Approximate Hardware Generation using Symbolic Computer Algebra employing Gröbner Basis

Saman Froehlich Daniel Große Rolf Drechsler

Cyber-Physical Systems, DFKI GmbH and Group of Computer Architecture, University of Bremen, Germany
saman.froehlich@dfki.de {grosse,drechsle}@cs.uni-bremen.de

Abstract—**Many applications are inherently error tolerant. Approximate Computing is an emerging design paradigm, which gives the opportunity to make use of this error tolerance, by trading off accuracy for performance.**

The behavior of a circuit can be defined at an arithmetic level, by describing the input and output relation as a polynomial. Symbolic Computer Algebra (SCA) has been employed to verify that a given circuit netlist matches the behavior specified at the arithmetic level.

In this paper, we present a method that relaxes the exactness requirement of the implementation. We propose a heuristic method to generate an approximation for a given netlist and use SCA to ensure that the result is within application-specific bounds for given error-metrics. In addition, our approach allows for automatic generation of approximate hardware wrt. application-specific input probabilities. To the best of our knowledge taking input probabilities, which are known for many practical applications, into account has not been considered before. We employ the proposed approach to generate approximate adders and show that the results outperform state-of-the-art, handcrafted approximate hardware.

I. Introduction

Approximate Computing is an emerging field of research, which deals with exploiting the inherited error tolerance of applications. Approximate Computing tries to improve the performance of these applications in terms of computation time, power consumption and/or hardware complexity by introducing additional errors, which are either timing induced or caused by functional approximation. This paper focuses on the later approach of functional approximation.

A lot of research in the field of automating the design of data paths from high-level specifications using *Symbolic Computer Algebra* (SCA) has been done (see e.g. [1]). At the same time, SCA has been employed in the field of verification of circuits [2], [3], [4], [5].

In this paper we propose to use the techniques derived in both fields as follows: We present a heuristic method to automatically generate an approximate circuit for a given high-level specification employing SCA for error-metric evaluation. Starting with the non-approximated netlist, we remove gates to approximate the circuit. Using SCA we guarantee that user specific bounds for error-metrics hold. We show that the results of our approach can outperform state-of-the-art handcrafted approximate hardware.

It has been shown that for many practical applications non-uniform input distributions occur (see e.g. [6]). Hence, we extend our approach to take this information into account. By this, we allow for even further improved approximation.

II. Related Work

There has been a lot of research in the field of verifying circuits on gate-level using Gröbner Basis [2], [3], [4], [5]. The basic idea is that given a set of polynomials describing the internal structure of the circuit (its netlist), one can conclude that the resulting polynomial, which describes the relation of the output to the input, equals the desired functionality (which is also given in polynomial form). We employ techniques based on Gröbner Basis for error-metric computation.

Some approaches towards the generation of approximate hardware exist:

In [7] the authors have generated approximate hardware using *And-Inverter Graphs* (AIGs). This technique allows the reduction of hardware complexity and critical path length. However, they don't support input probabilities. Further, the authors only consider gates on the critical path, while we propose to adopt the considered gates based on the approximation goal.

SALSA, introduced in [8], presents a methodology for automatic approximate hardware generation. It utilizes Approximate Don't Cares and Q-functions for generating approximate hardware and maps it to a synthesis problem. Further, ASLAN [9] has been presented, which extends SALSA to an algorithm for automatic generation of approximate sequential circuits. This requires the definition of a Quality Evaluation Circuit, which is a design problem itself. Again these methodologies don't support input probabilities.

III. Preliminaries

We assume that the reader is familiar with the basics of SCA in the context of circuit verification. Essentially, SCA-based verification performs a series of divisions of the specification polynomial by the circuit polynomials (also known as Gröbner basis reduction). For details we refer the reader to [4], [3].

Over the past years several metrics have been used in approximate computing. One of the most popular metrics is the *worst-case error* (wc). It can be defined as

$$wc(f,\hat{f}) = \max_{x}\{|f(x) - \hat{f}(x)|\}. \tag{1}$$

In [10] the authors propose a *mean squared error* (mse) error-metric:

$$mse(f,\hat{f}) = \frac{(\sum_{x}(f(x) - \hat{f}(x))^2)}{2^n}. \tag{2}$$

Acknowledgments. This work was supported in part by the German Research Foundation (DFG) within the project MANIAC (DR 287/29-1), the Reinhart Koselleck project DR 287/23-1 and by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

IV. Approximate Circuit Generation

In this section we introduce the definition of the problem at hand and a description on how to determine the difference between an implementation of a given circuit and its specification in Section IV-A. In Section IV-B we describe how to use SCA for the computation of error-metrics.

A. Problem Formulation

Approximate Circuit Generation.
Given a pseudo-Boolean polynomial $f(X = x_1, \ldots, x_n) : \mathbb{B}^n \to \mathbb{Z}$, which is a definition of the system behavior of a circuit at an arithmetic level, a cost function c *and an error bound* B*, find a netlist* N *which realizes a function $\hat{f}(X)$, such that $\hat{f} = \min_N c(N)$ and $e(N, f) \leq B$ for some error function e.*

We propose to tackle this problem heuristically by removing gates from the netlist N of the implementation of f, until the error bound is reached. We remove a gate by replacing its output with one of its inputs. An algorithm for this problem is presented in Section V. This algorithm also requires to calculate the error function e. In general, e calculates the difference between the polynomial representation f and its approximate realization $\hat{f}$ and evaluates it in terms of an error-metric. Hence, we provide a method to determine a functional representation of the difference between a polynomial representation f and its approximate realization $\hat{f}$ next. Afterwards we show how to evaluate this representation in terms of different error-metrics.

Lemma 1. *In order to determine the difference between a polynomial representation f and its approximate realization $\hat{f}$, it is sufficient to consider the remainder R of the realization of f when it is expressed in terms of the Gröbner Basis $\hat{B}$ of $\hat{f}$. If the remainder is zero, then $\hat{f}$ is the exact implementation of f. Otherwise, $\hat{f}$ is the exact implementation of $f - R$.*

Proof. The proof is straight forward and can be deduced by using the representation of f and $\hat{f}$ in terms of the Gröbner Basis of the approximate circuit. □

B. Error-Metrics Calculation for Pseudo-Boolean Functions

After presenting a method to find a representation for the difference between a polynomial representation f and its approximate realization $\hat{f}$, we now derive how to evaluate the remainder R in terms of the error-metrics presented in Section III. In general, R is a pseudo-Boolean function $R : \mathbb{B}^n \to \mathbb{Z}$. Evaluating R for any input combination will yield the difference between f and $\hat{f}$ for the concrete input.

1) wc-error: Given the remainder R, the evaluation of the worst-case error results in the classical pseudo-Boolean optimization problem.

There is a lot of theory about finding global minima/maxima of pseudo-Boolean functions. This problem is known to be NP-hard and is tackled in different ways using solvers of very different fields [11]. We use the optimization feature of an SMT-solver in this work. Since R is a pseudo-Boolean function, it can be passed to the SMT-solver without any reformulation.

2) mse-error: We propose to calculate the average of a multilinear pseudo-Boolean function in a recursive way and use this formulation to calculate the mse-error.

Every pseudo-Boolean function $f(X)$ with $X = x_1, \ldots, x_n$ can be written in the form

$$f(X) = x_1 \cdot g(x_2, \ldots, x_n) + h(x_2, \ldots, x_n)$$

[12]. If the input probability $p(x_i)$ for which the input x_i is set to 1 is known, we conclude

$$\begin{aligned} avg(f(X)) &= \frac{\sum\limits_{x \in \mathbb{B}^n} f(x)}{2^n} \\ &= p(x_1) \cdot avg(g(x_2, \ldots, x_n)) + avg(h(x_2, \ldots, x_n)). \end{aligned} \tag{3}$$

We can use Eq. 3 to calculate the average value of pseudo-Boolean functions in a recursive manner. If the input probability of $p(x_i) = 0.5\ \forall i$, then

$$mse(R(x)) = avg(R(x)^2) \tag{4}$$

However, it makes sense to extend this definition if the distribution of the input probabilities is known. We propose that if the distribution of the input probabilities is known, this should be taken into account in the mse-error-metric. Thus, the average of $R(x)^2$ in Eq. 4 should always be calculated wrt. to the given probability distribution. If no distribution is given then naturally $p(x_i) = 0.5\ \forall i$ is assumed.

V. Proposed Algorithm

In this section we introduce the algorithm for the generation of an approximate circuit $\hat{f}(X)$ for a given functional description $f(X)$ and its initial non-approximated gate-level netlist N. We start with the overview of the proposed algorithm in Section V-A and assumptions we make in order to simplify the problem. It is followed by an algorithm to minimize the *Gate Count* (GC) for a given netlist in Section V-B. Afterwards we give a description on how to adapt the algorithm to optimize the circuit wrt. the *Critical Path Length* (CPL) in Section V-C.

A. Algorithm Overview

The goal of the proposed algorithm is to create an approximation $\hat{f}$ by replacing gates in N with one of their respective inputs.

1) Simplifications: We solve the problem heuristically by making the following assumptions:

1) Whenever we remove a gate, the error does not decrease
2) Whenever the remainder R gets too complex, the resulting error will be large. The complexity of the evaluation of R can be estimated by counting the number of literals.

The first assumption allows us to stop checking new combinations, whenever the error-bound is violated.

The second assumption allows us to configure the quality of approximation as a trade-off for run-time.

2) Vanishing Monomials: During the elimination process the size of the polynomial can grow exponentially in the number of monomials [4], [3]. These monomials vanish if the specification meets the implementation. Approaches to deal with these blow ups have been proposed. We utilize the state-of-the-art method of logic reduction (a mix of XOR rewriting followed by common rewriting) as proposed in [3].

However, during the approximation process netlists may occur for which these monomials do not vanish. Since the computation time of the elimination process may become infeasible, we limit the number of monomials that may appear during the polynomial division. If this limit is reached, we assume that the implementation does not meet the required error bound.

B. Algorithm

In this section we introduce a basic algorithm to determine an approximation for the original circuit.

Algorithm 1 Approximate Circuit Generation

```
 1: function ACI(f,N,errFun,B)
 2:   $result = N$;
 3:   $newApproximation.N = N$;
 4:   $newApproximation.error = B$;
 5:   for all $g \in N.gates$ do
 6:     for all $i \in g$ do
 7:       $\hat{N} = N$;
 8:       replace($\hat{N}, g, i$);
 9:       $error$ =errFun($\hat{N}, f$);
10:       if $error < newApproximation.error$ then
11:         $newApproximation.N = \hat{N}$;
12:         $newApproximation.error = error$;
13:       end if
14:     end for
15:   end for
16:   if $newApproximation.N \neq N$ then
17:     return ACI($f, newApproximation.N, errFun, B$);
18:   else
19:     return $result$;
20:   end if
21: end function
```

We construct a greedy algorithm depicted in Algorithm 1 as a heuristic to find an approximation $\hat{f}$ for a given function f. The inputs are the symbolic representation of the function f which is to be approximated and its corresponding netlist N. The error-metric is encoded in the parameter $errFun$ (see Section III) and its bound by the parameter B.

In Line 5 we iterate over all gates in N. We replace the current gate by one of its inputs in Line 8. We calculate the error of the approximation in Line 9. If the expression of the remainder R gets too complex, we assume that the error is ∞. If the error bound for the error calculated by replacing the gate holds, we check if it is smaller than for any other previous computed approximation in Line 10. If so, we store the current approximation. In Line 16, we check if any approximation has been found, for which the error bound holds. If this is the case, we recursively call our algorithm and remove another gate. If not, we return the netlist which was given, since no other gate can be removed without breaking the error bound.

C. Optimization wrt. Critical Path Length

If one wants to optimize for delay instead of area, only gates on the critical path are of interest. So instead of removing gates from anywhere in the circuit, the algorithm can be restricted to the critical path. The critical path has to be determined initially and redetermined after each removal.

VI. Experimental Results

We have implemented the parallelized version of Algorithm 1 in C++. The wc-error is evaluated using the optimization feature of the SMT-solver Z3 [13]. All experiments have been carried out on an Intel® Xeon® CPU E5-2630 v3 @ 2.40GHz with 64GB memory running Linux (Fedora release 22). We set the maximum number of monomials which may occur during the division process to $10,000$ for all experiments.

A. Optimization wrt. Gate Count

We have optimized an 8-bit *Ripple-Carry-Adder* (RCA) using the wc-error-metric and the mse-error-metric in terms of *Gate Count* (GC). We have set the complexity-limit of the remainder to 750 for the mse-error and to 200 for the wc-error. The results can be seen in Table I and Table II. The first column gives the name of the architecture. The second column shows the calculated error and the third column the remaining number of gates. The fourth column denotes the computation time in CPU seconds. We set 4, 8, 16, 32, 64 and 128 as limits for the wc-error. We used 96 and 408 as limits for the mse-error-metric, since these are values that many architectures of handcrafted approximate architectures from the repository [14] use. The first row shows the values for the golden, non-approximated RCA architecture in both tables. The remaining rows show the results of our algorithm. Their names are encoded as follows: $< approximation\ goal >$ _ $< error - metric >$ _ $< error\ bound >$. It can be seen that when optimizing wrt. GC, large improvements in the number of gates can be achieved.

Table II also shows the effect when taking input probabilities into account. For the fourth and fifth row (GC_mse_96_prob and GC_mse_408_prob), we have set the probability of the lower half input bits of each input word to 0.5, while setting the probability of the upper half input bits of each input word to 0.1. By this, we simulate that the domain of the inputs is mostly below 16. As can be seen specifying input probabilities allows us to reduce the GC even further, since more gates in the region which influences the higher order outputs can be removed: Instead of reducing the GC to 29 for mse-error limit of 96, we now get a circuit with only 20 gates (second row vs. fourth row). For a mse-error limit of 408, the reduction in gate count is from 24 to 12 (third row vs. fifth row).

B. Optimization wrt. Critical Path Length

In this subsection we consider another optimization criteria, i.e. we optimize wrt. *Critical Path Length* (CPL) (cf. Section V-C). To evaluate the quality of the proposed approach, we compare against state-of-the-art handcrafted approximate adder architectures with a wc-error of 64 or 128, respectively. They have been taken from the repository [14] and

TABLE I

APPROXIMATIONS FOR AN 8-BIT ADDER IN TERMS OF GATE COUNT FOR wc-ERROR

Approximation	wc-error	gate count	calc. time [s]
RCA_8	0	49	-
GC_wc_4	3	42	4.8
GC_wc_8	7	39	6.5
GC_wc_16	12	29	11.4
GC_wc_32	28	21	16.2
GC_wc_64	59	16	20.6
GC_wc_128	124	14	21.5

TABLE II

APPROXIMATIONS FOR AN 8-BIT ADDER IN TERMS OF GATE COUNT FOR mse-ERROR

Approximation	mse-error	gate count	calc. time [s]
RCA_8	0	49	-
GC_mse_96	21.5	29	12.5
GC_mse_408	249.5	24	16.9
GC_mse_96_prob	87.9	20	11.0
GC_mse_408_prob	390.4	12	11.7

synthesized to gate-level using AND, OR and XOR gates with Yosys 0.7 [15]. In our approach, we have set the complexity-limit of the remainder to 2000 and use 64 and 128 as bounds since these were the most common wc-errors computed for the state-of-the-art approximate adder architectures. We use ABC 1.01 [16] to calculate the delay of our results and of the handcrafted adders after mapping them to the library $mcnc.genlib$.

The results can be seen in Fig. 1(a) and Fig. 1(b), respectively. The y-axis denotes the calculated delay in *ns* for each architecture. The black bar represents the delay of the golden non-approximated RCA. The gray bars refer to different adder architectures (we have used the same abbreviations as given in the Library [14]). The dashed bar refers to the result of our proposed approach. The naming is the same as introduced in the previous section. All results were computed in less than $120s$.

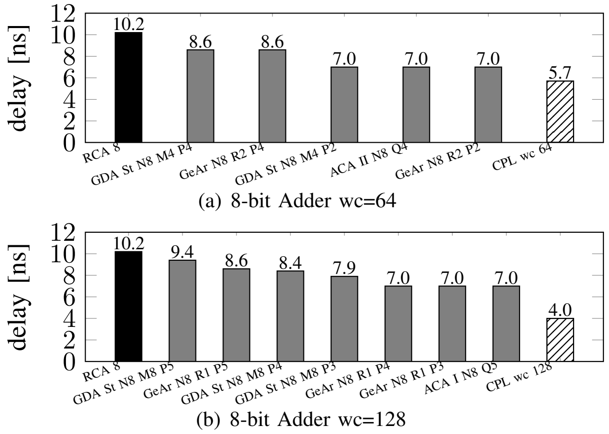
As can be seen our proposed approach has reduced the delay of the RCA significantly and outperformed all compared architectures.

We have also considered larger circuits as benchmarks by approximating a 16-bit adder using the wc-error-metric in terms of CPL. We have used 1024 and 4096 as error bounds and again compared the results to state-of-the-art handcrafted approximate adders from the repository [14]. Results can be seen in the Figures 2(a) and 2(b). All results were computed in less than $4h$. Again the results from the heuristic outperform the architectures from the repository.

VII. CONCLUSIONS

We have proposed a method to automatically generate an approximate circuit for a given high-level specification under accuracy constraints wrt. a given optimization goal. Our method employs *Symbolic Computer Algebra* (SCA) for error metric evaluation. SCA produces a remainder polynomial representing the error of the approximation. This polynomial can be easily interpreted and evaluated.

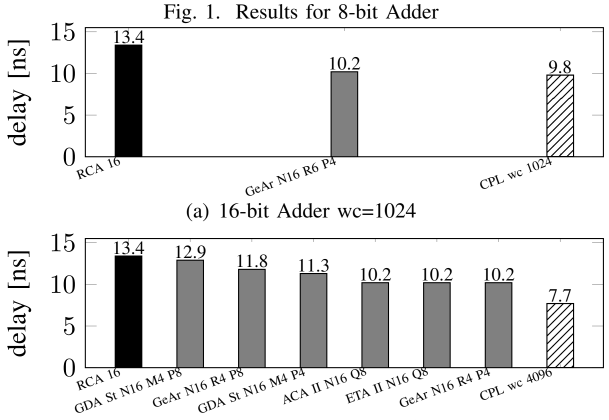
We have used our approach to optimize an 8-bit RCA in terms of GC wrt. different error-metrics. Furthermore, we considered the critical path length as alternative optimization goal during approximation. In the experiments we have shown that our approach produces much better results in comparison to state-of-the-art handcrafted approximated architectures.



(a) 8-bit Adder wc=64

(b) 8-bit Adder wc=128

Fig. 1. Results for 8-bit Adder



(a) 16-bit Adder wc=1024

(b) 16-bit Adder wc=4096

Fig. 2. Results for 16-bit Adder

REFERENCES

[1] A. Peymandoust and G. De Micheli, “Application of symbolic computer algebra in high-level data-flow synthesis,” *TCAD*, vol. 22, no. 9, pp. 1154–1165, 2003.

[2] S. Ghandali, C. Yu, D. Liu, W. Brown, and M. J. Ciesielski, “Logic debugging of arithmetic circuits,” in *ISVLSI*, 2015, pp. 113–118.

[3] A. S. Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler, “Formal verification of integer multipliers by combining Gröbner basis with logic reduction,” in *DATE*, 2016, pp. 1048–1053.

[4] F. Farahmandi and B. Alizadeh, “Groebner basis based formal verification of large arithmetic circuits using gaussian elimination and cone-based polynomial extraction,” *MICPRO*, vol. 39, no. 2, pp. 83 – 96, 2015.

[5] D. Ritirc, A. Biere, and M. Kauers, “Column-wise verification of multipliers using computer algebra,” in *FMCAD*, 2017, pp. 23–30.

[6] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” *TCAD*, vol. 32, pp. 124–137, 2013.

[7] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, “Approximation-aware rewriting of AIGs for error tolerant applications,” in *ICCAD*, 2016, pp. 83:1–83:8.

[8] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, “Salsa: Systematic logic synthesis of approximate circuits,” in *DAC*, 2012, pp. 796–801.

[9] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, “Aslan: Synthesis of approximate sequential circuits,” in *DATE*, 2014, pp. 364:1–364:6.

[10] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, “Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in *DATE*, 2017, pp. 258–261.

[11] T. Berthold, S. Heinz, and M. E. Pfetsch, “Nonlinear pseudo-boolean optimization: Relaxation or propagation?” in *SAT*, 2009, pp. 441–446.

[12] Y. Crama, P. Hansen, and B. Jaumard, “The basic algorithm for pseudo-boolean programming revisited,” *Discrete Applied Mathematics*, vol. 29, no. 2, pp. 171–185, 1990.

[13] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in *TACAS*, 2008.

[14] Chair for Embedded Systems - Karlsruhe Institute of Technology, “Gear - approxadderlib.” [Online]. Available: http://ces.itec.kit.edu/GeAR.php

[15] C. Wolf, “Yosys - yosys open synthesis suite.” [Online]. Available: http://www.clifford.at/yosys/about.html

[16] A. Mischenko, M. Case, R. Brayton, and S. Jang, “Scalable and scalably-verifiable sequential synthesis,” in *ICCAD*, 2008, pp. 234–241.