Precise Error Determination of Approximated Components in Sequential Circuits with Model Checking

Arun Chandrasekharan[1] Mathias Soeken[2] Daniel Große[1,3] Rolf Drechsler[1,3]

[1]Group of Computer Architecture, University of Bremen, Germany

[2]Integrated Systems Laboratory, EPFL, Switzerland

[3]Cyber Physical Systems, DFKI GmbH, Bremen, Germany

{arun,grosse,drechsle}@cs.uni-bremen.de, mathias.soeken@epfl.ch

ABSTRACT

Error metrics are used to evaluate the quality of an approximated circuit or to trade-off several approximated candidates in design exploration. Precisely determining the error of an approximated circuit is a hard problem since the errors accumulate over time depending on the composition and nature of individual components. In this paper, we present methods based on model checking to precisely determine error behavior in sequential circuits that contain approximated combinational components. Our experiments show that such an analysis is very significant and crucial to properly deduce the effects of approximations.

1. INTRODUCTION

There is a high demand for approximate computing that allows to achieve significant improvements in performance by relaxing the requirements of correct functional or non-functional properties (see, e.g., [20]). A large number of applications allow approximation including DSP, multimedia, recognition, or mining. Reasons that allow approximation are, e.g., the limited perceptual capability of humans or imprecision due to non-available exact results.

In recent years, approaches have been presented for approximated building blocks (e.g, [18, 22, 3]) or synthesis methods to derive approximated components from golden designs (e.g., [16, 19, 13]). The degree and acceptance of approximation is measured with respect to error metrics, such as error rate, worst-case error, or average-case error. Several heuristic and statistical methods have been proposed to estimate the error of an approximated circuit. Precisely computing error metrics is a hard problem, but it is inevitable when aiming for high quality results or when trading off candidates in design space exploration. Exact methods based on *Boolean Satisfiability* (SAT) or *Binary Decision Diagrams* (BDDs) have been proposed in the past for combinational circuits [20, 17].

Precisely determining the error metric of a combinational circuit is already very helpful in the design of approximate computing, but the obtained numbers may not be accurate when considering approximated components in sequential circuits. As an example, although the worst-case can be computed for the approximated component in isolation, the accumulated worst-case in the sequential circuit may differ significantly, since only a subset of the input patterns may actually be assignable. Further, the sequence of successive input patterns for the approximated component depends on the sequential logic and composition of the overall circuit.

In this paper, we propose algorithms based on model checking that are able to prove the absence of errors such as accumulated worst-case, maximum worst-case, or average error-rate—although theoretically possible when only considering the approximated component in isolation. For this purpose, we propose the concept of an *approximation miter* that incorporates both error computation and accumulation over time in sequential circuits. State-of-the-art model checking techniques can then be used to prove the presence or absence of an error.

To summarize, the major contributions of this paper are:

- Presentation of a generic configurable approximation miter; it enables the formulation of exact formal error determination instances
- Reduction of exact error metric computation to decision or optimization problems using formal verification techniques
- First exact approach to analyze the effect of errors introduced by approximated components in sequential circuits

The remainder of this paper is structured as follows. First, in Section 2 the preliminaries are provided. Then, our approaches for precise error determination of approximated components in sequential circuits are introduced. The experimental evaluation is given in Section 4. Finally, the paper is concluded in Section 5.

2. PRELIMINARIES

This section gives the basics on formal verification already with the focus on its application in the context of approximate computing. The second part of the section briefly reviews the basic error metrics typically used to measure the quality of approximated combinational components.

DAC '16, June 05-09, 2016, Austin, TX, USA

© 2016 ACM. ISBN 978-1-4503-4236-0/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2897937.2898069

2.1 Formal Verification

In the last decade formal verification techniques have been intensively used in the industry. Essentially, formal verification can be divided into model checking[1] and (sequential) equivalence checking. Based on the recent advancements using e.g. BDDs, SAT, induction, interpolation, and *property-directed reachalibity* (PDR), model checking and sequential equivalence checking have become quite similar.

Equivalence checking is the subclass of formal verification where two different implementations of the same logic is exhaustively verified to be *equivalent*. The notion of equivalence is that both implementations produce the same sequence of outputs under all possible sequences of input conditions, starting from the same initial state [12, 14]. Equivalence checking when applied to approximated hardware aims to establish whether both of the implementations produce outputs that do not differ beyond a certain predefined quality threshold, under all possible sequences of input combinations.

When sequential elements are present in the design, the number of states for equivalence checking increases exponentially and becomes easily non tractable. To deal with this state explosion problem several approaches have been proposed. One such approach is *Bounded Model Checking* (BMC) [4] where the length of the sequence over which the formal verification is employed is finite. When applied to sequential equivalence checking the circuit is unrolled to a predefined number of time frames and the overall problem including the properties is solved using a SAT solver. This approach is feasible as the current state-of-the-art SAT solvers [8] can solve millions of clauses in moderate time. Besides BMC, tools have evolved to use *inductive reasoning* [24] and more recently PDR [5, 1] techniques which give vast improvements over the original method.

2.2 Error Metrics

Different error metrics have been proposed to qualitatively and quantitatively measure the quality of approximations [2, 20, 10]. A brief overview of the metrics relevant to our work is given below.

Let $f : \mathbb{B}^n \to \mathbb{B}^m$ be a Boolean function and $\hat{f} : \mathbb{B}^n \to \mathbb{B}^m$ an approximation of it. Then, the *worst-case error*, also called *error significance*, is defined as

$$e_{\mathrm{wc}}(f, \hat{f}) = \max_{x \in \mathbb{B}^n} \left| \mathrm{int}(f(x)) - \mathrm{int}(\hat{f}(x)) \right|, \tag{1}$$

i.e., the maximum among the absolute differences of f and $\hat{f}$ over all the possible input combinations, where 'int' denotes the integer representation of the bit vector.

Similarly, the *average-case error*

$$e_{\mathrm{ac}}(f, \hat{f}) = \frac{\sum_{x \in \mathbb{B}^n} \left| \mathrm{int}(f(x)) - \mathrm{int}(\hat{f}(x)) \right|}{2^n} \tag{2}$$

gives the average error introduced by approximations per input combination.

Finally, the *error-rate*

$$e_{\mathrm{er}}(f, \hat{f}) = \frac{\sum_{x \in \mathbb{B}^n} [f(x) \neq \hat{f}(x)]}{2^n} \tag{3}$$

counts the percentage of input assignments that lead to a different output pattern. All these error metrics can be precisely computed for combinational circuits with the symbolic algorithms given in [17].

[1] also called property checking

3. PRECISE ERROR DETERMINATION

In this section the proposed approaches for exact error determination for approximated components in sequential circuits are presented. At first, we give the general idea introducing the approximation miter. Afterwards, we explain the components of the approximation miter and show its generic configurable form. Altogether, we can ask different questions which can be precisely answered based on suitable configurations of the approximation miter and the reduction to decision or optimization problems.

3.1 General Idea

Our proposed approach precisely computes accumulated errors of combinational components in sequential circuits. The main idea is to apply model checking to an *approximation miter* which is illustrated in Fig. 1. The approximation miter consists of five circuits: (i) the non-approximated original circuit C, (ii) the circuit $\hat{C}$ that is obtained by replacing some combinational components with approximated implementations, (iii) a circuit E to compute the error of $\hat{C}$'s outputs with respect to C's outputs and a given error metric, (iv) an accumulator A that accumulates the computed errors in each cycle and (v) a decision circuit D. The decision circuit exposes the single output 'bad' of the approximation miter, which is 1 if and only if the accumulated error violates a given requirement. Formal model checking techniques can be applied to show that the output signal never evaluates to 1 either unbounded or within some given number of cycles.

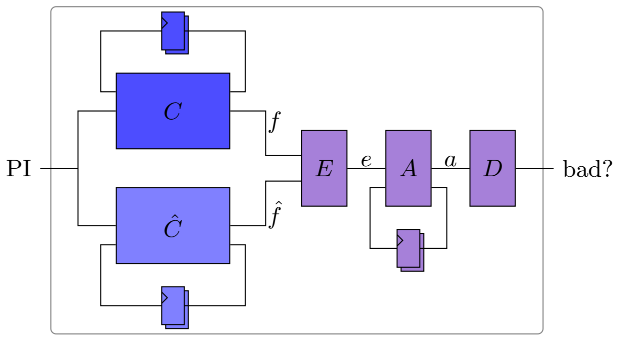

Figure 1: General idea

3.2 Approximation Miter

This section illustrates the components of the approximation miter in detail. Note that these are only selected examples that can all be exchanged by user defined implementation to create custom scenarios. Some example scenarios based on the selected implementations are presented in the end of this section.

3.2.1 Error computation

The stateless error computation E takes as input the output results of C and $\hat{C}$, accessible through the wires f and $\hat{f}$. Not necessarily all outputs are of interest and therefore not all outputs of C and $\hat{C}$ may be contained in f and $\hat{f}$. Further, the order of bits is set to adhere the desired numeric interpretation.

For worst case and average case type computations, E is implemented as

$$e = \left|\operatorname{int}(f) - \operatorname{int}(\hat{f})\right|. \tag{E1}$$

For error rate type computations, E is

$$e = f \oplus \hat{f} \tag{E2}$$

or

$$e = \sum_{i=1}^{m} \left(f_i \oplus \hat{f}_i\right), \tag{E3}$$

where the first equation computes a bit-string that is 1 at positions where the output results differ and the second equation computes the amount of that bits.

3.2.2 Accumulation

The accumulation has state and can store intermediate results. A straight-forward implementation is to add all errors:

$$\begin{aligned} a' &= a' + e \\ a &= a', \end{aligned} \tag{A1}$$

where a' is a state variable that stores the current sum of all errors and is initialized with 0. In the remainder of this section, all state variables are primed and initialized with 0. An accumulator can also track the maximum error:

$$\begin{aligned} a' &= \max(a', e) \\ a &= a'. \end{aligned} \tag{A2}$$

When dealing with error rate and bit masks, binary operations are of interest. As an example to accumulate changed bits one can use the following implementation:

$$\begin{aligned} a' &= a' \mid e \\ a &= a', \end{aligned} \tag{A3}$$

where '|' refers to bitwise OR. Additional state variables can be employed for more complex computations. In order to keep track of an average error, also the number of cycles need to be tracked:

$$\begin{aligned} a' &= a' + e \\ c' &= c' + 1 \\ a &= a'/c' \end{aligned} \tag{A4}$$

3.2.3 Decision

Typically the decision circuit implements a comparison with respect to a threshold X, the most common one being

$$\text{bad} = a \geq X, \tag{D1}$$

that asserts 'bad' if the accumulated error is greater or equal than the given threshold.

3.3 Approximation Questions

The approximation miter with the previous discussed implementations for its components can now combined in several different ways that allows to answer major questions, of which five are presented in the following.

Question 1: What is the earliest time that I can exceed an accumulated worst-case error of X?

This question can be answered by using the implementations E1, A1, and D1 and then performing BMC. If BMC returns a counter-example at time step t, one tries to find another counter-example up to time step $t-1$.[2] This procedure is repeated until BMC cannot find a counter-example anymore; the earliest time is the latest found t.

Question 2: What is the maximum worst-case error?

This question is not a decision problem but an optimization problem. For this purpose we use the approximation miter with implementations E1, A2, and D1. The *worst-case error* is found out using binary search with PDR. The binary search is shown in Algorithm 1. X is set to one half of $2^m - 1$ in the first loop, with lower bound 0 and upper bound $2^m - 1$ where m is the bit-width of the considered output vector. PDR is used to solve the miter and if PDR returns `True`, the lower bound is set to X, otherwise the upper bound is set to $X - 1$. The upper and lower bounds are refined iteratively until they converge to the *worst-case error.*

Algorithm 1 Finding maximum worst-case error

```
 1: function FIND_WORST_CASE_ERROR
 2:     $lower\_bound \leftarrow 0$
 3:     $upper\_bound \leftarrow 2^m - 1$
 4:     while $lower\_bound < upper\_bound$ do
 5:         $X = \left\lceil \frac{(upper\_bound + lower\_bound)}{2} \right\rceil$
 6:         $status = PDR(Miter(E1, A2, D1), X)$
 7:         if $status$ = True then
 8:             $lower\_bound = X$
 9:         else
10:             $upper\_bound = X - 1$
11:         end if
12:     end while
13:     return $lower\_bound$
14: end function
```

Question 3: What is the earliest time that I can reach an accumulated error rate of X?

This question can be solved in the same way as Question 1, but with E2, A3, and a decision circuit that takes into consideration the number of bits in the accumulated error:

$$\text{bad} = \sum_{i=1}^{m} a_i \geq X$$

Question 4: What is the maximum error rate?

This question can be solved in the same way as Question 2 by only replacing E1 with E3, in the error computation. The initial lower bound remains the same but the upper bound is initialized to m, the bit-width of the output vector. We use similar binary search as given in Algorithm 1 with the PDR call on the $Miter(E3, A2, D1)$.

[2]This 2^{nd} check may be required for advanced BMC implementations which do not guarantee a counter-example of minimal length.

Question 5: Can I guarantee that the average-case error does not exceed X?

Sometimes one is interested that the error does not grow too large over time although some exceptions are tolerable. As an example, if only small errors have been made for a long time one can except a larger one. For this purpose, one needs to track the average case error which is done by using the approximation miter with E1, A4, and D1. The result is a decision problem, for which we have: If no counter-example can be found, the average-case behavior can be guaranteed.

In total, we have demonstrated that the proposed approximation miter is generic and can be configured such that different major questions can be formulated. By reducing the respective problems either as decision or optimization problem they can be answered using model checking techniques.

At this point we would also like to differentiate our work with the techniques presented in [13], where the outputs of original and approximated circuits are compared using a user provided *Quality Evaluation Circuit*, QEC, which is formally verified for liveness and safety properties. However in [13], it is not clear how the individual error metrics are evaluated and thus the quality is ensured. In particular, the optimization problems addressed in our work are very unlikely to be represented as a circuit in the form of QEC.

In the following section we demonstrate our approaches on several examples.

4. EXPERIMENTAL RESULTS

We have implemented all algorithms in C++ as part of our own formal verification package.[3] The program reads Verilog RTL descriptions of the the approximated and non-approximated design using Yosys [21] to create the approximate miter. We use ABC [11] to perform model checking of the miter. In our experimental evaluation we tried to answer Questions 1–4 from Section 3.3 on various open designs from OpenCores and GitHub. Question 5 involves the use of a divider as given in (A4) and the divider is hard to verify formally [7]. Our initial experiments on Question 5 did not conclude on practical designs and as a result the implementation of Question 5 is left out for future work. The experiments are carried out on an Octa-Core Intel Xeon CPU with 3.40 GHz and 32 GB memory running Linux 4.1.6.

The experimental evaluation is described in two parts. First, an extensive case study of approximated adders in sequential multipliers is discussed in Section 4.1. Second, the generality and scalability of the approach is demonstrated by applying it to various designs in Section 4.2.

4.1 Approximated Sequential Multiplier

We evaluated the different error metrics for several approximate adder architectures given in [6]. Several configurations of the adders are possible depending on the application and the required error characteristics. All these adders are combinational circuits and Table 1 summarizes the error metrics computed for these circuits. The short-form notations used are as given in the repository. Both 8-bit and 16-bit versions of the adders are given in the table. The number of gates are taken from ABC [11]. Each category has a *Ripple Carry Adder* entry given at the end. This is the normal non-approximated adder and serves as golden reference model.

[3]The package and the benchmarks are available at the repository https://gitlab.com/arunc/maniac_verify.git

Table 1: Error Metrics for Approximation Adders

Architecture	gates	e_{wc}	e_{ac}	e_{er}
8-bit Adders				
Almost Correct Adder [9]				
ACA_II_N8_Q4	63	64	3.75	18.75%
ACA_I_N8_Q5	78	128	0.44	4.69%
Gracefully Degrading Adder [23]				
GDA_St_N8_M4_P2	67	64	3.75	18.75%
GDA_St_N8_M4_P4	73	64	0.19	2.34%
GDA_St_N8_M8_P1	54	168	31.50	60.16%
GDA_St_N8_M8_P2	65	144	7.75	30.08%
GDA_St_N8_M8_P3	73	128	1.88	12.50%
GDA_St_N8_M8_P4	79	128	0.44	4.69%
GDA_St_N8_M8_P5	81	128	0.09	1.56%
GDA_St_N8_M8_P6	81	128	0.02	0.39%
Accuracy Configurable Adder [15]				
GeAr_N8_R1_P1	54	168	31.50	60.17%
GeAr_N8_R1_P2	65	144	7.75	30.08%
GeAr_N8_R1_P3	73	128	1.88	12.50%
GeAr_N8_R1_P4	78	128	0.44	4.69%
GeAr_N8_R1_P5	79	128	0.09	1.56%
GeAr_N8_R1_P6	79	128	0.02	0.39%
GeAr_N8_R2_P2	63	64	3.75	18.75%
GeAr_N8_R2_P4	69	64	0.19	2.34%
Ripple Carry Adder				
RCA_N8	72	0	0	0%
16-bit Adders				
Almost Correct Adder				
ACA_II_N16_Q4	131	17472	1023.75	47.79%
ACA_II_N16_Q8	152	4096	15.94	5.86%
ACA_I_N16_Q4	161	34944	511.88	34.05%
Error Tolerant Adder [25]				
ETAII_N16_Q4	131	17472	1023.75	47.79%
ETAII_N16_Q8	152	4096	15.94	5.86%
Gracefully Degrading Adder [23]				
GDA_St_N16_M4_P4	160	4096	15.94	5.86%
GDA_St_N16_M4_P8	170	4096	0.06	0.18%
Accuracy Configurable Adder [9]				
GeAr_N16_R2_P4	149	16640	63.94	11.55%
GeAr_N16_R4_P4	152	4096	15.94	5.86%
GeAr_N16_R4_P8	158	4096	0.06	0.18%
GeAr_N16_R6_P4	155	1024	3.94	3.08%
Ripple Carry Adder				
RCA_N16	161	0	0	0%

The study of bounds of error metrics for these adder architectures by itself is interesting. A more significant information is their impact within a circuit. The isolated combinational behavior of an approximate adder could be very different from when used as a building block in a bigger (sequential) circuit. We present the case study of 8-bit sequential multipliers using these approximate adder architectures as a motivation. Although these circuits are simple to construct, their error analysis reveals the need and usefulness of our approach.

The design used in this experiment is an unsigned 8-bit sequential multiplier with 4-bit inputs. Partial products are computed in each clock cycle and added using an 8-bit adder. The number of clock cycles taken to complete the multiplication[4] is input data dependent with extra checking for the operands being 1 or 0. The adder used in the circuit is instantiated with the published approximation adders from the repository [6]. Several adders are chosen from this repos-

[4]The `OE` signal which indicates the availability of a valid output has to be additionally specified to the tool in this case.

itory with similar approximation architectures and different configurations. While the design of the non-approximated multiplier is straight forward, several interesting results can be deduced from the approximate versions.

The approximated multiplier designs are evaluated for the questions presented in Section 3.3. The results are given in the upper half of Table 2. In this table, the design details such as *Area*, *Delay*, *Gates* and *Regs* are taken from the corresponding synthesis run of ABC [11] with the command *resyn*. *Delay* is the worst timing delay achieved and *Regs* is the number of sequential elements in the design. The next four columns are the results of the *Questions* presented in Section 3.3. In Q1 and Q3, X is the error assumed and t is the corresponding clock-cycles determined by the tool. The subsequent four columns are the run-times in seconds taken to answer these questions. The last entry in the upper half of the table is the Multiplier with *Ripple Carry Adder* which is the non-approximated reference design.

Some multipliers with approximation adder architectures such as *ACA_I_N8_Q5*, *GDA_St_N8_M8_P4* etc. are equivalent to the non-approximated multiplier. In this case, the designer can safely choose any of them for the given application. This differs drastically from the error behavior of the individual adders given in Table 1.

Another interesting aspect is the maximum error rate of the circuits (result of Q4). For example, even though the multiplier using the adder *GeAr_N8_R1_P3* has a faster error accumulation (result of Q1) compared to the multiplier with adder *GeAr_N8_R2_P2* (36 cycles are less than 41 which makes it faster), the maximum error rate possible (result of Q4) is better with the former one. This is useful in designing circuits such as the one used for *error detection and correction* since such circuits rely on the error rate rather than the magnitude of the error introduced. Multipliers using the adders *GDA_St_N8_M8_P1* and *GeAr_N8_R1_P1* have the worst accumulation tendency (result of Q2) and this correlates with their combinational error behavior.

4.2 Generality and Scalability

To demonstrate generality and scalability of our proposed approach, various further benchmarks and approximation scenarios are provided in the lower half of Table 2. It has to be noted that in these examples, the approximations introduced are rather arbitrary without any proper design principles involved. The first design is a 16-bit IIR filter in which the *Multiply and Accumulate* (MAC) unit is replaced by a smaller one. The coefficients and the MAC unit of the original filter are 18-bit wide and the design is a *Direct Form-II Transposed* IIR filter. In the approximated version, the width of the MAC unit is simply reduced to 17-bit. The filter has a maximum worst-case error (result of Q2) of 31542, which is less than one half of the maximum output value of $2^{15} - 1$ (since the output represent a signed value). But it takes at least 19 cycles for the error to accumulate and cross 50,000 mark. Furthermore, the maximum number of bit-flips (not necessarily the lower bits of the output) in any given clock cycle for this design is 11.

The second design in the lower half of the Table 2 is a 17-bit output FIR filter, with approximations introduced in the values of coefficients. In this case, the error introduced is higher with accumulated error crossing 100,000 within 3 cycles of operation and at most 16 bits among the 17 bits toggle for this design in any clock cycle. Similar interpretations can be obtained from the error behavior of the other designs shown in the table such as Quantizer, Stepper Motor and Binary to BCD converter.

The results confirm the applicability of our proposed approach. The run-times heavily depend on the underlying model checking algorithms. Improvement in model checking therefore has a direct positive effect on our methodology as well.

5. CONCLUSIONS

In this paper we proposed a methodology together with tools that are able to argue precisely about the error behavior of approximated combinational components in sequential circuits. This is of high importance, because (i) there exists a large number of design and synthesis methods for approximate computing without providing precise error metrics (even in the combinational case), and (ii) the error can vary significantly when approximated components are embedded into sequential circuits.

We presented an approximation miter that can be configured in various ways to answer a variety of verification questions. These questions can be answered by solving model checking problems. We have evaluated the usefulness, effectiveness, and scalability of our approach in an experimental evaluation with realistic designs. Our methods allow for a higher quality and better possibilities for design space exploration in design for approximating computing.

Acknowledgments. This work was supported by the German Research Foundation (DFG) in the project MANIAC (DR 287/29-1), by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative, by the German Federal Ministry of Education and Research (BMBF) in the project EffektiV (01IS13022E), by H2020-ERC-2014-ADG 669354 CyberCare, and by the German Academic Exchange Service (DAAD).

6. REFERENCES

[1] A. Bradley. Incremental, inductive model checking. In *2013 20th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 5–6, 2013.

[2] M. Breuer. Determining error rate in error tolerant vlsi chips. In *Electronic Design, Test and Applications*, pages 321–326, Jan 2004.

[3] V. Chippa, S. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Design Automation Conf.*, pages 1–9, May 2013.

[4] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[5] N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *Int'l Conf. on Formal Methods in CAD*, pages 125–134, 2011.

[6] GeAr-ApproxAdderLib. Chair for Embedded Systems - Karlsruhe Institute of Technology, 2015. http://ces.itec.kit.edu/1025.php.

[7] M. Haghbayan, B. Alizadeh, P. Behnam, and S. Safari. Formal verification and debugging of array dividers with auto-correction mechanism. In *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, pages 80–85, Jan 2014.

[8] http://www.satcompetition.org/. The international sat competitions, 2014.

[9] A. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conf.*, pages 820–825, June 2012.

[10] J. Miao, A. Gerstlauer, and M. Orshansky. Approximate logic synthesis under general error magnitude and frequency con-

Table 2: Evaluation of Questions in Section 3.3 for various designs

Design Details						Questions Asked				Run-time Comparison			
Circuit	Approximation architecture used	Area*,†	Delay* (ns)	Gates*	Regs*	Q1 (X, t)	Q2	Q3 (X, t)	Q4	Q1 (sec)	Q2 (sec)	Q3 (sec)	Q4 (sec)
Multiplier (8-bit)	Almost Correct Adder												
	ACA_II_N8_Q4	507	9.60	205	39	(1000, 41)	128	(5, 11)	3	5.12	7.62	2.72	3.20
	ACA_I_N8_Q5	530	11.20	222	39	(1000, 0)	0	(3, 0)	0	14.02	11.27	7.90	4.11
	Gracefully Degrading Adder												
	GDA_St_N8_M4_P2	507	9.60	205	39	(1000, 41)	128	(5, 11)	3	3.93	9.26	1.70	3.93
	GDA_St_N8_M4_P4	540	10.30	219	39	(1000, 0)	0	(3, 0)	0	21.89	12.56	1.77	4.27
	GDA_St_N8_M8_P1	526	7.40	205	39	(1000, 26)	224	(6, 11)	4	0.97	5.22	1.00	4.14
	GDA_St_N8_M8_P2	578	8.00	239	39	(1000, 36)	144	(5, 11)	3	1.40	8.50	0.76	3.71
	GDA_St_N8_M8_P3	588	8.80	241	39	(1000, 36)	160	(4, 11)	2	2.60	7.77	2.78	5.29
	GDA_St_N8_M8_P4	569	9.40	229	39	(1000, 0)	0	(3, 0)	0	18.12	12.92	6.20	4.76
	GDA_St_N8_M8_P5	614	9.10	244	39	(1000, 0)	0	(3, 0)	0	17.91	12.24	6.13	6.22
	GDA_St_N8_M8_P6	609	10.30	248	39	(1000, 0)	0	(3, 0)	0	15.15	16.57	8.62	4.93
	Accuracy Configurable Adder												
	GeAr_N8_R1_P1	526	7.40	205	39	(1000, 26)	224	(6, 11)	4	1.67	5.56	1.52	4.72
	GeAr_N8_R1_P2	578	8.00	239	39	(1000, 36)	144	(5, 11)	3	3.50	9.68	2.32	3.68
	GeAr_N8_R1_P3	532	9.60	220	39	(1000, 36)	160	(4, 11)	2	3.23	7.92	4.02	5.32
	GeAr_N8_R1_P4	530	11.20	222	39	(1000, 0)	0	(3, 0)	0	14.03	12.69	7.89	4.80
	GeAr_N8_R1_P5	551	11.20	229	39	(1000, 0)	0	(3, 0)	0	14.39	11.78	7.93	4.75
	GeAr_N8_R1_P6	539	11.20	223	39	(1000, 0)	0	(3, 0)	0	11.39	24.02	7.42	4.26
	GeAr_N8_R2_P2	507	9.60	205	39	(1000, 41)	128	(5, 11)	3	5.14	11.43	2.72	4.65
	GeAr_N8_R2_P4	530	11.20	219	39	(1000, 0)	0	(3, 0)	0	14.64	12.64	7.32	4.28
	Ripple Carry Adder												
	RCA_N8	526	12.00	217	39	(1000, 0)	0	(3, 0)	0	0	0	0	0
IIR Filter (16-bit) ‡	Smaller MAC unit	2292	28.30	912	168	(50000, 19)	31542	(16, 33)	11	2.31	272.05	2.34	231.68
FIR Filter (17-bit) ◇	Modified coefficients	2149	38.49	827	64	(100000, 3)	65536	(16, 2)	16	1.01	15.78	6.00	2.00
Quantizer (11-bit) ±	Change in quantization value	4984	7.30	2078	568	(10000, 40)	274	(9, 5)	9	72.98	17.95	3.01	13.86
Stepper Motor (4-bit)	Counters with reduced width	1479	14.20	658	115	(100, 30)	11	(4, 18)	4	1.00	87.01	1.01	350.15
Binary to BCD (4-bit) ∓	Skipped value in conversion	829	7.60	360	61	(200, 41)	8	(4, 17)	4	3.01	12.05	1.01	2.44

Questions:

Q1: What is the earliest time (t) that I can exceed an accumulated worst-case error of X?
Q2: What is the maximum worst-case error?
Q3: What is the earliest time (t) that I can reach an accumulated error rate of X?
Q4: What is the maximum error rate?

$\star$ As reported by ABC [11] with synthesis command *resyn* and library *mcnc.genlib*
† ABC reports area normalized to *INVX1*

‡ IIR Filter is *Direct-form-II Transposed*
◇ FIR Filter parameters N=16, M=17
∓ 16-bit binary input to 5x4-bit BCD output
± Design has 8 x 11-bit outputs. Only one is verified

straints. In *International Conference on Computer-Aided Design*, pages 779–786. IEEE, 2013.

[11] A. Mishchenko, M. Case, R. Brayton, and S. Jang. Scalable and scalably-verifiable sequential synthesis. In *International Conference on Computer-Aided Design*, pages 234–241, Nov 2008.

[12] M. N. Mneimneh and K. A. Sakallah. Principles of sequential-equivalence verification. *IEEE Design & Test of Comp.*, 22(3):248–257, 2005.

[13] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan. Aslan: Synthesis of approximate sequential circuits. In *Design, Automation and Test in Europe*, pages 1–6, March 2014.

[14] H. Savoj, D. Berthelot, A. Mishchenko, and R. Brayton. Combinational techniques for sequential equivalence checking. In *Int'l Conf. on Formal Methods in CAD*, pages 145–149, 2010.

[15] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A low latency generic accuracy configurable adder. In *Design Automation Conf.*, pages 1–6, June 2015.

[16] D. Shin and S. Gupta. Approximate logic synthesis for error tolerant applications. In *Design, Automation and Test in Europe*, pages 957–960, March 2010.

[17] M. Soeken, D. Große, A. Chandrasekharan, and R. Drechsler. BDD minimization for approximate computing. In *ASP Design Automation Conf.*, pages 474–479, 2016.

[18] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation and Test in Europe*, pages 1367–1372, March 2013.

[19] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conf.*, pages 796–801, June 2012.

[20] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. MACACO: modeling and analysis of circuits for approximate computing. In *International Conference on Computer-Aided Design*, pages 667–673, 2011.

[21] C. Wolf. Yosys - Yosys Open SYnthesis Suite, 2015. http://www.clifford.at/yosys/about.html.

[22] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmaeilzadeh, and K. Bazargan. Axilog: Language support for approximate hardware design. In *Design, Automation and Test in Europe*, pages 812–817, March 2015.

[23] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On reconfiguration-oriented approximate adder design and its application. In *International Conference on Computer-Aided Design*, pages 48–54, Nov 2013.

[24] L. Zhang, M. Prasad, and M. Hsiao. Incremental deductive inductive reasoning for SAT-based bounded model checking. In *International Conference on Computer-Aided Design*, pages 502–509, 2004.

[25] N. Zhu, W. L. Goh, and K. S. Yeo. An enhanced low-power high-speed adder for error-tolerant application. In *Integrated Circuits, ISIC '09. Proceedings of the 2009 12th International Symposium on*, pages 69–72, Dec 2009.