also impact the learning process negatively. This is especially true for controlling signals if the machine learning algorithm maps the data to continuous functions. E.g. boolean inputs or clocks (which can be distinguished from numbers in high level system designs using their type information) are often used as controlling signals (such as reset, request, grant, valid, etc.). They often change the semantics behind the output, messing up learning methods such as linear regression that rely on a continuous mapping from inputs to outputs. One example would be a module with three inputs $a, b \in \mathbb{R}$ and $c \in \{0, 1\}$ and the result of the module r being

$$r = \begin{cases} a + b & \text{if } c = 1 \\ a \cdot b & \text{otherwise} \end{cases} \tag{1}$$

For these cases, the data needs to be split with regard to c before being processed in order to enable the learning algorithm to quickly determine the correct relation. Therefore, Boolean variables are eliminated from the training data set prior to the learning phase.

For a training data set with more than one predictor (especially for predictors with very little variance), classifying the training data set (i.e. splitting it into subsets) based on this predictor (instead of using it as an input for a regression analysis) may return more precise results. This predictor is denoted as a classifier where the number of features is equal to the number of unique values within its data set. A predictor's value is considered as a feature if the amount of its appearances in the predictor value set is at least equal to L. L is defined as the minimum amount of observations that a subset of the training data set must contain after classification to keep the accuracy of the learning model. Finding the optimal value of L is challenging as it directly affects how the training data set is partitioned into subsets and thus contributes to the overall accuracy of the learning model.

According to [21], this optimal value is usually defined by some clustering algorithms such as *k-means clustering* (clustering observations set in multiple subsets such that observations within the same subset are as similar as possible and k is the number of subsets). To determine the optimal value of L, an adaptation of the *elbow method* for *k-means clustering* is used. The proposed method defines different values for L from a high value (e.g. 50% of the training data set size) to a low value (e.g. 5% of the training data set size) to create different subsets of the training data set. In the next step, a particular value of L is selected if decreasing more than this value has no or negligible improvement on the learning model's accuracy. The original training data set is thus divided into several subsets (denoted as training subsets), where each subset is defined by a unique feature of the classifier predictor. Note that in case that designers have some pre-knowledge about the design the parameter L can be also set manually allowing them to control the behavior of the algorithm.

Algorithm 1: Classification

```
Data: Training data set $T$ and constant $L \leftarrow [50\%size(T)\ to\ 5\%size(T)]$
Result: classifying $T$ based on classifier predictor $P$
foreach predictor $P$ in $T$ do
    $S_P \leftarrow \emptyset$;
    foreach value $l$ in $L$ do
        $S_l \leftarrow classify(P, T)$;
        $S_{temp} \leftarrow \emptyset$;
        foreach subset $S$ in $S_l$ do
            if $length(S) \leq l$ then
                merge($S_{temp}, S$);
                remove($S_l, S$);
        add($S_l, S_{temp}$);
        add($S_P, S_l$);
    add($S_t, S_P$);
```

Algorithm 1 shows the classification algorithm on the training data set T with the constant parameter L which is defined as the minimum number of observations for each member of S_t. All subsets that have less members than L are merged into a new set S_{temp}. Note that increasing the number of classifiers (by decreasing the size limitation L more that %5 of the training data set) can reduce the number of observations for each subset. This may decrease the accuracy of the prediction function for each subset of the training data set, as subsets with fewer members may provide unreliable information for the learning algorithm.

Algorithm 2: Finding Predictive Model

```
Data: Training data set $t$, constant $n_g$
Result: Predictive model $PM$ and $QoS_{th}$
Function FPM(t) is
    $L_{PM} \leftarrow \emptyset$;
    $L_{QoS} \leftarrow \emptyset$;
    $t_{outlier}$ = outlier($t$);
    foreach formula $f \in$ [linear, interactions, quadratic] do
        $PM_f \leftarrow Linear(t, f)$;
        $PM.add(PM_f) \leftarrow PM_f.predictorFnc()$;
        $L_{QoS}.add(PM_f) \leftarrow PM_f.QoS$
    if $(length(t) \leq n_g)$ then
        foreach Gaussian kernel $G_k \in$ [RQ, SqExp, M5, Exp] do
            $PM_{G_k} \leftarrow Gaussian(t, G_k)$;
            $PM.add(PM_{G_k}) \leftarrow PM_{G_k}.predictorFnc()$;
            $L_{QoS}.add(PM_{G_k}) \leftarrow PM_{G_k}.QoS$
    else
        $PM_{tree} \leftarrow tree(t)$;
        $PM.add(PM_{tree}) \leftarrow PM_{tree}.predictorFnc()$;
        $L_{QoS}.add(PM_{tree}) \leftarrow PM_{tree}.QoS$
    $QoS_{th} \leftarrow \min(L_{QoS})$;
    $PM \leftarrow L_{PM}(index(QoS_{th}))$;
    return $PM, QoS_{th}$;
```

C. Predictive Model Creation

A predictive model is created by determining relations between one or more predictors and the corresponding response in the training data set. The core question is whether or not a predictive model can be found to approximate the behavior of a module with respect to the QoS of the design. The problem for a given module is formulated as follows: for each member t_i of the training data set T (based on definition 2), a function

$f_i : S_{in} \rightarrow (out_i, V_{out_i})$ where $1 \leq i \leq n_{out}$ and the distribution of the predictors S_{in} and the function f are both unknown. The task is then to find a predictive model PM that describes the underlying data with $PM_i(S_{in}) \approx f_i$ for all observations in t_i with respect to the pre-defined QoS of the design.

Regression analysis [22] is one possible solution to extract a predictive model from a set of observations. The PM can be created with different regression methods depending on the observation's properties and the requirements of the resulting model. Since the distribution of predictors and response is unknown, two assumptions are possible.

- Either the relation follows a known function and only a finite set of parameters needs to be estimated, allowing the predictive function to be estimated using a parametric regression model such as linear regression,
- or the relation follows an arbitrary function that can be estimated using a non-parametric regression model such as a Gaussian Process (GP) or tree regression (decision tree).

In order to increase the chance of locating a proper model, the training data set is modified by excluding outliers from it. An outlier is an observation point in the training data set that is very different from other observations. The outlier is often calculated using the distance metric on the value of response variable [23]. An observation in the training data set is considered as an outlier if the value of its response variable is more than K interquartile ranges below the lower quartile q_l or above the upper quartile q_u of its response value set. Therefore, all values v_{out_i} out of range $[q_l - K(q_u - q_l)$, $q_u + K(q_u - q_l)]$ are outlier. We consider the default value $K = 1.5$, which can be considered a standard value [23]. The experiment (discussed in section IV) also shows that the aforementioned value is the most fitting one to calculate the percentage of outliers in the considered domain.

Algorithm 2 illustrates the method that creates the predictive model using the training data set t with constant parameter n_g (which is defined as the maximum number of observations for the GP analysis). It consists of two main steps, first applying multiple linear and later a GP or tree regression analysis. The linear regression analysis (Algorithm 2 – Lines 5 to 8) is performed using three different underlying functions: *linear*, *interactive* (which assumes that inputs may not be completely independent) and *quadratic* (which consists of linear, interactive and quadratic terms in the predictors). The underlying function of *quadratic* including all aforementioned terms is $f(S_{in}) = \alpha + \Sigma_{i=1}^{n_{in}}\beta_i s_{in_i} + \Sigma_{i=1,j=i+1}^{n_{in}-1}\beta_{ij} s_{in_i} s_{in_j} + \Sigma_{i=1}^{n_{in}}\beta'_i s^2_{in_i}$ where α represents the intercept and β, β' coefficients. n_{in} is the number of predictors in t. The goal of learning process is to estimate the intercept and coefficients of each term in order to minimize the root-mean-square error (RMSE) between the predicted and true model.

In the second phase, the GP (Algorithm 2 – Lines 9 to 13) or tree regression analysis (Lines 14 to 17) is performed in case that the predictors and response are inferred from an unknown distribution. The GP regression is often able to estimate such a relationship even with low number of observations. It creates a predictive model of response based on the idea that for an arbitrary point c, the observation o with a similar value X of $(v_{in1_i}, v_{in2_i}, ..., v_{inn_i})$ in the predictors space S_{in} have a high impact on the predictive model $f(c)$. The similarity is measured using the co-variance distance metric in predictor space which is defined as $K(X, c)$. The co-variance function can be calculated using different underlying functions known as *Rational Quadratic* (RQ), *Squared Exponential* (SqExp), *Matern5* and *Exponential* (Exp) [22]. As the number of observations increases, the GP analysis becomes slower due to its computational complexity. Therefore, the method is only applied to the training data sets that contain less than n_g observations. In this paper we consider $n_g = 100$.

For a training data set that has a large number of observations, the tree regression analysis is used instead of the GP, as it comes with a worst case linear complexity. It follows the decisions on predictors values in the tree from the root node down to a leaf node. The leaf nodes present the response values.

The result of the FPM function is the predictive model that has the best QoS over all regression analyses. In case of identical QoS results, the simplest model is selected to replace the computational part of the design. As illustrated in Fig. 1, (during phase 3) the extracted PM is automatically translated to a SystemC source code fragment that is used to replace the computational part of the module. Linear models are translated to the corresponding equation. Decision trees are chained $if - else\ if$ statements. In case of GP models, for each value of predictors the estimated response is replaced in $if - else\ if$ statements. It means that the GP model is not transferred itself to the design but its responses for the training data are embedded instead. This allows designers to anticipate the impact of an approximation and run repeated runs with the given data, but it cannot be applied to inputs it was not trained on. If the PM for each module satisfies the reference QoS (QoS_{ref}) of the design, the module is marked as approximable and its RMSE is defined as the threshold QoS ($QoS_{threshold}$). It means that the computational part of the module can be approximated without affecting the QoS_{ref} if the approximated model satisfies its $QoS_{threshold}$. Like prior works [10], [4] we assume that the error estimation module as well as the QoS_{ref} are provided by user.

Algorithm 3: Significance Analysis

```
Data: Training data set $t$ and predicted model $PM$
Result: Ranking the significance of each predictor in $t$
foreach predictor $P$ in $t$ do
    $t_l \leftarrow$ lower_bound $(P, t)$;
    $t_u \leftarrow$ upper_bound $(P, t)$;
    $t_l.response \leftarrow PM\ (t_l)$;
    $t_u.response \leftarrow PM\ (t_u)$;
    $RMSE_l \leftarrow$ rmse $(t_l.response, t.response)$;
    $RMSE_u \leftarrow$ rmse $(t_u.response, t.response)$;
    $RMSE_{mean} \leftarrow$ mean $(RMSE_l, RMSE_u)$;
    $Significance\_List_t.add\ (P, RMSE_{mean})$;
sort($Significance\_List_t$);
```

D. Data Significance Analysis

After finding a predictive model that approximates the behavior of a module, it is important to know which of the module's input signals have the largest impact on its output signals. This can be defined as the significance of each input signal. Therefore, the goal of this last phase is to specify the significance of predictors in a module's training data set.

For predicted models that are based on linear regression analysis, the coefficients of predictors in the underlying formula of the model are used to rank their significance. A predictor with greater absolute coefficient values is interpreted

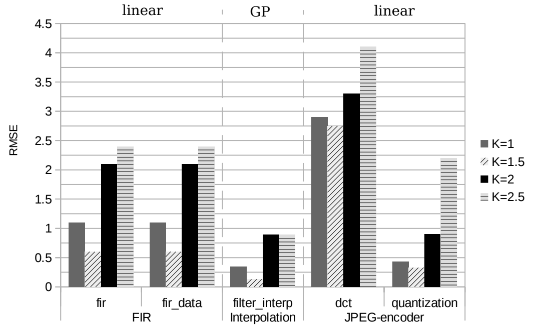

Fig. 2: The impact of parameter K (defining the percentage of outliers for each design) on the predictive model's accuracy.

as having a more significant impact on the response. This way, the contribution of each input signal on each of its module's outputs is calculated.

If the predicted model is instead based on GP or decision tree regression, the contribution of each predictor is determined by measuring the RMSE between the response of the predicted model with and without injecting an error to the predictor's values while other predictors stay untouched. Greater RMSEs indicate a more significant impact on the response. The error injection step is performed using an adaptation of the *bitflip error* model (which is used by several prior works [12], [9], [10] for sensitivity analysis). The error for each predictor is calculated based on 5% of the lower (Algorithm 3 – Lines 2, 4 and 6) and upper (Lines 3, 5 and 7) bound of its original values. This computation is shown in Algorithm 3.

IV. Experimental Evaluation

The method is applied to several standard ESL benchmarks provided by [24] and OSCI [25]. For each benchmark, the simulation behavior is extracted by AIBA [26] from its runtime behavior and stored in a log file. The extracted information is then automatically translated to a structured model. This information is parsed to automatically generate a set of Matlab code fragments (one for each module of the model) that include three phases; the creation of the training data set, the creation of predictive models and the significance analysis on the training data set. Both programs (for parsing the AIBA output and generating Matlab code) were written in python. The experimental evaluation is described in two parts. First, two case studies from image processing domain are illustrated in detail in Section IV-A. Second, we give a brief discussion on the obtained experimental results to evaluate the quality of the proposed method in Section IV-B. To quantify the QoS loss of each design due to approximation, we consider signal-to-noise ratio (SNR) for the *JPEG-encoder* and *Sobel* designs and RMSE for all other designs. All analyses have been performed on a PC equipped with 8 GB RAM and an Intel core i7-2760QM CPU running at 2.4 GHz.

A. Case Studies

The experimental results for different types of ESL benchmarks are shown in Table I. The first two columns list the names and types of the designs, respectively. Column *LoC* and *#M* present the lines of code and the number of modules in each design, respectively. Column *#AM* shows the number of modules that have been detected as candidates to be approximated by the proposed method. Column *AMN* and *OutN* list the names of the approximable modules and their output signals for each design, respectively. Column *PM* presents which type of regression analysis was used as learner algorithm to evaluate the sensitivity of modules. Column *#Obsv* shows the number of observations used in the regression analysis to estimate the module's behavior. The *Outl* column shows the percentage of outliers in the observation data that was excluded from the training data set. These values are obtained based on the formula in Section III-C. To determine the optimal percentage of outliers for each design, we studied the effect of different values of K on the accuracy of the estimated model. As illustrated in Fig. 2, for each design $K = 1.5$ provides the most accurate estimate of its true behavior. The *Clf* column presents the names of the input variables that are interpreted as classifiers in the training data set. Column *RMSE* illustrates the root-mean-square error between the estimated model and the original behavior of the module. The *PG* column shows the simulation performance gain for the entire design (in percent) that is obtained by replacing the original computational part of each module with the predicted model. The execution duration of the proposed method is reported in columns *Runtime* including the execution time for each phase (*p1* to *p4*) and the total execution time *TE* of the method. The time reported in *p3* covers all steps in phase 3 including outlier exclusion, regression analysis and evaluating the replacement of the original code of design by the predicted model. Column *CET* shows the total time of each design's compilation and execution without any instrumentation.

In order to evaluate the quality of the proposed method, we consider the *JPEG-encoder* and *Sobel* designs [24] which are extensively used in image processing domain as compression and edge detection algorithms, respectively in detail. The degree of module resilience was evaluated for each design by specifying two levels of approximation: *Mild* and *Aggressive*, each realized by adjusting the QoS of each design.

1) JPEG-encoder: As shown in Table I, two out of five modules of the *JPEG-encoder* design are candidates for approximation where the behavior of both modules were estimated by linear regression model. Due to the RMSE, the *quantization* module with less RMSE has lower resilience for approximation than the *dct* module. The performance gain in case of approximating *quantization* is small in comparison to the *dct*. This is because the complexity of the computational part of the *quantization* is close to the predicted model – which is linear. In contrast, approximating the *dct* module provides a noticeable performance gain. Take e.g. the *dct* module of the design. Fig. 3a shows the original image of the *JPEG-encoder* design without any modification. Fig. 3b illustrates the *Mild* approximation of the model that the computational part of the *dct* module was replaced with the linear function

$$PM(S_{in}) = 0.1096 * S_{in} - 14.11 \qquad (2)$$

where 3% of the observations were excluded from its training data set as outliers. The predicted model is presented in Fig. 4 including all observations with no outliers being excluded. For the *Mild* approximation, the dct_out values out of range $[-100, 100]$ were considered outliers. Note that for the excluded observations, we keep the original computational part of the module. In case of the *Aggressive* approximation, the same PM in equation 2 was used, with all observations including the outliers being estimated by the predicted model. Fig. 3c shows the corresponding output image.

TABLE I: Experimental Results of all Case Studies

ESL model	Description	LoC	#M	#AM	AMN	OutN	PM	#Obsv	Outl%	Clf	RMSE	PG%	Run-time					CET(s)
													P1(m)	P2(s)	P3(s)	P4(s)	TE(m:s)	
3-stages pipeline[1]	Floating point computation	511	5	3	stage1	sum	linear	50	-	-	0.01	-1.2	0.3	1.1	5.2	0.1	0:47.6	3.5
						diff	linear				0.01	-1.2			5.2	0.1		
					stage2	prod	linear	50			0.9	-0.9			5.5	0.1		
						quot	linear				1.1	-0.8			5.6	0.1		
					stage3	powr	linear	50			3.14	-2			5.6	0.1		
FIR[1]	Filter	834	4	2	fir	result	GP	100	2	-	0.6	3	0.3	1.3	6.3	0	0:34.6	4.2
					fir_data	result	GP	100			0.6	2.7			6.1	0.6		
Interpolation[2]	4-Stages Filter	587	1	1	filter_interp	odata	GP	100	1	-	0.13	8.6	2	0.7	6.9	0.5	2:08.1	5.8
RISC-CPU[1]	Processor	1960	10	3	exec	dout	linear	250	-	opcode	(1-3)*–> 0.01	-0.2**	23	3.4	40.8†	0.1**	27:24.9	12.5
											(5-6)–> 1.1	0.4			26.2	0.1		
											(7-14)–> 0.1	-0.4			92.4	0.1		
					floating	fdout	linear	150		opcode	(1-3)–> 0.01	-0.2			25.7	0.1		
											(4)–> 0.61	0.2			25.9	0.1		
					mmxu	mmxdout	linear	150		opcode	(4-7)–> 0.8	0.3			48.4	0.1		
Sobel[2]	Image processing (edge detection)	713	1	1	sobel	output_row	tree	131072	-	-	3.15	23	25	5.4	8.8	0.3	25:14.5	3.3
							linear				31.65	16			7.2	0.1	25:12.7	
JPEG-encoder[2]	Image processing (compression)	1422	5	2	quantization	quantization_out	linear	131072	1.5	-	0.33	4	35	6.3	19.2	0	35:57.1	17.6
					dct	dct_out	linear	131072	3		2.75	21			16.6	0		
									-		2.75	33			14.3	0		

[1] OSCI and [2] [24] **LoC:** Lines of Code **#M:** number of Modules **#AM:** number of Approximable Modules **AMN:** Approximable Module Name **OutN:** Output Name **#Obsv:** number of Observations **Clf:** Classifier **P1** to **P4:** Phase 1 to Phase 4 **TE:** Total Execution **CET:** Compilation and Execution Time *opcode range 1 to 3 **The average value (for PG and P4) over the opcode range †Total run-time of P3 for all opcode in the range

As demonstrated in Table I, the performance gain of the entire design in the *Aggressive* approximation is higher than the *Mild* approximation as the whole computational part of the module is replaced with a simple linear model. However, the quality of the output image of the *Aggressive* model is lower than the *Mild* model. The performance gain is 33%. The SNR for the former model is $18.43\ dB$, for the latter $25.55\ dB$. As the module only has one input variable, no sensitivity analysis is performed.

2) Sobel: As illustrated in Table I, the *sobel* module was approximated using two different regression models – tree and linear. The best approximation was obtained using the tree regression model. This model is considered the *Mild* approximation as it keeps the quality of output image up in comparison to its original result. The result of the tree regression analysis is presented in Fig. 5b. The SNR (in comparison to the original output image in Fig. 5a) is 14.8 dB. Moreover, it improves the design's performance by 23%.

The *Aggressive* approximation provides a lower quality of the output image but requires a higher RMSE threshold for the module. Fig. 5c illustrates the output image when it is processed by a linear regression including quadratic and interaction terms as follows

$$PM(S_{in}) = 196.45 - 0.63765 * s_{in_1} + 1.582 * s_{in_2} - 0.59222 * s_{in_3} + 0.0054967 * s_{in_1} * s_{in_2} + 0.07435 * s_{in_1} * s_{in_3} + 0.040788 * s_{in_2} * s_{in_3} - 0.036598 * s_{in_1}^2 - 0.030124 * s_{in_2}^2 - 0.054513 * s_{in_3}^2 \quad (3)$$

where $PM(S_{in}) \approx output\_row$ and S_{in} representing the set of input signals. The SNR – 10.3 dB in this case – shows that 30% of the output quality is lost in comparison to the *Mild* approximation.

As the *sobel* module includes three inputs, we performed a sensitivity analysis on the predicted model to measure their respective impacts. Since the predicted model is estimated using linear regression, the absolute value of variables' coefficient in the underlying formula is used to rank the inputs. Due to its larger coefficient, the second input variable of the design appears to be more important for determining the output's value.

B. Integration and Discussion

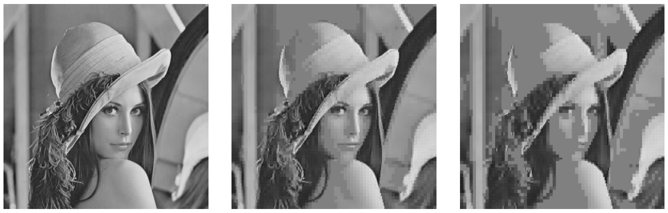
(a) Original (b) Mild Approx. (c) Agressive Approx.

Fig. 3: Image (a) is the original output of *JPEG-encoder* design. (b) and (c) are the output images of approximating the *dct* module using a linear regression model with and without excluding outliers, respectively.

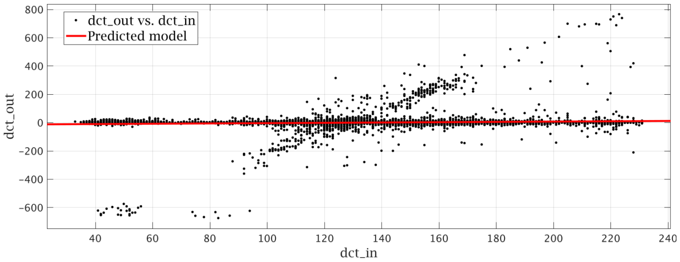

Fig. 4: Predicted model of *dct* module using linear regression model of equation 2.

To demonstrate the generality and scalability of the proposed approach, various further benchmarks are provided in the Table I. Unlike the traditional sensitivity analysis methods (at the algorithmic level) that only specify the resilience portion of a program, the proposed method provides the designer

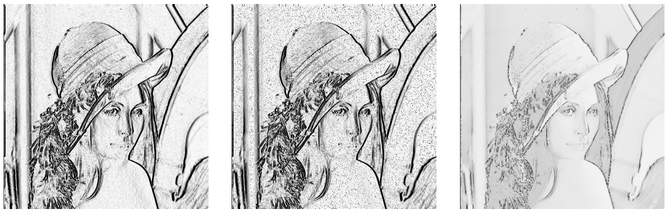
(a) Original (b) Mild Approx. (c) Aggressive Approx.

Fig. 5: Image (a) is the original output of *Sobel* design. (b) and (c) are the output images where the *sobel* module is estimated using tree and linear regression model, respectively.

with more information by answering the questions listed in Section I. Regarding the first question, the proposed method can identify the number of modules that can be approximated. In case of question 2, the RMSE threshold reported in Table I provides an upper bound error rate for each output of the design's modules that are marked as approximable to meet the QoS of the design. The performance gain presented in Table I answers the third question of this paper. It provides the designer with an early estimation on the potential performance improvement that can be achieved by approximating modules of a design. The negative performance gain reported in Table I (e.g. *3-stage pipeline* design) shows the complexity of the estimated model may be higher than the original computational part of the approximable module. In these cases, modules can be approximated but the predicted model is not the best choice. Note that the goal of this research is to evaluate the resilience of modules of a given design and not to find the best approximation alternative for the module. However in case that the predictive model satisfy the reference QoS of the design as well as potential other design criteria, the model can be used as an approximation alternative for the module (e.g. *sobel* module in the Mild approximation). With decreased simulation run-time (i.e. gaining performance) for a high level design, designers still get the valuable results how the design behaves with an approximated module – allowing them to focus on this relevant design to inspect for approximation at lower levels.

Experiments concerning the fourth question are not yet working automatically, currently requiring the designer to manually find the best combination of approximable modules for a design to improve the design metrics. It can be considered an optimization problem that is, however, left for future work.

The performance of the proposed method depends on two phases: the time that is spent to (1) retrieve the run-time information of a given ESL design (phase 1) and (2) perform the regression analyses on the extracted information (phase 2 to 4). As illustrated in Table I the first phase is the major part of the total execution time. Even for complex designs such as the *JPEG-encoder* or the *RISC-CPU* it is within reasonable boundaries in comparison with their compilation and execution time though, allowing it to be used in common development environments. The improvement in the first phase has a direct positive effect on total execution time. The complexity of the regression analysis mostly depends on the creation of the predictive model, which comes with a (worst case) quadratic computational complexity (for GP regression). The fast run-time in phase 3 (order of second) is because of two main reasons. First, we defined the granularity of finding approximable portion of a given design by module in comparison to the traditional sensitivity analysis where the granularity is variables in a program. Second, we take advantage of fast learning model of regression analysis that the predicted model can be estimated in order of second. However using complex machine learning techniques (e.g. neural network) may increase the ability of learning phase to find more approximable module of a design, the time needs for the training and analysis phases is significantly high. This can reduce the applicability of the method.

Although our method has some limitations, we believe this new promising line of research is helpful for making approximate computing truly a cross-cutting activity in the early stages of design process.

V. Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project SecRec under grant no. 16K1S0606K, the subproject P01 "Predictive function" of the Collaborative Research Center SFB1232, funded by the German Research Foundation and by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

VI. Conclusion

In this paper, we proposed the first method to automatically detect the approximable components of ESL designs. The method is based on statistical analysis on the simulation behavior of each design's module by applying different machine learning techniques to estimate the relationship between its output and input signals. We mapped the simulation behavior of each module onto a training data set and performed different types of regression analysis on it. For the approximable module, the QoS threshold provides the designer with an estimation on output quality of the module in order to meet the reference QoS of the design. Moreover, a data significance analysis was performed to rank the importance of input signals for each output of the module. Several ESL benchmarks were run to evaluate the effectiveness of the proposed method.

References

[1] G. Martin, B. Bailey, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[2] "IEEE Standard SystemC Language Reference Manual," 2006, pp. 1–423.

[3] C. Schulz-Key, M. Winterholer, T. Schweizer, T. Kuhn, and W. Rosenstiel, "Object-oriented modeling and synthesis of SystemC specifications," in *ASP-DAC*. IEEE Press, 2004, pp. 238–243.

[4] B. Nongpoh, R. Ray, S. Dutta, and A. Banerjee, "Autosense: A framework for automated sensitivity analysis of program data," *TSE*, pp. 1–1, 2017.

[5] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, p. 62, 2016.

[6] Z. Du, A. Lingamneni, Y. Chen, K. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *ASP-DAC*, 2014, pp. 201–206.

[7] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *SIGPLAN Not.*, no. 6, 2011, pp. 164–174.

[8] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," *SIGPLAN Not.*, pp. 213–224, 2011.

[9] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *DAC*, 2013, pp. 1–9.

[10] P. Roy, R. Ray, C. Wang, and W. F. Wong, "Asac: Automatic sensitivity analysis for approximate computing," *SIGPLAN Not.*, pp. 95–104, 2014.

[11] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *DATE*, 2014, pp. 1–6.

[12] M. Carbin and M. C. Rinard, "Automatically identifying critical input regions and code in applications," in *ISSTA*, 2010, pp. 37–48.

[13] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *DAC*. ACM, 2010, pp. 555–560.

[14] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: imprecise adders for low-power approximate computing," in *ISLPED*, 2011, pp. 409–414.

[15] J. Lee and A. Shrivastava, "Static analysis to mitigate soft errors in register files," in *DATE*, 2009, pp. 1367–1372.

[16] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *DATE*, 2011, pp. 1–6.

[17] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *MICRO*, 2012, pp. 449–460.

[18] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *ISCA*, 2012, pp. 356–367.

[19] V. Vassiliadis, J. Riehme, J. Deussen, K. Parasyris, C. D. Antonopoulos, N. Bellas, S. Lalis, and U. Naumann, "Towards automatic significance analysis for approximate computing," in *CGO*, 2016, pp. 182–193.

[20] G. Gillani and A. Kokkeler, "Improving error resilience analysis methodology of iterative workloads for approximate computing," in *CF*, 2017, pp. 374–379.

[21] A. Kassambara, *Practical Guide to Cluster Analysis in R: Unsupervised Machine Learning*. CreateSpace Independent Publishing Platform, 2017.

[22] P. Stalph, *Analysis and design of machine learning techniques: evolutionary solutions for regression, prediction, and control problems*. Springer Science, 2014.

[23] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *AIR*, pp. 85–126, 2004.

[24] B. C. Schafer and A. Mahapatra, "S2CBench: Synthesizable SystemC benchmark suite for high-level," *IEEE Embedded Systems Letters*, no. 3, pp. 53–56, 2014.

[25] A. S. Initiative., http://www.accellera.org/downloads/standards/systemc, 2016.

[26] M. Goli, J. Stoppe, and R. Drechsler, "AIBA: an Automated Intra-cycle Behavioral Analysis for SystemC-based design exploration," in *ICCD*. IEEE, 2016, pp. 360–363.