# Chapter 9
# Automated Search-Based Functional Approximation for Digital Circuits


**Lukas Sekanina, Zdenek Vasicek, and Vojtech Mrazek**

## 9.1 Introduction

This chapter deals with an automated design method that has been developed for functional approximation of digital circuits. The method is based on an iterative technology-independent modification of a given implementation of a combinational circuit. The circuit approximation problem is, in fact, transformed to a multi-objective optimization problem and solved by means of the state-of-the-art optimization method based on genetic programming.

*Genetic programming* (GP) is an evolutionary computation technique that automatically solves design problems without requiring the user to know or specify the form or structure of the solution in advance [22]. GP evolves computer programs, traditionally represented in memory as tree structures or sequences of instructions. In order to design and optimize digital circuits, a special version of GP, Cartesian GP (CGP), has been developed and applied outside the approximate computing area [14].

There are several reasons why the CGP approach is especially useful for the circuit approximation. Existing automated circuit approximation methods are usually constructed as heuristic methods trying to provide the best tradeoff(s) between key design parameters, typically involving the error, power dissipation, and delay. CGP is an advanced search-based heuristic method which naturally provides such a multi-objective optimization scenario. In addition to providing many design alternatives, there are no restrictions in terms of constraints on target circuits (i.e., candidate circuits do not have to obey a predefined form such as, e.g., an and-inverter graph), except those specified by the user. The evaluation procedure, assessing the

L. Sekanina (✉) · Z. Vasicek · V. Mrazek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno, Czech Republic
e-mail: sekanina@fit.vutbr.cz; vasicek@fit.vutbr.cz; imrazek@fit.vutbr.cz



S. Reda, M. Shafique (eds.), *Approximate Circuits*,
https://doi.org/10.1007/978-3-319-99322-5_9

quality of candidate approximations, can contain formal verification mechanisms of selected properties in order to provide formal guarantees in terms of error or other parameters. Finally, GP can operate at the gate and register-transfer (RT) levels and it can easily be integrated into a standard circuit design flow.

An obvious disadvantage is a limited scalability of CGP because many candidate approximate circuits have to be generated and evaluated. However, recent works have shown that the scalability problem can be eliminated and CGP can provide high-quality tradeoffs between key design parameters even for complex circuits such as 32-bit approximate multipliers [1], complex approximate sorting and median networks [26], or image operators [24].

The rest of the chapter is organized as follows. Section 9.2 introduces the principles of CGP and its utilization for the purposes of circuit approximation. Three approximation strategies based on CGP are introduced. In Sect. 9.3, a special attention is given to various approaches developed for the evaluation of candidate circuits. In particular, the error calculation using simulation and formal verification techniques is presented. This is followed by introducing a light approach to estimation of circuit parameters (such as the area, delay, and power dissipation). Case studies dealing with approximate implementations of arithmetic circuits and image operators are presented in Sects. 9.4 and 9.5. Conclusions are given in Sect. 9.6.

## 9.2 Genetic Programming for Circuit Design and Approximation

After introducing the principles of GP, the rest of this section is devoted to CGP and its utilization for circuit design and approximation. In particular, three approximation strategies based on CGP are presented.

### *9.2.1 Genetic Programming*

Genetic programming, like any other evolutionary computation method, operates the so-called *population* (i.e., a set of candidate solutions). The first population is usually randomly generated, but it can also be seeded using existing designs if it is useful. This is often the case of approximate circuit evolution in which the initial population typically contains the original circuit and, if possible, its various different implementations. The candidate solutions are represented in GP in different ways, for example, as syntactic trees or sequences of instructions. An example of the circuit representation based on directed acyclic graphs will be discussed in Chap. 9.2.2.

Each solution belonging to a given population is evaluated with the so-called *objective* (or *fitness*) *function*. For example, if the objective is to minimize the error probability, the fitness function is just the error probability determined for the candidate circuit. If there are two or more fitness functions, reflecting not only the error but also the area or delay, we speak about a *multi-objective optimization*. Driven by the fitness function(s), GP performs a parallel search in the space of all possible candidate designs. New candidate circuits are generated from the current population by means of *genetic operators* such as selection, crossover, and mutation. These operators work at the level of circuit representation (see examples in Chap. 9.2.2). The process of generating new populations of candidate circuits (i.e., one run of GP, or evolution, in short) is terminated when a desired solution is discovered or the available time is exhausted.

The result of a single-objective evolution is usually only one solution; that is, the solution showing the best fitness score. A multi-objective evolutionary algorithm should produce a set of solutions showing the best tradeoffs between target objectives. We are primarily interested in the solutions belonging to the *Pareto set* which contains the so-called *non-dominated* solutions [5]. Consider three objectives to be minimized, for example, the area, the worst case error, and the delay in the case of digital circuit approximation. Circuit $C_1$ *dominates* another circuit $C_2$ if the following conditions hold:

- $C_1$ is no worse than $C_2$ in all objectives, and
- $C_1$ is strictly better than $C_2$ in at least one objective.

Modern GP methods integrate this concept of dominance into their selection mechanisms and try to find all solutions belonging to the Pareto optimal front.

## *9.2.2 Cartesian Genetic Programming*

Cartesian genetic programming grew from a method of evolving digital circuits developed by Miller et al. in 1998 [15]. CGP especially differs from other GP branches in: (1) the solution representation and (2) the search mechanism. The key ingredients of CGP are briefly introduced in the following paragraphs. Detailed description is available in [14].

### 9.2.2.1 Circuit Representation

From a hardware designer point of view, every candidate circuit is represented as a special netlist containing a constant number of components ($N$). These components are (virtually) organized in a two-dimensional grid of $n_c$ columns and $n_r$ rows ($N = n_c \cdot n_r$). The number of primary inputs and outputs is denoted $n_i$ and $n_o$. Type of components depends on the level of abstraction used in modeling, where logic gates

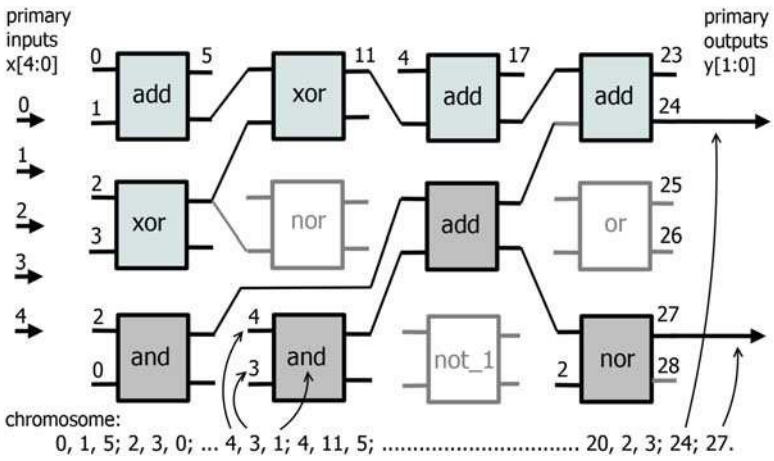



**Fig. 9.1** A combinational circuit represented in CGP with parameters: $n_i = 5$, $n_o = 2$, $n_c = 4$, $n_r = 3$, $n_a = 2$, $n_b = 2$, $\Gamma$ = {xor (encoded with 0), and (1), or (2), nor (3), not_1 (4), add (5)}. Three nodes are inactive

and RT-level components are naturally supported. Every component has up to $n_a$ inputs and $n_b$ outputs.

A unique address is assigned to all primary inputs and to the outputs of all components to define an addressing system enabling circuit topologies to be specified. The primary inputs are labeled $0 \ldots n_i - 1$ and the components' outputs are labeled $n_i, n_i + 1, \ldots, n_i + n_b \cdot n_c \cdot n_r - 1$. As no feedback connections are allowed in the basic version of CGP, only combinational circuits can be created. Figure 9.1 shows a gate-level 5-input/2-output circuit consisting of nine gates and having four logic levels on the longest input–output path. This circuit is represented in the CGP grid with $n_c = 4$ and $n_r = 3$ and the outputs of its components are labeled $5 \ldots 28$. There are 3 unused components.

Each component is represented using $n_a + 1$ integers in the netlist, where $n_a$ integers specify destination addresses for its inputs and one integer is a pointer to the table $\Gamma$ containing all supported functions. A component placed in the $j$-th column can obtain its input values either from primary inputs or from the components placed in previous columns. The whole circuit is then represented using the so-called chromosome (i.e., simplified netlist) consisting of:

$$N_g = n_c \cdot n_r \cdot (n_a + 1) + n_o \text{ integers.} \tag{9.1}$$

The last part of the chromosome contains $n_o$ integers specifying either the nodes where the primary outputs are connected to or logic constants ("0" and "1") which can directly be connected to the primary outputs.

The main feature of this encoding is that while the size of the chromosome is constant (for a given $n_o$, $n_a$, $n_r$, and $n_c$), the size of circuits represented by this chromosome is variable (from 0 to $n_c \cdot n_r$ components can be involved) as some components can remain disconnected. This redundancy has been identified as a crucial property of the efficient search in the space of digital circuits [14].

#### 9.2.2.2 Search Method

Every chromosome represents one design point in the design space. In CGP, new designs are created by introducing small random modifications to the chromosome. This operation is called the *mutation* and it typically modifies $h$ integers of the chromosome. All randomly introduced modifications must lead to valid circuits, i.e., only valid function codes and connections can be created.

Algorithm 1 presents the search method usually used for the single-objective circuit approximation by means of CGP [14]. The initial population $P$ is seeded by: (1) the original circuit ($p$), (2) alternative (accurate) implementations of $p$ (if they are available), and (3) circuits created from $p$ by mutation. The total population size is $1 + \lambda$ individuals. After evaluating the initial population, the following steps are repeated until the termination condition is not satisfied: (1) a new parent is selected, (2) $\lambda$ offspring circuits are created from the parent by means of mutation, and (3) the population is evaluated.

One mutation can affect either the component function, the component input connection, or the primary output connection. A mutation is called neutral if it does not affect the circuit's fitness. If a mutation hits a non-used part of the chromosome, it is detected and the circuit is not evaluated because it has the same fitness (i.e., quality) as its parent. Otherwise, the fitness is calculated. For further details about CGP and its extensions and parameters setting, please see [14].

**Algorithm 1:** CGP

**Input:** CGP parameters, fitness function, original circuit $p$
**Output:** The highest scored individual and its fitness

```
1 P ← CreateInitialPopulation(p);
2 EvaluatePopulation(P);
3 while ⟨terminating condition not satisfied⟩ do
4     α ← SelectHighest-scored-individual(P);
5     if fitness(α) ≥ fitness(p) then
6         p ← α;
7     P ← {p} ∪ {λ offspring of p created by mutation};
8     EvaluatePopulation(P);
9 return p, fitness(p);
```

### 9.2.3 CGP in Circuit Approximation

CGP can evolve high-quality implementations of digital circuits from scratch, only on the basis of a behavioral description provided [14, 33]. CGP can also be employed to optimize existing designs, for example, to reduce the number of gates [28]. In the context of approximate computing, three approximation strategies have been developed.

#### 9.2.3.1 Resources-Oriented Method

Let us suppose that $M$ is the minimum number of components (gates) that are needed in order to construct a fully functional circuit. In the first approximation strategy, CGP is used to minimize the error criterion under the assumption that only $m_i$ components (gates) are available and $m_i < M$. This can be easily achieved when CGP is intentionally employed with insufficient resources ($m_i = n_c \cdot n_r$). In order to obtain different tradeoffs between the error and the number of components, CGP is executed several times with different $m_i$ as the parameter. The main advantage is that the user can control the used area (and power consumption) precisely by means of $m_i$. The method was employed to approximate small multipliers and 9-input and 25-input median circuits operating over 8 bits [30].

#### 9.2.3.2 Error-Oriented Method

In the error-oriented method, the target error level $e_{\max}$ (e.g., the average error magnitude) is specified by the user. Two different error-oriented approaches have been developed. In both cases, CGP is initialized with a precise implementation and employed (with sufficient resources) to modify the initial implementation to exhibit the target error $e_{\max}$ provided that the number of components is reduced as much as possible. If various tradeoffs between the error and the number of components are requested, CGP is executed several times with $e_{\max}$ as the parameter. The error-oriented approach tends to be less computationally demanding than the resources-oriented method.

The first approach is based on a two-phase design procedure and represents a natural way how to perform approximation of digital circuits. The goal of the first phase is to modify the initial implementation to exhibit the target error $e_{\max}$. After obtaining a circuit satisfying this requirement, the fitness function is changed. The objective is now to minimize the number of components (or another criterion) providing that $e_{\max}$ is left unchanged (is within a predefined interval). The two-stage error-oriented method was applied to design various adders and multipliers [23, 29].

Another option is to employ a single-phase CGP where the target error serves as a constraint. The goal of CGP is to minimize the number of components providing that the error is not worse than $e_{\max}$. As the search is forced towards more compact

solutions, the error is implicitly forced to be as close as possible to the target error value. By means of the single-phase error-oriented method, approximate multipliers showing specific properties were evolved for artificial neural networks implemented on a chip [18].

#### 9.2.3.3 Multi-Objective CGP

In the multi-objective method, the error and other key circuit parameters (area, delay, and power consumption) are optimized together by a multi-objective CGP [19]. The multi-objective CGP represents candidate circuits using CGP encoding. The new candidate circuits are created by means of a point mutation operator. The search is not conducted by Algorithm 1, but a multi-objective extension of CGP has to be taken. Mrazek et al. [19], for example, used a modified variant of Non-dominated Sorting Genetic Algorithm (NSGA-II). NSGA-II sorts individuals according to the dominance relation into multiple fronts. The first front contains all non-dominated solutions along the Pareto front. Each subsequent front is constructed by removing all the preceding fronts from the population and finding a new Pareto front.

The multi-objective CGP is the most promising approach because it reconstructs the Pareto front in each CGP generation and tries to cover all possible compromise solutions. However, in real-world applications, we are typically interested in only several (predefined) design targets; for example, approximate implementations have to be developed for a few error levels known in advance. Then, it is usually computationally less expensive to execute a single-objective CGP optimizing a given parameter several times and having the remaining ones as the constraints.

### *9.2.4 Properties of the CGP-Based Approximation Method*

There are no constraints imposed on circuits that can be obtained by means of CGP except those specified by the user. A suitable setting of CGP thus enables to constrain the size and maximum delay of all candidate circuits and restrict the set of supported functions, which is useful when one needs, for example, to avoid using certain gates in approximate circuits because they are expensive (such as exclusive-ORs). If there is a specific requirement, for example, in the case that an approximate multiplier is evolved, but the exact result is requested for some predefined subset of inputs (see [18]), the fitness function will consider this requirement. If a candidate circuit satisfies this requirement, it can be evaluated in terms of error and other parameters. Otherwise, the worst possible score is immediately assigned to that circuit.

CGP is known for its high computational requirements because many candidate circuits (often in the order of millions) have to be generated and evaluated in a single run. However, with widely available parallel computer clusters and fast fitness

evaluation based on parallel simulation and formal methods, CGP is now highly competitive even when circuits such as 32-bit multipliers are approximated [1].

The CGP-based approximation can easily be integrated into a conventional design flow. For example, when a gate-level approximation is conducted, CGP starts with the netlist representing the original (exact) circuit and outputs another netlist representing the approximate circuit. The resulting netlist is then used in the standard design flow.

### *9.2.5 Other Approximation Methods Based on Evolutionary Computation*

Except the CGP-based methods, evolutionary computation has only infrequently been used in approximate circuit design. For example, Lotfi et al. performed the sensitivity analysis to find safe-to-approximate variables in OpenCL kernels [12]. The objective was to optimize the precision of these variables by means of the genetic algorithm with the aim of minimizing the resource utilization on FPGA while meeting the target quality. Nepal et al. presented ABACUS method introducing approximate operations on abstract syntax trees representing behavioral register-transfer level descriptions of digital circuits. They applied a multi-objective approximations conducted by means of a non-dominated sorting genetic algorithm (NSGA II) [21].

## 9.3 Evaluation of Candidate Designs

In each iteration, it is necessary to evaluate to what extent a given candidate approximate circuit satisfies functional and nonfunctional requirements imposed by the specification. While there are common approaches on how to evaluate the electrical parameters (area, delay, and power consumption), determining the quality of a candidate approximation is in general a nontrivial problem. The evaluation in a target application is typically time consuming. For the search-based synthesis, it is crucial, however, to perform the checking as quickly as possible because this procedure is employed in iterative design process. Hence, an alternative approach that does not require a direct interaction with a target application is typically employed. Typically, an error metric is used to assess the quality of a given approximation. Such a metric should be carefully chosen so that it reflects the performance of a given approximation considering the target application. For small problem instances, exhaustive simulation represents a viable option because the current CPUs enable to evaluate up to 256 input combinations in parallel. For more complex instances, we can adopt techniques of formal equivalence checking. But, the nature of the approximate circuits involves to replace the strict formal

equivalence checking with more advanced methods that enable to perform the so-called *relaxed equivalence checking*, i.e., checking that two circuit designs are equal up to some bound. Compared to the formal equivalence checking, only little has been done in this area and the relaxed equivalence checking still represents an open and challenging problem.

### *9.3.1 Quality of Approximate Circuits*

The functionality of approximate circuits is typically expressed using one or several error metrics. In addition to the error rate, the average-case as well as the worst-case situation can be analyzed. Among others, mean absolute error (MAE) and mean square error (MSE) are the most familiar metrics that are based on the average-case analysis. Selection of the right metrics is a key step of the whole design. When an arithmetic circuit is approximated, for example, it is necessary to base the error quantification on an arithmetic error metric since the error magnitude could have a significant impact on target application. For general logic, where no additional knowledge is available and where there is not a well-accepted error model, Hamming distance or error rate is typically employed.

Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be an $n$-input $m$-output Boolean function that describes correct functionality (specification) and $\hat{f} : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be an approximation of it, both implemented by two circuits, namely F and $\hat{\text{F}}$. The following paragraphs summarize the error metrics that have been employed in the literature to quantify the deviation between the outputs produced by a functionally correct design and an approximate design.

#### 9.3.1.1 Arithmetic Error Metrics

The *worst-case arithmetic error*, sometimes denoted as *error magnitude* or *error significance* [2], is defined as:

$$e_{wce}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} |\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))| \tag{9.2}$$

where $\text{nat}(x)$ represents a function $\text{nat} : \mathbb{B}^m \rightarrow \mathbb{Z}$ returning a decimal value of the $m$-bit binary vector $x$. Typically, a natural binary representation is considered, i.e., $\text{nat}(x) = \sum_{i=0}^{m-1} 2^i x_i$. The worst-case error represents the fundamental metric that is typically used as a design constraint that helps to guarantee that the approximate output can differ from the correct output by at most $\epsilon$ (i.e., the condition $e_{wst}(f, \hat{f}) \leq \epsilon$ is satisfied during the whole design process).

Rather than the absolute worst-case error, *relative worst-case error* is employed

$$e_{wcre}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} \frac{|\,\mathrm{nat}(f(x)) - \mathrm{nat}(\hat{f}(x))|}{\mathrm{nat}(f(x))} \tag{9.3}$$

to constrain the approximate circuit to differ from the correct one by at most a certain margin. Note that a special care must be devoted to the cases for which the output value of the original circuit is equal to zero, i.e., the cases when the denominator approaches zero. This issue can be addressed by either omitting test cases when $nat(f(x)) = 0$ or biasing the denominator as employed in [19].

The *average-case arithmetic error* (*mean absolute error*) is defined as the sum of absolute differences in magnitude between the original and approximate circuits, averaged over all inputs:

$$e_{mae}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} |\,\mathrm{nat}(f(x)) - \mathrm{nat}(\hat{f}(x))| \tag{9.4}$$

When we replace the expression in the sum by the equation for relative error distance, we can calculate the *mean relative error*:

$$e_{mre}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} \frac{|\,\mathrm{nat}(f(x)) - \mathrm{nat}(\hat{f}(x))|}{\mathrm{nat}(f(x))}. \tag{9.5}$$

#### 9.3.1.2 General Error Metrics

In addition to the arithmetic error metrics, there are metrics that are not related to the magnitude of the output of the correct or approximate circuit.

*Error rate* referred to as *error probability* represents the basic measure that is defined as the percentage of inputs vectors for which the output value differs from the original one:

$$e_{prob}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} [f(x) \neq \hat{f}(x)] \tag{9.6}$$

In many cases, it is worth to consider also the Hamming distance between $f(x)$ and $\hat{f}(x)$. The *worst-case Hamming distance* denoted also as *bit-flip error* [4] is defined as:

$$e_{bf}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} \left( \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \right) \tag{9.7}$$

and gives the maximum number of output bits that simultaneously outputs a wrong value.

The average number of changed output bits denoted as *average Hamming distance* can be expressed as follows:

$$e_{mhd}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in B^n} \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \tag{9.8}$$

Note that $e_{prob}(f, \hat{f}) = e_{mhd}(f, \hat{f})$ when applied to single-output functions, i.e., when $m = 1$.

#### 9.3.1.3 Problem-Specific Error Metrics

In some cases, neither the common metrics (e.g., error rate) nor the arithmetic metrics provide a satisfactory assessment of the quality of approximate circuits. Hence, various problem-specific error metrics have been introduced. For example, *distance error* was proposed to evaluate the quality of approximate median and sorting circuits [17, 26]. The common problem of the previously mentioned metrics is that they are data dependent. To model the error introduced by the approximations of median and sorting networks, the authors proposed to measure the distance between the rank of the returned element and the rank expected by the specification.

Two additional metrics can be inferred from the distance error: *average distance error* defined as the sum of error distances averaged over all input combinations producing an invalid output value and *worst case distance error* defined as the maximal distance error calculated over all the input combinations.

Chandrasekharan et al. [3] analyzed the behavior in sequential circuits that contain approximate combinational components. Although the worst case can be computed for the approximate component in isolation, the accumulated worst case in the sequential circuit may differ significantly [3]. The sequence of successive input patterns for the approximate component depends on the sequential logic and composition of the overall circuit. Hence, *accumulated worst-case error* and *accumulated error rate* have been introduced.

### *9.3.2 Error Analysis Based on Simulation*

In order to evaluate the quality of the approximate circuits, a common approach is to employ a circuit simulator that calculates responses for all input vectors. This step involves the interpretation of a CGP genotype for each vector. One of the key features of CGP encoding is that it can directly be used as an intermediate code that is processed by an interpreter [32]. To maximize efficiency of the interpreter, a common approach for the gate-level CGP is to employ a bit-level parallel simulation [14]. The idea of parallel simulation is to utilize bitwise operators operating on multiple bits in a high-level language (such as C) to perform more

than one evaluation of a gate in a single step. This approach benefits from the fact that the modern processors are equipped with specialized SIMD instructions. For example, the widely available Advanced Vector Extension (AVX) instruction set allows us to operate with 256-bit operands. It means that every circuit with eight inputs can completely be simulated in one pass by applying a single 256-bit test vector at each input. Therefore, the obtained speedup is 256 against the sequential simulation. When more complex circuits need to be evaluated, multiple 256-bit vectors are applied sequentially. In general, the obtained speedup is $w$ on a $w$-bit processor (assuming $2^n \geq w$). In practice, the speedup typically varies depending on the number of CGP nodes due to the overhead introduced by the interpreter itself. As shown in [32], the performance of the simulator can be substantially improved if the interpreter is avoided and replaced by a native machine code that directly calculates the responses. Despite of that, the number of input combinations grows exponentially with respect to the number of primary inputs.

For example, the exhaustive simulation of a circuit having 32 inputs and 1500 gates takes about 5 min using an interpreted 256-bit parallel simulator executed on a Xeon CPU operating at 2.6 GHz. From a practical point of view, the error analysis method whose runtime requires more than few seconds is unattractive for the search-based synthesis because it leads to enormous overall runtime. Hence, an alternative and more scalable technique is requested. Many authors simplify the problem and evaluate the functionality of approximate circuits by applying a subset of the set of all input vectors. Monte Carlo simulation is typically utilized to measure the error of the output vectors with respect to the original solution [9, 20, 35]. The number of randomly generated vectors required for a given confidence level is determined ad hoc or analytically by means of an equation which reflects the number of primary inputs, confidence level, and the margin of error [36]. Unfortunately, this approach provides no guarantee on the error and makes it difficult to predict the behavior of an approximate circuit under conditions different from those used during simulation.

### *9.3.3 Formal Approaches in Error Analysis*

In order to overcome limitations of the simulation, various formal approaches can be employed [25]. Determining whether two Boolean functions are functionally equivalent represents a fundamental problem in formal verification. Although the functional equivalence checking is an NP-complete problem, several approaches have been proposed so far to reduce the computational requirement for practical circuit instances. State-of-the-art verification tools are based on Reduced Ordered Binary Decision Diagrams (ROBDD) and satisfiability (SAT) solvers. ROBDDs have been traditionally used to solve the equivalence checking problem due to their canonical property. The decision procedure is trivial and reduces to pointer comparison. However, it is the requirement for canonicity that makes ROBDDs inefficient in representing certain classes of functions. It is a well-known fact that the size of BDD is sensitive to the chosen ordering of the variables and the variable

ordering should not be chosen randomly [25]. There are functions whose BDD size is always polynomial in the number of input variables (e.g., symmetric functions). On the other hand, there are functions for which the BDD size is always exponential, independent of variable ordering. It has been proven that not only multipliers but also integer division, remainder, square root, and reciprocal exhibit exponential memory requirements for any variable ordering [25].

Currently, the SAT solver-based (or simply SAT-based[1]) equivalence checking represents a method of the first choice. Modern SAT algorithms are very effective at coping with large problem instances and large search spaces [25]. The basic principle is to translate the problem of functional equivalence of two combinational circuits to the problem of deciding whether a Boolean formula given in conjunctive normal form (CNF) is satisfiable or not. This can be done using a miter which contains the combinational circuits whose corresponding outputs are connected via XOR gates and whose outputs are feed into a single OR gate. To prove functional equivalence, it is necessary to prove that the output of the miter (i.e., the OR gate) is always false.

Most formal verification approaches that employ testing exact equivalence are not directly extendable for relaxed equivalence checking; however, the ideas behind efficient testing of exact equivalence can serve as a basis for developing efficient methods for checking relaxed equivalence. A common approach to error analysis is to construct an auxiliary circuit referred to as approximation miter. This circuit instantiates both the candidate approximate circuit and the accurate (reference) circuit and compares their outputs to quantify the error. The comparison is typically ensured by means of an error computation block. The structure of the approximation miter is shown in Fig. 9.2. For arithmetic error metrics, a two's complement subtractor followed by a circuit which determines absolute value is employed. Such a block determines the absolute difference as requested by the equations defined in Sect. 9.3.1.1. For Hamming distance and error rate, XOR gates connecting the corresponding outputs are sufficient. If we want to prove whether the error is bounded by some constant, the output of the error computation block is feed into a decision circuit which compares the error with a predefined bound.

For computing the worst-case error, SAT-based solver can be employed. The approximation miter is converted to a CNF formula and the resulting formula is used together with an objective function as input of the SAT solver. Worst-case error analysis is typically based on an iterative approach. Usually, a variant of binary search is applied. It starts with the most significant bit and gradually determines the exact value of each bit. A much simpler task is to check whether a predefined worst-case error is violated by the candidate approximate circuit. The common SAT-based error checking performs much faster than the SAT-based worst-case error analysis since it does not require iterative processing. CNF of the approximate

[1]Note that the SAT problem can be solved using a solver based on ROBDDs. By a SAT-based solver, we mean a variant of SAT algorithm typically based on DPLL backtracking operating at the level of CNF.

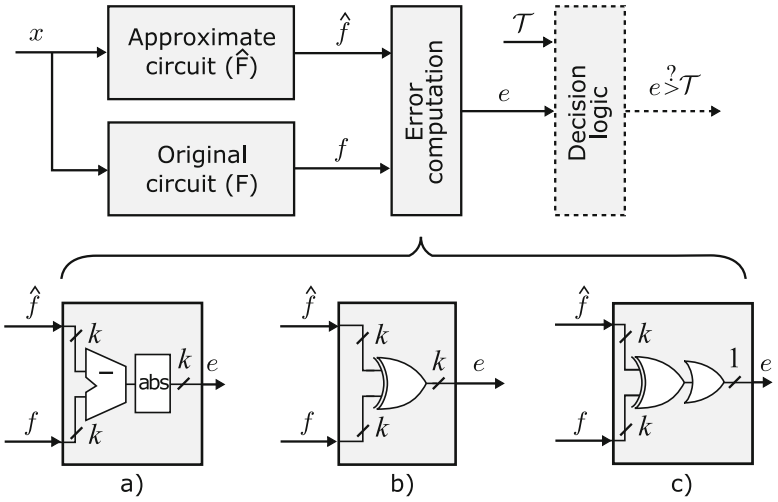



**Fig. 9.2** Approximation miter for the average-case and worst-case error analysis. Implementation of the error computation block for: (**a**) arithmetic error, (**b**) Hamming distance, and (**c**) error rate

miter is submitted to a SAT solver which gives us the answer. Although the SAT-based combinational equivalence checking performs poorly for some problem instances, e.g., for multipliers, the SAT-based error checking was used to design 32-bit approximate multipliers and 128-bit adders providing high-quality trade-off between the worst-case arithmetic error and area [1]. The authors modified CGP to drive the search towards promptly verifiable approximate circuits. The key idea was to introduce a hard time-limit for a chosen SAT solver which causes that all candidate circuits violating this limit are discarded.

While violating the worst error can be detected, no practically useful method capable of establishing the average-case error, error rate, and total Hamming distance using a SAT-based solver has been proposed up to now. The common feature of these metrics is that it is necessary to determine the number of input assignments that evaluates output of an approximation miter to true. This problem generalizes the SAT problem and is known as the model counting problem or simply #SAT. The model counting represents a challenging problem since it has been demonstrated that #SAT is extremely hard even for some polynomial-time solvable problems [25]. As a consequence of that, the available #SAT solvers are able to handle only small instances in reasonable time.

The ROBDDs seem to be the only viable option how to calculate this type of error metrics. One of the main advantages of ROBDDs is the possibility to efficiently perform many of the operations needed for the manipulation of Boolean functions. For example, the ROBDDs enable to efficiently determine the number of satisfying assignments. This can be done in linear time with respect to the number of BDD nodes by calling the *SATcount* operation.

The Hamming distance computed using BDDs was introduced in [31] in the context of CGP-based approximation of general logic. The computation of the average-case Hamming distance is relatively straightforward (see Eq. (9.9)). The average-case Hamming distance can be obtained by converting the miter (shown in Fig. 9.2b) to corresponding ROBDD and calling SATcount operation for each XOR gate. Finally, we sum the obtained results and divide them by the total number of input assignments.

$$e_{mhd}(f,\hat{f}) = \frac{1}{2^n}\sum_{\forall x\in B^n}\left(\sum_{i=0}^{m-1} f_i(x)\oplus\hat{f}_i(x)\right) = \frac{1}{2^n}\sum_{i=0}^{m-1}\left(\sum_{\forall x\in B^n} f_i(x)\oplus\hat{f}_i(x)\right)$$
$$= \frac{1}{2^n}\sum_{i=0}^{m-1}\text{SATcount}(f_i\oplus\hat{f}_i). \tag{9.9}$$

A similar approach can be employed to determine error rate (see Eq. (9.10)). The error rate is defined as the percentage of input vectors for which the approximate output differs from the original one. The output is classified as invalid even if only one bit is different. It means that it is sufficient to apply SATcount operation on the output of a common miter (shown in Fig. 9.2c) as the miter is constructed in such a way that it evaluates to true if and only if a certain input assignment yields an invalid response.

$$e_{prob}(f,\hat{f}) = \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n}[f(x)\neq\hat{f}(x)] = \frac{1}{2^n}\sum_{\forall x\in B^n}\left(\bigvee_{0\le i<m} f_i(x)\oplus\hat{f}_i(x)\right)$$
$$= \frac{1}{2^n}\text{SATcount}\left(\bigvee_{0\le i<m} f_i\oplus\hat{f}_i\right) \tag{9.10}$$

Recently, a BDD-based method for arithmetic worst-case and average-case error analysis was developed [34]. The computation of the average-case arithmetic error using BDDs is derived in Eq. (9.11). In order to determine the average-case error, we can create an auxiliary miter consisting of the combinational circuits whose outputs are feed into subtractor followed by a circuit which computes the absolute value (shown in Fig. 9.2a). The average-case arithmetic error can be then obtained by several calls of SATcount operation, one per each bit of the circuit producing the absolute value. The obtained numbers are weighted by appropriate powers of two and summed up. The average-case arithmetic error can be determined as:

$$e_{mae}(f,\hat{f}) = \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n} e_{f,\hat{f}}(x) = \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n}\left(\sum_{i=0}^{m-1} E_i(x)\cdot 2^i\right)$$

**Table 9.1** The worst recorded time needed to perform error analysis of approx. unsigned adders

| Approach | 16-bit adder | | | | 20-bit adder | | | | 32-bit adder | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $e_{prob}$ | $e_{wce}$ | $e_{mae}$ | $e_{mre}$ | $e_{prob}$ | $e_{wce}$ | $e_{mae}$ | $e_{mre}$ | $e_{prob}$ | $e_{wce}$ | $e_{mae}$ | $e_{mre}$ |
| Parallel simulation | 40 s | 41 s | 42 s | 46 s | 386 s | 390 s | 392 s | 408 s | n/a | n/a | n/a | n/a |
| BDD | 1 us | 1 ms | 1 ms | n/a | 17 us | 7 ms | 9 ms | n/a | 0.2 ms | 9 ms | 18 ms | n/a |
| SAT | n/a | 2 ms | n/a | n/a | n/a | 4 ms | n/a | n/a | n/a | 7 ms | n/a | n/a |

$$= \frac{1}{2^n} \sum_{i=0}^{m-1} \left( 2^i \sum_{\forall x \in \mathbb{B}^n} E_i(x) \right) = \sum_{i=0}^{m-1} 2^{i-n} \cdot \text{SATcount}(E_i), \tag{9.11}$$

where $e_{f,\hat{f}}(x) = |\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))|$ denotes the absolute error expressed using $m$ bits and $E(x) = \text{nat}^{-1}(e_{f,\hat{f}}(x))$ is the binary expansion of $e_{f,\hat{f}}(x)$.

### 9.3.4 A Brief Comparison of the Error Analysis Methods

To make a picture about the efficiency of the various approaches, we applied the 256-bit parallel simulation, BDD-based approach, and SAT-based approach on a set of $5 \times 10^6$ randomly generated approximate adders and measured the runtime required to perform error analysis. To determine the runtime, 16-bit, 20-bit, and 32-bit approximate adders (i.e., circuits with 32, 40, and 64 inputs, respectively) were chosen as a benchmark. Because the time varies with the actual error (especially in the case of the SAT-based approach), the worst recorded runtime is reported in Table 9.1. The measurement was conducted at Intel XEON E5-2670 running at 2.6 GHz. In the case of SAT, the binary search strategy was used to determine the exact value of $e_{wce}$.

The parallel simulation represents a universal method which is able to calculate virtually every error metric but its bad scalability is the main limiting factor. The BDD-based error analysis is a very efficient approach which scales very well on non-pathological cases of circuits, but it cannot be applied to calculate arbitrary error, e.g., the relative error requires division which is hard to implement. The SAT-based method also scales very well, but it is applicable to the worst-case error analysis only.

### 9.3.5 Electrical Parameters

Since the typical goal is to obtain energy-efficient approximate circuits, it is necessary to integrate this requirement into the fitness. Unfortunately, we cannot

afford to run a time-consuming synthesis by means of a professional design tool for each generated candidate circuit. Hence, the electrical parameters are typically estimated during the evolution. The most elementary approach is to base the search-based synthesis on the optimization of the number of gates or area on the chip. The reasons are as follows. Firstly, the power consumption of combinational circuits is usually highly correlated with these parameters [30]. Secondly, this method does not introduce any overhead since both parameters can easily be determined in linear time with respect to the number of gates of a candidate circuit. This simplification, however, may lead to unsatisfactory results especially for circuits consisting of few gates exhibiting high switching activity. For this reason, Hrbacek et al. employed a more precise power estimation technique based on the switching activity [6]. In general, the power consumption of a digital circuit consists of dynamic and static power component. Hrbacek et al. estimated the dynamic part according to the total load capacitance of the output and switching activity. Because the static part of the power consumption only depends on a function of the logic gate, the total static power consumption was based on summing static leakage for all gates of the candidate circuits. In addition to the power, the authors also considered delay that was calculated as a delay of the longest path. Delay of a gate and its leakage power and capacitance of its inputs were determined according to the liberty file available for the given semiconductor technology.

## 9.4 Case Study 1: Approximate Arithmetic Circuits

The first case study is devoted to the automated design of approximate arithmetic circuits. We restrict our attention to approximation of two arithmetic circuits—multipliers and adders. These circuits play a prominent role in practice since they are extensively used in many real-world systems for machine learning as well as signal processing. Not only the efficient implementation of digital filters (e.g., FIR, and IIR) and various transformations (e.g., DCT, and DFT) but also artificial neural networks, for example, profit from the extensive usage of adders and multipliers. Those applications are typically built on top of the familiar multiply-and-accumulate principle and exhibit a high degree of error resilience. Unfortunately, multipliers—due to their complex structure—represent one of the most difficult arithmetic circuits from the perspective of both approximation and verification. Despite of that, we show how to approximate even complex problem instances such as 32-bit multipliers without sacrificing the guarantee on the error.

### *9.4.1 Library of 8-Bit Approximate Adders and Multipliers*

This chapter deals with the designing of EvoApprox8b library which is a rich and well-focused publicly available library of approximate components that can be

immediately used in various applications or for benchmarking of circuit approximation methods. It contains hundreds of 8-bit alternative implementations of approximate adders and multipliers.

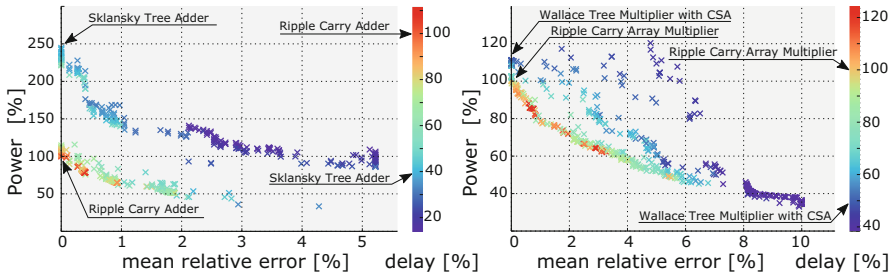
In order to develop the library, the authors employed a general-purpose approximation method based on the multi-objective CGP (see Sect. 9.2.3.3). CGP operates at the level of standard cells such as one bit adder and two and three input gates. The authors used a subset of components from a 180-nm technology library. The cells have one, two, or three inputs (e.g., full adder) and one or two outputs depending on the function. Some of the functions (e.g., BUF, and INV) have multiple sizes which differ in the maximum output load, area, power consumption, and delay. During the evaluation, a proper size is selected depending on the output load of the gate.

The goal is to simultaneously minimize delay, power consumption, and error to discover a set of approximate circuits along a Pareto front. The error is determined using an exhaustive simulation. The evaluation of the fitness consists of two steps. At the beginning, active nodes are identified in the CGP chromosome. Then, only the active nodes are evaluated by means of a 256-bit parallel circuit simulator. Power consumption is estimated by means of the switching activity analysis. The delay of a candidate circuit is calculated as the sum of the delays on the cells along the longest path. The delay of a cell is modeled as a function of its input transition time and capacitive load on the output of the cell.

The CGP parameters are set as follows: $h = 5\%$, $n_a = 3$, $n_b = 2$, $n_r = 1$, $\lambda = 500$ individuals in the population, $5 \times 10^6$ generations. The number of columns is $n_c = 200$ in the case of the adders and $n_c = 1000$ in the case of the multipliers. The set of functions $\Gamma$ includes 15 functions that reflect common standard cells available in technology libraries. In addition to the common two-input gates, the following cells are selected: NAND3 (3-input NAND), NOR3 (3-input NOR), MUX2 (2-to-1 multiplexer), AOI21 (3-input AND/NOR), OAI21 (3-input OR/NAND), FA (full adder), and HA (half adder).

In order to avoid a possible bias preventing discovery of some implementations, the initial population was seeded by various conventional implementations. The authors utilized 13 different adders and 6 different multipliers ranging from the basic implementations such as Ripple-Carry Adder or Ripple-Carry Array Multiplier to advanced architectures such as Higher Valency Tree Adder and Wallace Tree Multiplier. These accurate circuits were described in Verilog language and synthesized using Synopsys DC and 180 nm technology. The resulting netlists were directly converted to the corresponding CGP netlists and used as the members of the initial population. The goal of the evolution was to design approximate adders and multipliers having MRE less than 10%. In addition to that, the search was constrained so that $e_{wce}$ could be at most 5% of the output range and $e_{wcre}$ could not be worse than 1000%. All candidate solutions violating these requirements were discarded.

Figure 9.3a shows parameters of the evolved approximate adders. In total, 430 Pareto optimal 8-bit approximate adders were obtained from the initial population of 13 conventional adders. The electrical parameters (power and delay) are expressed relatively to the Ripple-Carry adder (RCA) which is considered as 100% in the




(a) 8-bit adders (b) 8-bit multipliers

**Fig. 9.3** Pareto fronts of the evolved 8-bit approximate adders and multipliers. (**a**) 8-bit adders. (**b**) 8-bit multipliers

figure. RCA is optimal in terms of power consumption among the conventional architectures, but significant savings can be achieved when relaxing the requirement of perfect functionality (see the orange and green points). On the other hand, the Tree Adder with Sklansky architecture exhibits best performance considering the maximum operating frequency. Two times higher area on the chip and 2.5 times higher power consumption is the cost we have to pay when we need to reduce delay by 40%. There are approximate adders that exhibit the same or even better delay but having substantially lower power consumption (see the purple and blue points). The best evolved low-power adders do not have MRE higher than 3%.

Figure 9.3b shows parameters of the 471 Pareto optimal 8-bit approximate multipliers that were evolved from 6 conventional implementations. The electrical parameters are related to the Ripple-Carry Array Multiplier (RCAM) architecture which represents 100%. RCAM architecture is the most efficient exact multiplier when considering the power. Wallace Tree multipliers, however, enable us to reduce the delay to 48% of RCAM at the cost of a slightly higher power consumption and area. The overhead is relatively low. The power increases by 22% and the area by 22%. Since the 8-bit multiplier is a more complex circuit compared to the 8-bit adder, the power consumption as well as delay decreases with increasing MRE. For 10% MRE, the approximate multipliers exhibit 60% power reduction. The results suggest that even higher reduction can be achieved if larger error is allowed.

To evaluate the performance of the proposed multipliers, we synthesized the evolved netlists using Synopsys DC for 45 nm technology. In addition to that, we implemented 58 approximate multipliers created by means of two analytical methods that are considered as state-of-the-art solutions (see, e.g., the comparative evaluation [9]). We applied the common truncation (bit-width reduction) technique as well as a more advanced truncation scheme proposed by Mahdiani et al. [13] to the Array multiplier consisting of Carry-Save adders. The common truncation is rather limited—only 7 instances can be implemented over 8 bits. The remaining cases are the result of advanced truncation. For comparison, we also implemented the 8-bit multiplier built using 2-bit approximate multipliers as proposed by

Kulkarni et al. [11]. Kulkarni's multiplier is one of the first approximate multipliers. The parameters of the synthesized circuits are summarized in Fig. 9.4. Despite of the fact that the EvoApprox8b library was optimized for 180 nm, the multipliers perform well even at 45 nm. Kulkarni's multiplier is extremely inefficient. Even the truncated multipliers exhibit significantly better parameters for the same MRE. Note that the MAE of the Kulkarni's multiplier is 1.54% which is out of the shown range. Considering MRE, MAE but also WCE and WCRE, EvoApprox8b designs outperform the other approaches. Moreover, they fill the missing spaces that are unreachable by truncation.

### 9.4.2 *Approximating Complex Arithmetic Circuits*

Highly competitive 8-bit approximate circuits were obtained by means of the multi-objective CGP, but there may be applications that require larger bit-widths and higher dynamic range. One possibility is to use the approximate 8-bit components as building blocks of more complex approximate adders and multipliers as demonstrated, for example, in [11]. The recursive construction of larger approximate multipliers from smaller approximate building blocks unfortunately does not provide the optimal results. The parameters of the obtained circuits are far from the optimum especially if the building blocks are small. This is illustrated in Fig. 9.4. The approximate circuits designed for a target bit-width exhibit substantially better parameters.

In order to approximate complex arithmetic circuits, Ceska et al. proposed a method which integrates a SAT-based approach for approximate equivalence checking into the CGP-based circuit optimization algorithm [1]. The principle is as follows. The goal is to gradually modify the original circuit and obtain an approximate circuit whose WCE is as close as possible to the predefined value and whose size represented by the number of gates is as small as possible. A single-

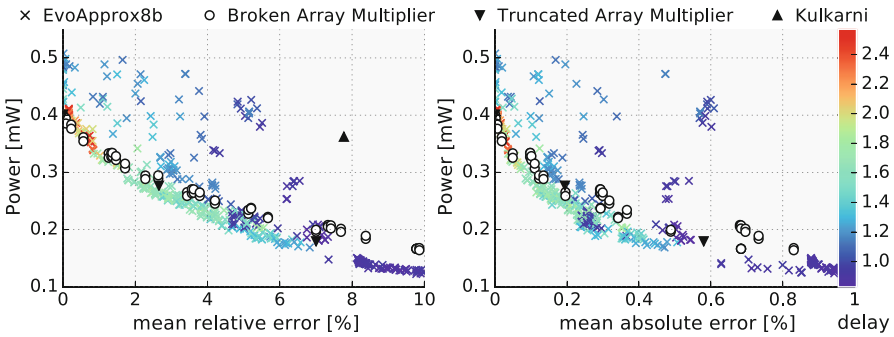



**Fig. 9.4** Parameters of 8-bit multipliers synthesized using 45 nm technology with $V_{cc} = 1$ V

objective CGP is employed. The set of various trade-offs between WCE and other parameters is obtained when we execute this algorithm several times with different target error levels as the parameter.

In order to avoid calculation of the exact value of WCE which may be time consuming especially for hard instances such as multipliers, the authors formulated the problem as follows. As proposed by Mrazek et al. in [18], the predefined error level is used as the constraint instead of being considered directly in the fitness function. If a candidate solution violates this constraint, it is discarded. One of the properties of the approximate circuits is that the error increases with increasing the number of removed gates. If we consider this fact and construct the fitness function to force the search towards more compact solutions (circuits having the minimal possible number of gates for a given target error level), WCE of the candidate circuits will be implicitly forced to be as close as possible to the target value.

To compute whether the WCE is violated, the concept of approximation miter illustrated in Fig. 9.2a is adopted. Instead of the direct usage of this scheme, an alternative approach is proposed which utilizes the fact that the target error level is a constant value. This observation enables to avoid a circuit determining the absolute value which leads to long carry chains. The optimized miter consists of the inspected approximate candidate circuit, the original circuit, a subtractor for error computation, and a decision block which checks whether the error introduced by the approximation is greater than a given threshold. The decision block is implemented as a comparator which performs one of two comparisons depending on the sign at the output of the subtractor, i.e., a comparison for either positive or negative difference is executed. The key feature of the comparator is that it can be implemented using simple AND/OR gates. The second trick introduced in [1] is a limit for the SAT solver which helps to avoid excessive run times. The SAT solver is given a predefined amount of time that can be used to decide whether the resulting CNF is satisfiable or not. When the runtime exceeds this limit, the SAT solver is terminated and the corresponding candidate solution is discarded. This strategy drives the search towards promptly verifiable candidate solutions and thus provides scalable approximation of complex circuits.

The approach is implemented as a new module in state-of-the-art academia synthesis tool ABC. The scalability is evaluated on large adders and multipliers. The most complex circuits are 32-bit multiplier and 128-bit adder. The CGP parameters are set as follows: $h = 5$, $n_a = 2$, $n_b = 1$, $n_r = 1$, $\lambda = 5$. The Array multiplier and Ripple Carry adder implemented using 2-input gates are employed as the initial solution for CGP. $n_c$ corresponded with the number of gates. The set of functions counted six common two-input logic gates, the buffer, and the inverter. CGP was executed for 2 h for adders and up to 6 h for 32-bit multipliers. The fitness function reflects only the area estimated as the sum of the relative areas of active gates. Fifteen target error levels were considered for each benchmark. The worst-case error analysis was implemented as follows. Firstly, a candidate solution was converted to AIG representation. Then, the approximate miter was created. Finally, SAT solver available in ABC was called. The maximal number of conflicts (20 K for 32-bit multipliers) was used as a termination condition for the SAT solver.

In the case of approximate adders, the SAT-based worst case checking is extremely fast and many non-dominated implementations were discovered (for a more detailed analysis, please see [1]). The reason is that the adders are structurally less complicated than multipliers and the number of outputs is lower. In addition to that, the adders do not imply huge search spaces due to the excessive number of CGP nodes. While the exact 128-bit adder consists of 1000 gates, 32-bit multiplier requires over 6300 gates.

Depending on the structural similarity, several hours may be required to prove equivalence of 32-bit multipliers by means of the most advanced techniques of combinational equivalence checking. Despite of that, over one thousand of different approximate 32-bit multipliers were discovered within a given amount of time by means of the proposed approach. Interestingly, only 37% of SAT calls were terminated due to the resource limit. To demonstrate quality of discovered solutions, we will discuss only the 16-bit multipliers as we can perform an additional error analysis and investigate the parameters that were not considered during the evolution. For the sake of clarity, only the parameters of the non-dominated 16-bit multipliers are shown in Fig. 9.5. The first plot shows the results for MAE, power, and delay. The second plot does the same, but for WCE. The 16-bit multipliers directly created by CGP exhibit better trade-offs not only for WCE but also for other error metrics. The 16-bit multipliers constructed from the best 8-bit multipliers of the EvoApprox8b library provide substantially worse results. These multipliers consume more power for the same error level and their delay is extremely large. In addition to that, the 16-bit approximations created by CGP cover the wider range of the WCE/MAE axis. Figure 9.5 also depicts the properties of 16-bit Kulkarni's multiplier. The construction from 2-bit multipliers gives even worse results than the construction from 8-bit multipliers. The power consumption of Kulkarni's multiplier is only by 20% better compared to the accurate 16-bit tree multiplier.

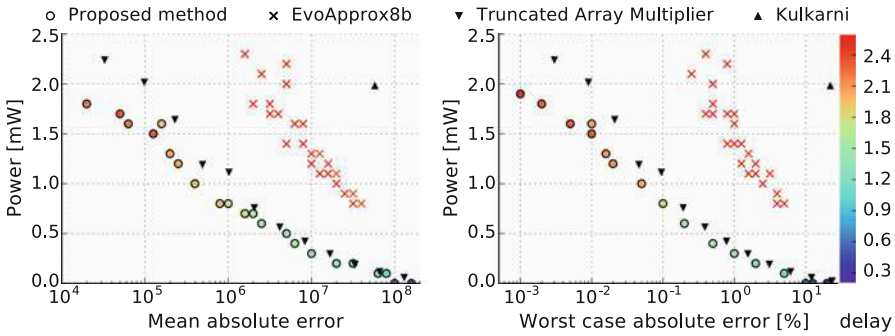



**Fig. 9.5** Parameters of various 16-bit multipliers synthesized using 45 nm technology with $V_{cc} = 1$ V

## 9.5 Case Study 2: Approximate Image Filters

The following case study is devoted to the automated design of approximate implementations of image filters, in particular *shot noise* filters. Its main purpose is to demonstrate that by means of a suitable CGP setting, a completely different class of circuits (with respect to the circuits presented in Sect. 9.4) can be approximated with CGP. In the case of image filters, CGP operates at the level of more complex components (such as 8-bit adders, minimum and maximum blocks) and uses the fitness function based on evaluating candidate circuit response in the application-specific context. We will compare the quality of filtering and electrical parameters of conventional image filters, approximate versions of these conventional image filters obtained by CGP and image filters evolved by CGP from a randomly generated initial population. All filters will operate with grayscale images in which every pixel is encoded on 8 bits.

### *9.5.1 Median Filter and Its Extension*

Conventional implementations of shot noise elimination filters are usually based on calculating the median over the pixels belonging to the filtering window (i.e., the kernel). The *median filter* (MF) can be implemented in several different ways [7]. In high-performance image and video processing systems, a pipelined implementation is often used to meet challenging performance constraints. The elementary component of such the pipeline is a *compare-and-swap* (C&S) operation which is in fact a small 2-input sorting network producing a sorted sequence by outputting the minimum and maximum of the input values. When properly arranged, the median value of a 9-item input vector (i.e., from the $3 \times 3$ pixel kernel) can be obtained using 19 C&S operations with latency of 9 stages [24]. The implementation of a 25-input median circuit then requires 99 C&S operations. The *center weighted median filter* (CWMF) represents a special extension of MF in which the central value of the window is counted with additional weight [10]. Compared to the median filter, this modification can preserve more details along the horizontal and vertical directions while suppressing additive white and/or impulsive-type noise.

In addition to the removal of noisy pixels, the median filters often remove desirable details from the image. In order to address this problem, *adaptive median filter* (AMF) is constructed as a multilevel filter which tries to detect and subsequently replace corrupted pixels only [8]. At each level, filtering windows of different sizes are utilized. Usually, two levels working with the $3 \times 3$ and $5 \times 5$ filtering window, respectively, are sufficient to obtain a very good image quality. A hardware implementation of AMF consists of two median filters, circuitry that determines minimal and maximal values for each filter window, and delay buffers to compensate different latency of median filters and simple logic (Fig. 9.6a).

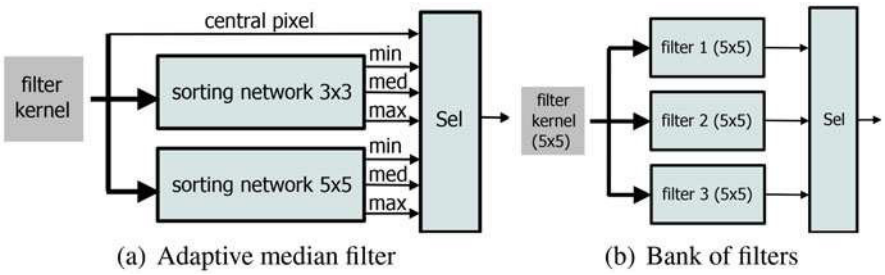



**Fig. 9.6** Extensions of the conventional median filter. (**a**) Adaptive median filter. (**b**) Bank of filters

Another straightforward extension of the median filter was proposed in [27]. If $k$ relatively simple, but different image filters are available, they can be employed to perform the filtering task over a given kernel in parallel (Fig. 9.6b). The resulting value is then determined from their outputs, for example, using the $k$-input median. This approach will be called the *bank of filters* in the rest of this chapter.

### 9.5.2 Approximate Median Filters

A transistor-level approximation of the C&S operation was proposed for median filters in [16]. In our case study, two CGP-based approximation strategies (AS1 and AS2 according to [24]) working at the level of C&S operations are implemented and compared.

AS1: In order to approximate MF with CGP, the resources-oriented method is employed. CGP tries to minimize the error of a candidate solution which is composed with up to $m$ 8-bit C&S operations, where $m$ is insufficient for creating a fully functional MF. The quality of a candidate approximate median function with $n = 2k+1$ inputs was determined using the so-called *distance error* $d_e = |j-k+1|$, which is the distance of the item chosen as the output value (i.e., $j$th lowest value) from the true median (i.e., the $k+1$th lowest value) [26]. Approximate CWMFs are obtained using the same techniques, but the size of the input vector increases as the central pixel value appears several times in the input vector. The approximate AMFs were constructed by replacing the exact 9-input median and 25-input median with their selected approximate implementations evolved by means of CGP. The rest of the AMF circuitry remained unchanged.

AS2: If a training image pair consisting of noisy image and noiseless (golden) image is available, a suitable image filter can be evolved with CGP from scratch as shown in [33]. The filter is composed of two-input elementary functions such as the addition, minimum, maximum, and various logic functions operating over 8-bit operands and producing 8-bit outputs. The error is measured by means of the *mean*

*absolute error* (MAE) between the outputs $O_f$ produced by a candidate filter and reference (golden) outputs $O_g$ for a given training image pair, formally:

$$MAE = \frac{1}{Z}\sum_{i=1}^{Z} |O_f(i) - O_g(i)|, \tag{9.12}$$

where $Z$ is the number of filtered pixels.

CGP typically produces many structurally different implementations showing very similar filtering properties. Evolved filters of this type will be denoted EVO. Three selected EVO filters are employed to form a bank of filters (BNK).

### 9.5.3 Experiments and Results

All conventional filters were described in VHDL and synthesized using Synopsys Design compiler with 45 nm PDK. The goal of the synthesis was to produce pipelined implementations operating at frequency at least 1 GHz. The basic component of all these filters, the 8 bit compare-and-swap operation, was implemented using an 8-bit magnitude comparator and two 8-bit multiplexers. In summary, we obtained the following implementations: median filter operating on $3 \times 3$ ($5 \times 5$) kernel denoted as MF9 (MF25), center weighted median filter operating on $3 \times 3$ pixels with the weight equal to 3 (CWMF9), and adaptive median filter (AMF25) with the kernel size $5 \times 5$ pixels.

In order to obtain approximate median filters in the resources-oriented scenario (AS1), CGP was seeded with the known implementation of an $n$-input median network exhibiting the minimal number of C&S operations and a corresponding $n$-input approximate implementation of the median network was evolved. CGP operated with $n_i = n$, $n_o = 1$, $n_a = 2$, $n_b = 2$, $\lambda = 20$, $h = 5$, and $10^7$ ($6.10^5$ respectively) generations were produced for 9-input (25-input, respectively) circuits. The function set $\Gamma$ contained 8-bit C&S functions and identity function.

In total, several hundreds of approximate implementations were produced by CGP for $n = 9, 11$ and 25. We identified ten Pareto-dominant solutions for each $n$ and synthesized them using Synopsys Design compiler to obtain their electrical parameters. More than 75% of power budget is due to switching activity of registers which were inserted to evolved designs in order to meet the performance constraint. Approximate adaptive median filters were composed of evolved approximate MF9 and MF25 implementations. The approximate versions of common filters are denoted by $\alpha$ #$\beta$, where $\alpha$ represents the conventional filter and $\beta$ is the identifier of the obtained approximation, for example, AMF25 #19 denotes the 19th version of the approximate adaptive median filter with the $5 \times 5$ pixel kernel.

In order to evolve image filters from scratch (AS2), CGP started with a randomly generated initial population and used two-input 8-bit functions (such as minimum, maximum, addition, absolute difference, and conditional assignment) and other

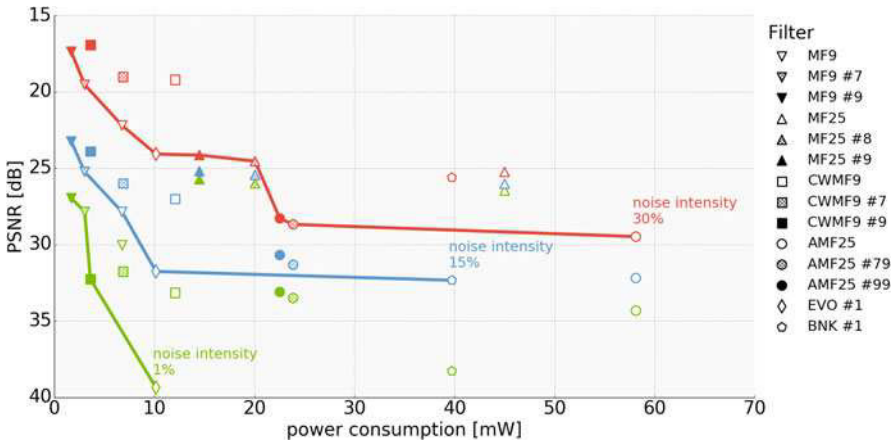



**Fig. 9.7** Mean PSNR and power consumption of selected image filters

settings ($n_i = 25$, $n_o = 1$, $n_c = 7$, $n_r = 9$, $n_a = 2$, $n_b = 1$, $\lambda = 7$, $h = 15$) according to [33]. All filters were evolved using fitness function given in Eq. (9.12) and appropriate training and golden images consisting of 384×256 pixels. The input training image was corrupted with 20% salt and pepper noise.

For the following comparisons, we selected two evolved filters representing two completely different design corners—low-cost EVO #1 and high-cost BNK #1. Both filters operate with the filter window consisting of $5 \times 5$ pixels. EVO #1 consists of 27 8-bit components (including 17 min/max functions) and occupies approximately the same area as MF9 but consumes about 50% more power. This is an interesting result because it operates on nearly three times higher number of inputs. BNK #1 was assembled from three different evolved filters. It shows very high quality of filtering especially for high noise intensities.

Figure 9.7 compares the quality of filtering and power consumption of all considered filters. The quality of filtering was evaluated using a set of 30 images and measured as the mean peak signal-to-noise ratio (PSNR) for three noise intensities (1%, 15%, and 30% pixels corrupted by salt&pepper noise). The mean PSNR indicates that filters evolved in AS2 mostly outperform other filters. AMF performs well, but it is a very expensive solution. Approximate AMFs are still very good in terms of PSNR, but they significantly reduced power consumption of the original AMF. However, power consumption of EVO#1 is 37% with respect to AMF25#19 and the quality of output images produced by EVO #1 is significantly higher. Approximate MFs and CWMFs should be used only if a very inexpensive implementation is required and lower filtering quality is acceptable.

We can conclude that, in this particular application, it is better to evolve image filters from scratch rather than to introduce approximate implementations to conventional median-based filters.

## 9.6 Conclusions

We have shown in this chapter how the problem of functional approximation of combinational circuits can be formulated as a multi-objective optimization problem and solved using Cartesian genetic programming. We focused on an efficient computation of error metrics because it is the main performance bottleneck if many candidate approximate circuits have to be generated and evaluated in one CGP run. Evolved approximate circuits were compared with available approximate implementations created by conventional techniques. Our comparison revealed outstanding quality of evolved approximate circuits.

The automated approximation approach conducted by means of CGP can be used not only for obtaining particular approximate implementations but also for generating a huge library contacting millions of approximate designs. Then, for purposes of a particular application, the user can quickly select an appropriate solution by means of a suitable user interface. Such a library would significantly accelerate the whole design process and integrating approximate circuits into a wide spectrum of applications.

**Acknowledgement** This work was supported by the Czech science foundation project 16-17538S.

## References

1. Ceska M, Matyas J, Mrazek V, Sekanina L, Vasicek Z, Vojnar T (2017) Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In: Proceedings of 36th IEEE/ACM international conference on computer aided design. IEEE, Piscataway, pp 416–423
2. Chan WTJ, Kahng AB, Kang S, Kumar R, Sartori J (2013) Statistical analysis and modeling for error composition in approximate computation circuits. In: 31st IEEE international conference on computer design (ICCD), pp 47–53
3. Chandrasekharan A, Soeken M, Große D, Drechsler R (2016) Precise error determination of approximated components in sequential circuits with model checking. In: Proceedings of DAC'16. ACM, New York, pp 129:1–129:6
4. Chen TH, Alaghi A, Hayes JP (2014) Behavior of stochastic circuits under severe error conditions. Inf Technol 56:182–191
5. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
6. Hrbacek R, Mrazek V, Vasicek Z (2016) Automatic design of approximate circuits by means of multi-objective evolutionary algorithms. In: Proceedings of the 11th international conference on design and technology of integrated systems in nanoscale era. IEEE, Piscataway, pp 239–244
7. Huang T, Yang G, Tang G (1979) A fast two-dimensional median filtering algorithm. IEEE Trans Acoust Speech Signal Process 27(1):13–18. https://doi.org/10.1109/TASSP.1979.1163188
8. Hwang H, Haddad R (1995) Adaptive median filters: new algorithms and results. IEEE Trans Image Process 4(4):499–502. https://doi.org/10.1109/83.370679

9. Jiang H, Liu C, Maheshwari N, Lombardi F, Han J (2016) A comparative evaluation of approximate multipliers. In: IEEE/ACM international symposium on nanoscale architectures. IEEE, Piscataway, pp 191–196
10. Ko S, Lee Y (1991) Center weighted median filters and their applications to image enhancement. IEEE Trans Circuits Syst 15:984–993. https://doi.org/10.1109/31.83870
11. Kulkarni P, Gupta P, Ercegovac MD (2011) Trading accuracy for power in a multiplier architecture. J Low Power Electron 7(4):490–501. https://doi.org/10.1166/jolpe.2011.1157
12. Lotfi A, Rahimi A, Yazdanbakhsh A, Esmaeilzadeh H, Gupta RK (2016) Grater: an approximation workflow for exploiting data-level parallelism in FPGA acceleration. In: 2016 design, automation and test in Europe conference and exhibition, DATE 2016, pp 1279–1284. https://doi.org/10.3850/9783981537079_0805
13. Mahdiani HR, Ahmadi A, Fakhraie SM, Lucas C (2010) Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. IEEE Trans Circuits Syst I Reg Papers 57(4):850–862. https://doi.org/10.1109/TCSI.2009.2027626
14. Miller JF (2011) Cartesian genetic programming. Springer, Berlin. https://doi.org/10.1007/978-3-642-17310-3
15. Miller JF, Thomson P, Fogarty T (1998) Designing electronic circuits using evolutionary algorithms. Arithmetic Circuits: A Case Study. Wiley, New York, pp 105–131
16. Monajati M, Fakhraie SM, Kabir E (2015) Approximate arithmetic for low-power image median filtering. Circuits Syst Signal Process 34(10):3191–3219. https://doi.org/10.1007/s00034-015-9997-4
17. Mrazek V, Vasicek Z (2016) Automatic design of arbitrary-size approximate sorting networks with error guarantee. In: 2016 26th international workshop on power and timing modeling, optimization and simulation. IEEE Computer Society, Piscataway, pp 221–228
18. Mrazek V, Sarwar SS, Sekanina L, Vasicek Z, Roy K (2016) Design of power-efficient approximate multipliers for approximate artificial neural networks. In: Proceedings of the IEEE/ACM international conference on computer-aided design. ACM, New York, pp 811–817. https://doi.org/10.1145/2966986.2967021
19. Mrazek V, Hrbacek R, Vasicek Z, Sekanina L (2017) Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In: Design, automation and test in Europe conference and exhibition, DATE 2017, pp 258–261. https://doi.org/10.23919/DATE.2017.7926993
20. Nepal K, Li Y, Bahar RI, Reda S (2014) ABACUS: a technique for automated behavioral synthesis of approximate computing circuits. In: Proceedings of the conference on design, automation and test in Europe, EDA consortium, DATE'14, pp 1–6
21. Nepal K, Hashemi S, Tann H, Bahar RI, Reda S (2017) Automated high-level generation of low-power approximate computing circuits. IEEE Trans Emerg Top Comput. https://doi.org/10.1109/TETC.2016.2598283
22. Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk
23. Sekanina L, Vasicek Z (2013) Approximate circuits by means of evolvable hardware. In: 2013 IEEE international conference on evolvable systems, IEEE CIS, Proceedings of the 2013 IEEE symposium series on computational intelligence (SSCI), pp 21–28
24. Sekanina L, Vasicek Z, Mrazek V (2017) Approximate circuits in low-power image and video processing: the approximate median filter. Radioengineering 26(3):623–632
25. Vasicek Z (2017) Relaxed equivalence checking: a new challenge in logic synthesis. In: 2017 IEEE 20th international symposium on design and diagnostics of electronic circuits systems (DDECS), pp 1–6. https://doi.org/10.1109/DDECS.2017.7968435
26. Vasicek Z, Mrazek V (2017) Trading between quality and non-functional properties of median filter in embedded systems. Genet Program Evolvable Mach 18(1):45–82. https://doi.org/10.1007/s10710-016-9275-7
27. Vasicek Z, Sekanina L (2007) An area-efficient alternative to adaptive median filtering in FPGAs. In: Proceedings of 2007 international conference on field programmable logic and applications. IEEE, Piscataway, pp 216–221

28. Vasicek Z, Sekanina L (2011) A global postsynthesis optimization method for combinational circuits. In: Proceedings of the design, automation and test in Europe DATE 2011, EDAA, pp 1525–1528
29. Vasicek Z, Sekanina L (2014) Evolutionary design of approximate multipliers under different error metrics. In: IEEE international symposium on design and diagnostics of electronic circuits and systems. IEEE, Piscataway, pp 135–140
30. Vasicek Z, Sekanina L (2015) Evolutionary approach to approximate digital circuits design. IEEE Trans Evol Comput 19(3):432–444. https://doi.org/10.1109/TEVC.2014.2336175
31. Vasicek Z, Sekanina L (2016) Evolutionary design of complex approximate combinational circuits. Genet Program Evolvable Mach 17(2):1–24
32. Vasicek Z, Slany K (2012) Efficient phenotype evaluation in Cartesian genetic programming. In: Proceedings of the 15th European conference on genetic programming. LNCS, vol 7244. Springer, Berlin, pp 266–278
33. Vasicek Z, Bidlo M, Sekanina L (2013) Evolution of efficient real-time non-linear image filters for FPGAs. Soft Comput 17(11):2163–2180. https://doi.org/10.1007/s00500-013-1040-8
34. Vasicek Z, Mrazek V, Sekanina L (2017) Towards low power approximate DCT architecture for HEVC standard. In: Design, automation and test in Europe conference and exhibition, DATE 2017, pp 1576–1581. https://doi.org/10.23919/DATE.2017.7927241
35. Venkataramani S, Roy K, Raghunathan A (2013) Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits. In: Design, automation and test in Europe, DATE'13, EDA consortium, pp 1367–1372
36. Venkatesan R, Agarwal A, Roy K, Raghunathan A (2011) MACACO: modeling and analysis of circuits for approximate computing. In: 2011 IEEE/ACM international conference on computer-aided design (ICCAD). IEEE, Piscataway, pp 667–673