


Università degli Studi di Napoli Federico II

Ph.D. Program in

Information Technology and Electrical Engineering

XXXVII Cycle

Thesis for the Degree of Doctor of Philosophy

A Methodology for the Synthesis of Approximate Circuits Targeting ASIC and FPGA Technologies

by

Alberto Moriconi

Advisor: Prof. Nicola Mazzocca

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Ai miei nonni

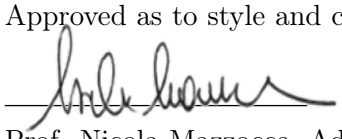
A Methodology for the Synthesis of Approximate Circuits Targeting ASIC and FPGA Technologies

Ph.D. Thesis presented
for the fulfillment of the Degree of Doctor of Philosophy
in Information Technology and Electrical Engineering
by

Alberto Moriconi

April 2025

Approved as to style and content by



Prof. Nicola Mazzocca, Advisor

Università degli Studi di Napoli Federico II
Ph.D. Program in Information Technology and Electrical Engineering
XXXVII cycle - Chairman: Prof. Stefano Russo

http://itee.dieti.unina.it

Candidate's declaration

I hereby declare that this thesis submitted to obtain the academic degree of Philosophiæ Doctor (Ph.D.) in Information Technology and Electrical Engineering is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.
Parts of this dissertation have been published in international journals and/or conference articles (see list of the author's publications at the end of the thesis).

Napoli, June 10, 2025

Alberto Moriconi

Abstract

Approximate computing is a recent design paradigm that exploits the inherent error resilience of a range of applications by trading off exact computation with some desirable property of the system, typically energy efficiency. In the scientific literature, many approximate computing techniques aiming at circuits having superior performances, in terms of speed, silicon area end energy requirements, have been proposed. However, they are not generic techniques and very often they are closely linked to a specific application.

In this thesis, we propose a generic and automatic methodology for the synthesis of approximate circuits based on the local rewriting of And-Inverter Graphs. Given a Hardware Description Language description of a combinational circuit, Satisfiability Modulo Theories based exact synthesis is used to build a catalogue of candidate substitutes for selected cuts of the circuit, that are formally proven to be size-optimal w.r.t. the number of AND nodes.

A heuristic based on a multi-objective simulated annealing algorithm is used to guide the choice of the cuts to substitute, in order to obtain a set of dominant configurations w.r.t. the error rate and the number of nodes in the circuit.

We evaluate our approach using different benchmarks, and, in order to measure actual gains, we perform actual synthesis of Pareto-optimal approximate configurations, both targeting the Application-Specific Integrated Circuit and Field Programmable Gate Array (FPGA) technologies. We also propose an alternative approach that uses a simple Look-Up Table power model to drive the optimization when targeting the FPGA technology.

Experimental results show that the proposed approach allows achieving significant savings, since resulting approximate circuits exhibit lower

circuit area, lower power dissipation and restrained error w.r.t. their exact couterparts.

Keywords: automated design methodology, approximate computing, multi-objective optimization, approximate logic synthesis, low area approximate circuits, low power approximate circuits. Ó

Sintesi in lingua italiana

L'approximate computing è un paradigma di design emergente che sfrutta l'inerente tolleranza all'errore di una gran quantità di applicazioni per rilassare i requisiti sulla correttezza del calcolo, ottenendo di contro dei benefici in termini energetici e di performance. Nella letteratura scientifica sono state proposte numerose tecniche di approximate computing con l'obiettivo di ottenere circuiti approssimati con performance superiori, in termini di velocità, occupazione di silicio e requisiti energetici. Tuttavia, tali tecniche non sono generiche e spesso sono applicabili solo a specifiche applicazioni.

In questa tesi è proposta una metodologia generica e automatica per la sintesi di circuiti approssimati basata sulla riscrittura locale di And-Inverter Graph. Partendo da una descrizione in un Hardware Description Language di un circuito combinatorio, tecniche di sintesi esatta basate su solver Satisfiability Modulo Theories sono utilizzate per costruire un catalogo di possibili sostituzioni per tagli pre-selezionati del circuiti, con garanzie formali rispetto alla loro ottimalità in termini di numero di nodi AND.

Un'euristica basata su un algoritmo simulated annealing multi-obiettivo è usata per guidare la scelta dei tagli da sostituire, in modo da ottenere un insieme di configurazioni dominanti rispetto al numero di nodi del circuito e della frequenza di errore.

La valutazione dell'approccio è svolta usando numerosi benchmark e, per misurare i guadagni effettivi, sono eseguite le sintesi delle configurazioni Pareto-ottime, sia per le tecnologie Application-Specific Integrated Circuit che Field-Programmable Gate Array (FPGA). È anche proposto un approccio alternativo che usa un semplice modello di dissipazione di potenza delle Look-Up Table per guidare l'approssimazione nel caso FPGA.

I risultati sperimentali mostrano che l'approccio proposto offre vantaggi

significativi, in quanto i circuiti approssimati ottenuti hanno area occupata minore, potenza dissipata minore ed errore contenuto rispetto agli originali esatti.

Parole chiave: metodologia di design automatica, approximate computing, ottimizzazione multi-obiettivo, sintesi di logica approssimata, circuiti approssimati ad area ridotta, circuiti di approssimazione a ridotta dissipazione di potenza.

Contents

Acronyms

#SAT Sharp Satisfiability. 27

ADC Approximate Don't-Care. 12

AEPIP Apparent Erroneous Primary Input Pattern. 8, 9

AIG And-Inverter Graph. xviii, xix, 2, 9, 29, 32–37, 39–42, 44, 47, 48, 52, 63, 64, 68, 69, 73, 77–79, 82, 83, 85, 91, 92, 99, 103, 122

ALS Approximate Logic Synthesis. 2, 5–7, 11, 21, 22, 31, 42, 107

AMOSA Archived Multi-Objective Simulated Annealing. 43, 44, 49, 50, 68, 69, 77, 110

ANN Artificial Neural Network. 68, 89

ASE Approximate Simplified Expression. 8, 9

ASIC Application Specific Integrated Circuit. xix, 2, 16, 48, 73–76, 79

ATM array-tree multipler. 50, 52, 70

ATPG Automatic Test Pattern Generation. 28

AWCE Absolute Worst-Case Error. xviii, 48, 50, 55, 56, 58, 59, 69, 70, 82–85, 92, 94, 97

AxC Approximate Computing. 1, 2, 5, 21, 42, 58, 74, 85

BDD Branch Decision Diagram. 29

BMF Boolean Matrix Factorization. 10

CL Convolutional Layer. 89

CLA carry-lookahead adder. 50, 55, 70

CNN Convolutional Neural Network. xx, xxi, 89, 90, 99–101

CSkA carry-skip adder. 50, 70

DAG Directed Acyclic Graph. 32, 42

DCT Discrete Cosine Transform. xviii, 58–61

DSE Design Space Exploration. 42, 44, 48–50, 68–70, 73, 77, 79, 82, 83, 87, 89, 92–94, 99

DTM Dadda-tree multipler. 50, 52, 70

EDC External Don't-Care. 12

ELIP Erroneous Local Input Pattern. 8, 9

EP Error Probability. xviii, 48, 50, 54, 69, 70, 92, 94

ES Exact Synthesis. xv, 35, 36, 40, 41, 44, 47, 58, 68, 77, 82

FCL Fully-Connected Layer. 89

FF Flip-Flop. 21

FIR Finite Impulse Response. xx, 87, 88

FPGA Field Programmable Gate Array. xix, xx, xxiii, 2, 15, 16, 37, 48, 63–65, 68–76, 78–80, 83, 85, 87, 90–97

FRAIG Functionally Reduced And-Inverter Graph. 49

HCA Han-Carlson adder. 50, 70

HDL Hardware Description Language. 2, 36, 44, 50, 68, 103

IER Inherent Error Resilience. 1

K-LUT K-Input Look-Up Table. 37

KSA Kogge-Stone adder. 55

LTL Linear Temporal Logic. 13

LUT Look-Up Table. xviii–xxi, xxiii, 15, 16, 37, 39–42, 44, 48, 63, 65–74, 77, 79, 80, 82–87, 93, 97–99, 103

MAE Mean Absolute Error. 83, 84, 97

MCSP Minimum Circuit Size Problem. 33

MCT Minterm Complement Threshold. 6

MED Mean Error Distance. xviii, 55, 57, 58

MIG Majority-Inverter Graph. 33

MNIST Modified National Institute of Standards and Technology. 90

MOCO Multi-Objective Combinatory Optimization. 42, 108

MOP Multi-objective Optimization Problem. 42–44, 89

NNMF Non-Negative Matrix Factorization. xvii, 10, 11

NSGA-II Non-dominated Sorting Genetic Algorithm-II. 15, 100

ODC Observability Don't-Care. 9, 12

PI Primary Input. 32, 36, 48, 55

PO Primary Output. 8, 32, 48, 55, 69

PSNR Peak Signal-to-Noise Ratio. xx, 26, 60, 86–88

QCC Quality Constraint Circuit. 11, 13

QEC Quality Evaluation Circuit. 12

RCA ripple-carry adder. 50, 55, 70

REPIP Real Erroneous Primary Input Pattern. 8, 9

RTL Register Transfer Level. 14, 21

SA Stuck-At. 7, 8, 28

SAT Boolean SATisfiability. 26, 27, 33

SDC Satisfiability Don't-Care. 8, 9

SMT Satisfiability Modulo Theories. 33, 34, 36, 41, 47, 103, 121, 122

SQCC Sequential Quality Constraint Circuit. 12, 13

SS Substitute Signal. 18

SSIM Structural SIMilarity index. 26

SwA Switching-Activity. xix, 64, 65, 67, 74, 77, 78, 92–94, 97, 99

TS Target Signal. 18

WTM Wallace-tree multipler. 50, 52, 55, 70

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Approximate Computing (AxC) is a recent approach to the design of digital systems that aims to trade-off correctness of results for some desirable property of the system, typically energy efficiency [HO13; XMK15].

It relies on the Inherent Error Resilience (IER) of some applications, that is their property of being able to produce acceptable outputs even when some of their underlying computations are incorrect or approximate [Chi+13].

The sources that contribute to IER can be classified as properties of

1. the **inputs**, that tipically are noisy and redundant, causing similar data to be processed multiple times,
2. the **outputs**, because a reference output may not exist and a range may be instead considered acceptable,
3. the **computation patterns**, that may introduce some level of self-healing or controllability of errors.

These properties are common to a number of domains, such as media processing (audio, video, graphics, and image), wireless communications, web search and data analytics. For example, real-world datasets may have significant redundancy, lowering the impact of individual errors; images are expected to be consumed by humans, whose perceptual limitations make minor variations undiscernable; some iterative algorithms may apport diminishing improvements with each iteration, allowing for early termination.

Techniques for AxC can be applied at many levels [Mit16]. For example, *precision scaling* consists in changing the bit-width of the input and/or intermediate results, and can be applied both during design [SR10] or dinamically [Tia+15]; *loop perforation* works by skipping some iterations of target loops [Sid+11].

These techniques target the software and the storage level; at the hardware level, numerous approximate versions of arithmetic circuits have been proposed, such as adders [Gup+11; Yan+13; AKL16] and multipliers [HBR15; LHL14; LL13]. These circuits are however finely hand-tuned and based on a precise insight of the realized function and of the algorithm, or on the analysis of the layout of the circuit at the transistor level.

Ideally, the adoption of approximate circuits could benefit from a systematic methodology and tools that would enable the user to provide some form of circuit specification (in a variety of formats and Hardware Description Languages (HDLs)) and appropriate quality constraints, and automatically generate approximate circuits that conform to such constraints and, by means of the approximation, provide energy, performance or area benefits.

These methodologies fall under the umbrella term of Approximate Logic Synthesis (ALS) [SG10], and they all need to tackle a number of fundamental problems:

1. How to obtain an approximate variant, given a circuit specification.

2. How to choose the error metrics used to quantify the difference between the original circuit and the variants, and how to evaluate them.

3. How to mitigate the complexity of the design space exploration, given the enormous number of possible alternative circuits.

The contribution of this thesis work is a new ALS methodology based on local transformations of AIGs. It improves the alternatives by providing a finer degree of control on the approximations, and has shown promising results in synthesis both targeting the ASIC and the FPGA technologies.

This thesis work is structured as follows: an overview of existing techniques for the automatic synthesis of approximate circuits and on the most used error metrics is provided in Chapter 2; in Chapter 3 introduces the proposed methodology, that has been implemented in an open source

synthesis suite and tested on a number of standard logic benchmarks, is detailed. The methodology is applied to a number of generic logic and arithmetic benchmarks; the experimental results for the ASIC and FPGA technologies are reported respectively in Chapter 4 and Chapter 5. In Chapter 6 a refinement for the power optimization in FPGA scenarios is proposed, driving the optimization step with a simple LUT power model; detailed experimental results and a case study is proposed in the same chapter.

The appendices are devoted to some theoretical background in Boolean circuits theory and combinatorial optimization, and a quick tutorial for some of the used tools.

Chapter 2

Background and related works

In this chapter, we give an overview of the background and related works on ALS.

In Section 2.1 we introduce the automatic synthesis techniques for AxC; Section 2.2 is devoted to a thorough discussion of the error metrics for the evaluation of approximate circuits.

Appendix A provides some additional background on Boolean functions and circuits.

This chapter extends the background content originally appeared in [Mor19] with novel approaches as reported in [Bar+22a] and [Bar+24].

2.1 Automatic synthesis techniques for AxC

A number of techniques have been proposed for the automatic synthesis in AxC; a taxonomy has been proposed in [Ran+19], where a first distinction is made between the synthesis of approximate circuits and of quality configurable circuits.

2.1.1 Approximate circuits

Approximate circuit synthesis can be introduced as the problem of modifying the functionality realized by a *golden circuit* C in a way that reduces area, depth or energy consumption, while satisfying predetermined error constraints. The techniques to obtain such approximate circuit C'

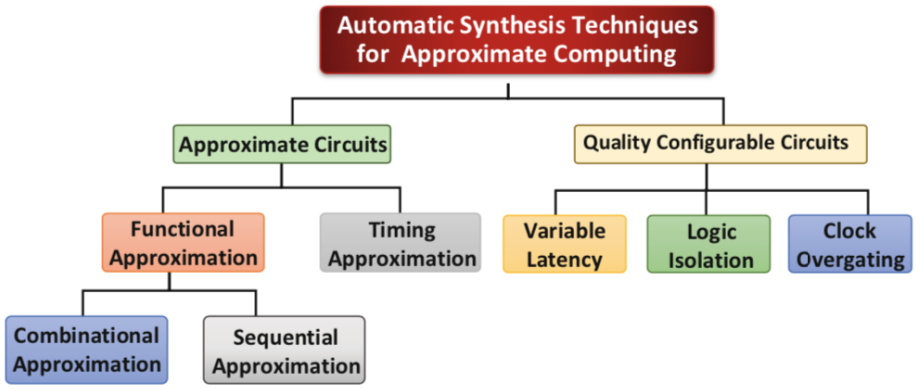


Figure 2.1. Synthesis techniques for AxC (from [Ran+19])

can be divided in (1) functional approximation and (2) timing approximation.

Functional approximation

Functional approximation techniques involve the modification of the logic implemented by the circuit. The methodology we present in this thesis work is based on functional approximation, therefore in this section existing methodologies and techniques are analyzed. The details of the presented methodology are discussed in Chapter 3.

Two-level simplification techniques Exact two-level simplification techniques, such as Karnaugh maps coverage and the Quine-McCluskey method, are important because they are simple to understand and provide a form that is both canonical (i.e. unique) and has the fewest terms for given specification [KB05].

An extension of the classical coverage method to ALS is proposed in [SG10]; given a Boolean function of n inputs, its Minterm Complement Threshold (MCT) can be defined as the maximum number of minterms in its care-sets (i.e. on- and off-sets) that can be complemented without violating a constraint on error probability; it can be shown that for a threshold r, it results $MCT = r \cdot 2^n$.

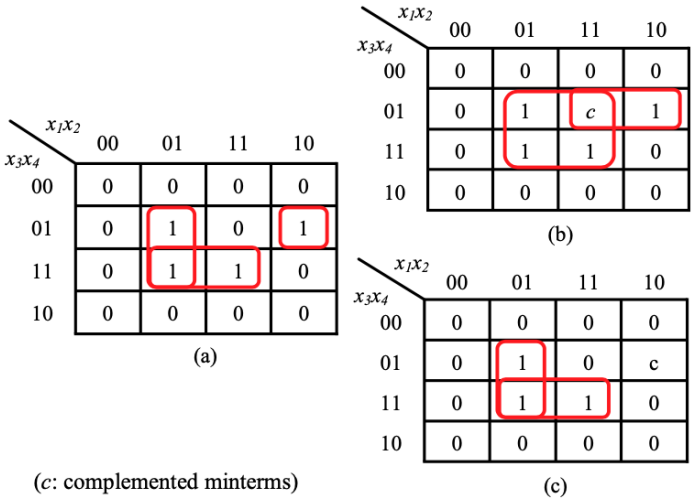


Figure 2.2. Literal reduction obtained by complementing a minterm (from [SG10])

The ALS problem is then reduced to the search of the complementation operation that can expand (or remove) prime implicants in such a way to reduce the literals, as shown in Figure 2.2. An *assisting expansion* heuristic is used to reduce the search space, pruning the minterms that can't be used to expand any prime implicant.

In [EWT18] a similar coverage technique is used, and a design space exploration technique based on the NSGA-II genetic algorithm [Deb+02c] is applied to obtain a number of pseudo-optimal alternative coverages. Appendix B further elaborates on the role of metaheuristics in ALS.

Node-level techniques A node-level approach based on the concept of *redundancy propagation* is proposed in [SG11]: when a Stuck-At (SA) fault is injected at a wire, it is possible to carry out simplifications both backwards, deleting all fan-ins of the wire up to branching points, and forward, depending on the traversed gates, as shown in Figure 2.3. A greedy heuristic is used to select target wires for fault injection, and candidate circuits are evaluated based on the error significance metric.

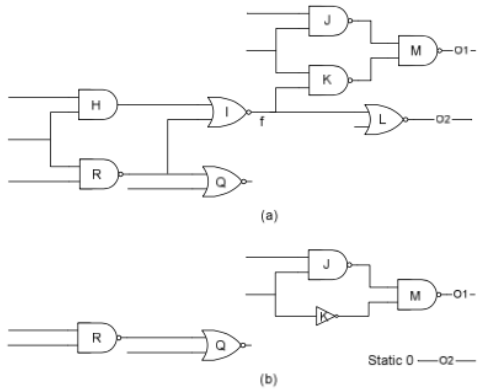


Figure 2.3. Example of redundancy propagation with wire $fSA1$ (from [SG11])

The *single-multi selection* approach [WQ16] can be considered an extension of the fault injection approach. Instead of introducing a SA fault at a given wire, the *factored form* of the node, i.e. a multi-level parenthesized expression that alternates between AND and OR operations, is approximated by removing one or more literals; such an expression is called an Approximate Simplified Expression (ASE), and an example of this modification is shown in Figure 2.4.

Although non-canonical, factored forms are useful because they allow to reduce wires in a design by reusing nodes with more complex expressions; for this reason, a number of minimization techniques exist such as [Bar+88; BHS90; FMK91].

Not all errors at an ASE node can propagate to the Primary Outputs (POs): for this reason, the methodology distinguishes between Apparent Erroneous Primary Input Patterns (AEPIPs) and Real Erroneous Primary Input Patterns (REPIPs); e.g. in Figure 2.4, 1110 is an AEPIP (because with $i_0 = 0$ the error at n_2 can't propagate to PO f), while 1111 is an REPIP. In the same vein, the local input pattern that elicits an error at a given node is said an Erroneous Local Input Pattern (ELIP) for the node.

In multi-level logic, each node has associated sets [Mic94] of

- Satisfiability Don't-Cares (SDCs), i.e. impossible patterns of inputs to and outputs from the node, and

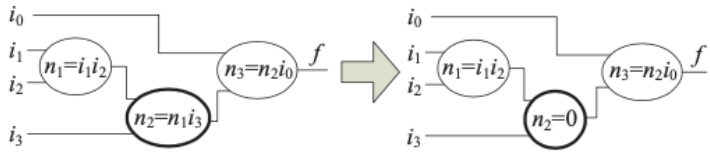


Figure 2.4. Example of approximate simplified expression (from [WQ16])

- Observability Don't-Cares (ODCs), i.e. input patterns to the node that make the network output insensitive to the output of the node.

The basic idea of the methodology is to enumerate first a subset of the SDCs and ODCs for the node; then only ELIPs that aren't SDCs or ODCs for the node are both possible inputs and propagable; the sum of the error rates for these ELIPs is an upper bound for the error rate of the ASEs, because not all input patterns that generate them are REPIPs (e.g. $(n_1, i_3) = (1, 1)$ in Figure 2.4 can be generated by the AEPIP 1110).

Two algorithms are proposed: in single-selection, the most effective node to shrink is chosen greedily, until the error rate excedes a given threshold. The multi-selection algorithm is based on the insight that the chice of multiple nodes at once can be formulated as a variation of the binary knapsack problem [Hil12], that is then solved applying a dynamic programming solver.

In [Cha+16b] an approximation-aware approach to AIG rewriting [MCB06] is proposed, where smaller cuts on critical paths are substituted with the constant 0.

The evaluation of error makes use both of approximation miters [Cha+16b] and exact algorithms based on a BDD representation [Soe+16].

Reinforcement learning has been recently proposed both for "standard" [Haa+18] and approximate logic synthesis.

Specifically, the Q-ALS [PNP19] methodology is based on the Q-learning [WD92] algorithm: a Q-agent is trained on a set of networks, represented as AIGs, learning for each node the maximum Hamming distance that doesn't violate a global error rate constraint, evaluated with a probabilistic approach

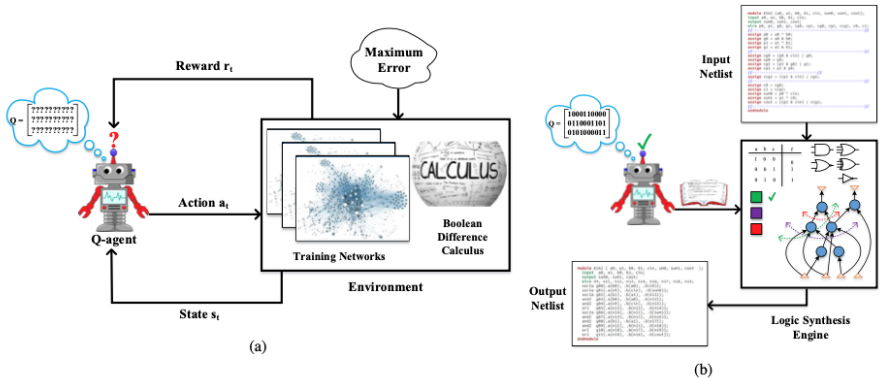


Figure 2.5. Training (a) and testing (b) in Q-ALS (from [PNP19])

based on Boolean differences [MPP11].

Boolean matrix factorization The *Blasys* methodology, proposed in [HTR18], is based on Boolean Matrix Factorization (BMF) [MV11], that is the approximation of a matrix by a product of two or more matrices; more specifically, NNMF is used, where a $k \times m$ matrix $\mathbf{M}$ is factored into two non-negative matrices $\mathbf{B}$ and $\mathbf{C}$, of size respectively $k \times f$ and $f \times m$, such that $\mathbf{M} \approx \mathbf{BC}$, where f is the *factorization degree*, and from a statistical standpoint represents the number of *features* of the original matrix to consider.

The circuit is first evaluated in order to find its truth table; the resulting matrix $\mathbf{M}$ is then factorized, and the resulting $\mathbf{B}$ and $\mathbf{C}$ matrices are then used to synthesize two circuits, respectively named *compressor* and *decompressor* circuit, that approximate the function realized by the original circuit introducing an area benefit, as shown in Figure 2.6.

The NNMF algorithm heuristically tries to minimize the Hamming distance between the original specification and the factored one, optionally taking into consideration a set of weights for the outputs. To address computational limitations, NNMF can be applied to subcircuits of a previously decomposed circuit.

Exploiting output observability don't-cares SALSA (*Systematic methodology for Automatic Logic Synthesis of Approximate circuits*) has

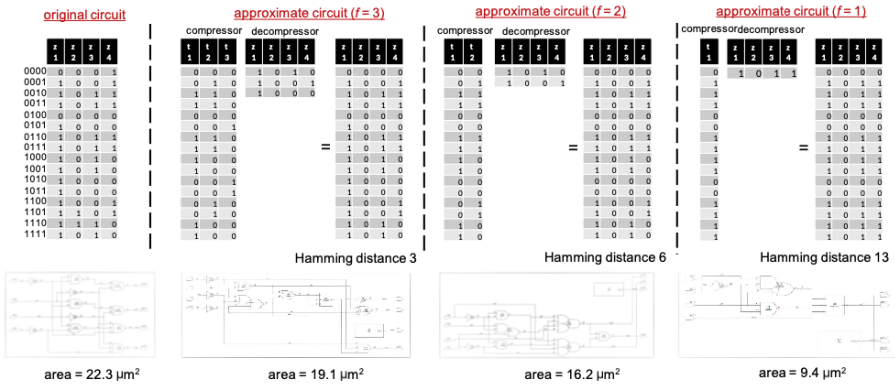


Figure 2.6. Results of NNMF with different f (from [HTR18])

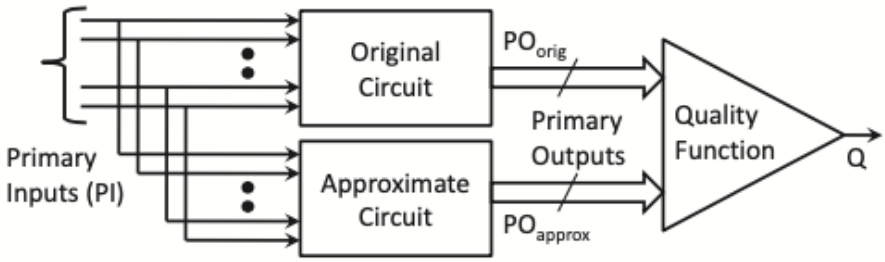


Figure 2.7. Quality constraint circuit (from [Ven+12])

been proposed in [Ven+12] as a reformulation of the ALS problem such that existing synthesis tool may be used without need for extensive modification.

The core idea is the use of a Quality Constraint Circuit (QCC), shown in Figure 2.7, whose primary inputs are the primary inputs of the original circuit; the primary outputs of the original and approximate variant of the circuit serve as the input of a *quality function*, whose output is a single bit Q that indicates if the constraints on the output are satisfied. Any of the quality metrics on the error magnitude previously discussed can be encoded this way.

The approximate circuit satisfies the constraints if the QCC is a tautology, i.e. if it evaluates to 1 for all inputs. For each output of the approx-

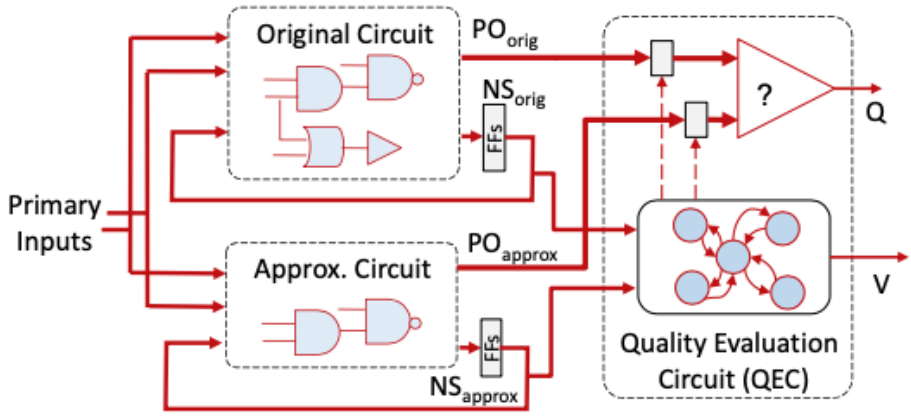


Figure 2.8. Sequential quality constraint circuit (from [Ran+14])

imate circuit, its ODC set is the set of primary input values for which Q is insensitive to the selected output [Mic94]; SALSA calls these ODC sets Approximate Don't-Cares (ADCs), because they can be set as External Don't-Cares (EDCs) for the approximate circuit with no constraint violation.

This procedure can be iterated over all approximate circuit outputs, or halted at any point. All of the intermediate circuits obtained this way are approximate variants that satisfy the given constraints.

Synthesis of sequential approximate circuits ASLAN (*Automatic methodology for Sequential Logic ApproximatioN*) [Ran+14] can be considered as an extension of the SALSA methodology to sequential circuits, in that the impact of the approximation on the primary outputs of the (sequential) circuit is characterized with a Sequential Quality Constraint Circuit (SQCC), shown in Figure 2.8.

Instead of a quality function, ASLAN makes use of a Quality Evaluation Circuit (QEC) that

- checks the state registers of the original and approximate circuit, and raises the *valid* output V when both circuits are ready for quality evaluation, and

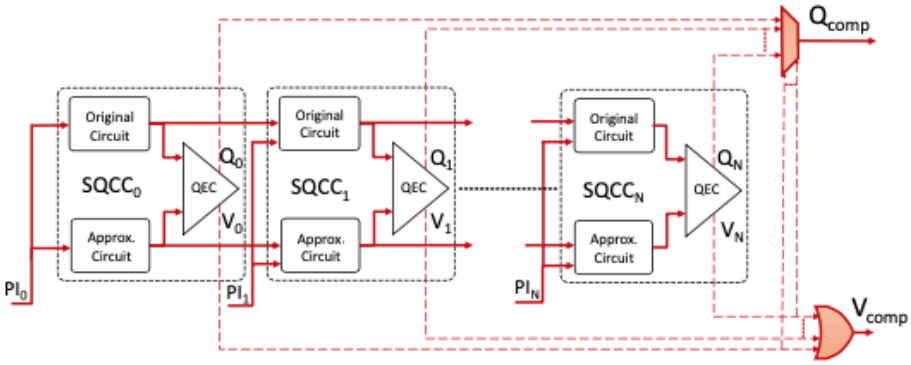


Figure 2.9. Sequential quality constraint circuit unrolling (from [Ran+14])

- checks that the outputs satisfy the encoded quality constraint and, if it is so, raises the *quality* output Q.

This behavior can be expressed as a model checking problem on the SQCC, where the following Linear Temporal Logic (LTL) [BK08] properties have to be true:

- the *safeness* property $\square(V \implies Q)$, that is, in all possible states, if V is true, Q is true, and
- the *liveness* property $\Diamond V$, that is, in all possible states, V eventually becomes true.

These properties are formally verified unrolling the circuit, as in Figure 2.9, and using the relevant V output as a selection input to the multiplexer that gathers the Q outputs.

Candidate combinatorial components are identified in the circuit, and a gradient descent approach is used to find a (local) quality-energy optimum, as shown in Figure 2.10, where two components C_1 and C_2 are considered; each configuration can be accepted if it satisfies both a selection heuristic (a figure of merit based on the introduced energy saving and local error) and the formal properties enforced by the SQCC.

It should be noted however that realizing QCCs and SQCCs is itself a non-trivial design task.

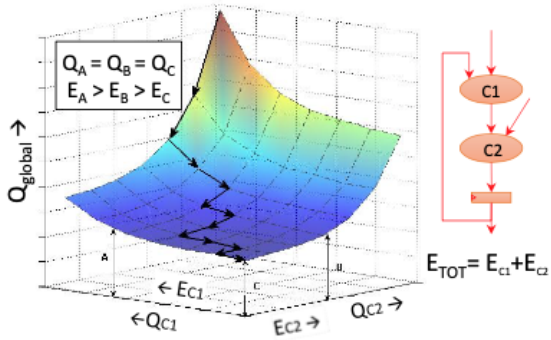


Figure 2.10. Selection of components (from [Ran+14])

Behavioral description transformation The approach used in ABACUS (Automated Behavioral Approximate Circuit Synthesis) [Nep+14] is different from the other proposed methodologies because it applies a number of transformation operators directly to a behavioral description of the circuit, such as simplification of data types, transformation of arithmetic expressions, variable-to-constant substitutions, and loop unrolling.

The obtained variants are evaluated using a greedy iterative algorithm.

Other recent advances Recent contributions from the scientific literature address at the circuit design problem by simultaneously optimizing both the error entailed by approximation, and hardware resource requirements [Bar+21; SVM19]. Authors of [SVM19], for instance, propose a Cartesian genetic-programming based method for automated functional approximation of digital circuits at either gate or Register Transfer Level (RTL) level. Every candidate circuit is represented as a two-dimensional grid of components, the type of which depends on the abstraction level. Each component is represented using a vector encoding input and output connections and the function being implemented by the component. Hence, the whole circuit is represented using the concatenation of such vectors, and approximate designs are created introducing mutations, i.e., random modifications affecting either input connections, the implemented function or the component output connection. Circuits are evaluated both in terms of error and silicon area, and those minimizing both are selected

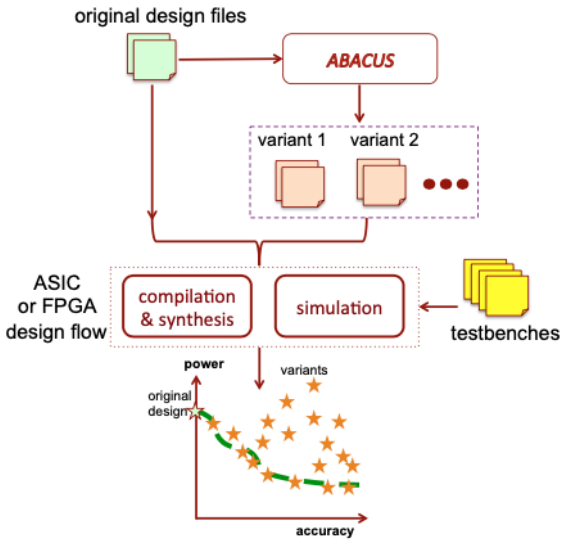


Figure 2.11. The ABACUS flow (from [Nep+14])

using Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [Deb+02a]. The approach is evaluated using several arithmetic circuits as case studies, optimizing for error, circuit delay and power consumption. In [SAP18] authors propose *circuit carving*, a methodology that applies binary tree exploration to determine the maximum portion of a circuit that can be removed without violating a given error constraint. In order to be able to evaluate the effects of the approximation technique on the circuit and to guide the binary tree exploration, weights have to be preliminarily assigned to gates in the circuit; this can be done exactly only for smaller circuits; otherwise, topological considerations are needed to determine an appropriate approach to the weight assignment.

Approaches specific to FPGA technology Several contributions exploit the underlying architecture of fabrics to break the carry chain at one or multiple positions, while exploiting unused LUT inputs to predict the carry [Pra+18; AAH21]. Authors of [UMK18; Ull+22b] exploits the structure of the 6-input lookup tables and carry chains of FPGAs, defining a methodology for recursively designing $n \times n$ multipliers based on $\frac{n}{2} \times \frac{n}{2}$

ones, optimized for FPGA-based systems. As drawbacks, rather than using the carry chain, the critical path delay and power consumption are reduced using additional LUTs to either compute or predict the carry for partial product summation.

In [Ull+22a], the authors propose a methodology for designing application-specific approximate arithmetic operators for FPGA based systems. The methodology utilizes the 6-input LUTs and the associated carry chains to implement approximate operators while exploiting multi-objective Bayesian optimization to search for approximate multiplier that satisfy an application's accuracy and performance constraints. Furthermore, to cope with the large design space, various machine-learning models have been adopted to estimate the behavioral accuracy and corresponding performance gains of approximate circuits.

A different approach has been proposed in [Pra+20], that exploits machine-learning models to sift libraries of approximate components, which have been previously designed using ASIC-oriented approximation techniques. That methodology articulates in two distinct phases, i.e., the training of predictors for FPGA hardware requirements and the actual Pareto-front construction. It has been tested on circuits belonging to the EvoApprox8 library of approximate components [Mra+17], searching for those providing optimal trade-offs between error and FPGA overhead, specifically the power consumption. Nevertheless, the machine-learning based estimation approach of hardware resources is unfeasible if no library of approximate components is available, and, additionally, it demands high-fidelity estimators, that, as authors themselves observed, may be thoroughly cumbersome to achieve, even if a large library of components is available.

Timing approximation

In timing approximation, it's not the implemented logic, but the operating conditions of the circuit that are modified; a typical example is *voltage overscaling* [HS99], where the operating voltage is lowered without lowering the clock frequency.

While this clearly reduces power dissipation, the naive application of this technique is bound to cause a large number of timing errors, due to the critical path delay possibly becoming greater than the clock period.

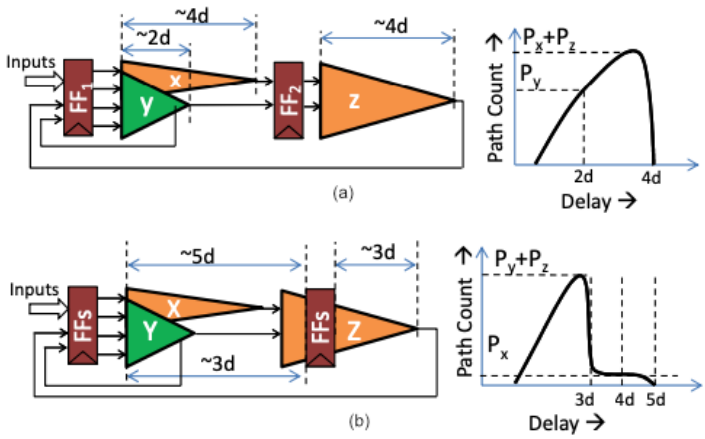


Figure 2.12. Original (a) and retimed (b) circuit (from [Ram+13])

Recovery Based Design techniques have been proposed, based on circuit instrumentation, in order to detect and recover timing errors [Bow+09]; however, circuits produced by standard synthesis tools tend to be ill-suited to timing approximation, because of the so-called *path wall* phenomenon: a great number of paths with similar delays due to the path balancing, a property that is usually desired (e.g. for pipelined circuits performance), but problematic for the application of this technique.

Retiming techniques [Ram+13] can be used to realize a path-delay distribution that gracefully degrades after a given path wall, therefore permitting some degree of control on timing errors, as shown in Figure 2.12.

2.1.2 Quality configurable circuits

In many applications, notably image processing, the error constraints may vary during operation and the algorithm resilience itself may depend on the particular section of the data that is being processed. For this reason, quality configurable circuits, i.e. circuits that can vary their accuracy at runtime, have been proposed.

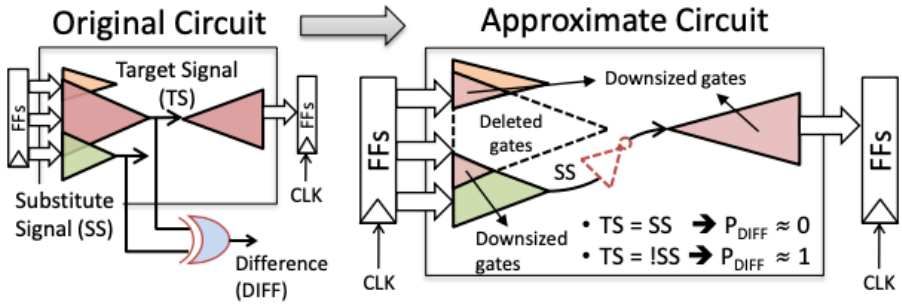


Figure 2.13. Substitute and simplify approach (from [VRR13])

Variable latency

In variable latency circuits, an approximate circuit C' can compute $\mathbf{f}_{C'}(i)$ for some input i with a delay d'; the circuit also has a mean (look-up table, comparation, ...) to determine if $\mathbf{f}_{C'}(i) \neq \mathbf{f}_C(i)$ and, in this case, compute the exact output with a cumulative delay $d > d'$.

A number of hand-crafted speculative arithmetic circuits have been proposed [VBI08]; however, some functional approximation techniques also provide the means to realize a variable latency version of the original circuit.

In SASIMI (*Substitute And SIMplIfy*) [VRR13] the key idea is to identify pairs of signals that are "similar", and to replace selected Target Signals (TSs) with Substitute Signals (SSs) that can be logic constants or other signals in the circuit. As shown in Figure 2.13, the SS is chosen in such a way to introduce some additional slack, so that not only the logic cone that generates the TS can be removed, but its transitive fan-out can also be downsized.

As shown in Figure 2.14, the logic generating the TS can be retained and additional comparation logic for the TS and the SS can be added. In approximate mode, the result of the comparation is simply ignored; in exact mode, the circuit operates in a single clock cycle if the TS and the SS are equal; otherwise, in an additional cycle, the exact result is computed from the point of substitution using the value of TS.

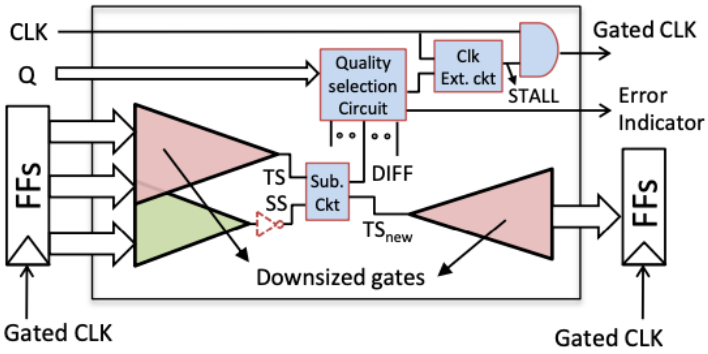


Figure 2.14. Quality configurable circuit using SASIMI (from [VRR13])

Logic isolation

Logic isolation achieves energy efficiency by spacially disabling parts of the circuit, without affecting its delay. For this reason, it lends better to system integration, and can be used to gradually degrade circuit accuracy in order to obtain different quality levels.

The technique aims to identify the parts of the circuit that contribute minimally to output accuracy and disable them, isolating the operands or the power supply (Figure 2.15). This gives clear advantages in dynamic power, as the isolated logic and its downstream logic will have no switching activity.

The circuit is divided in fan-out free logic cones, and each cone is processed indipendently; as little as a single wire, or as much as the entire cone can be isolated. As shown in Figure 2.16, from a power benefit perspective, isolating a greater part of the cone will give increasing savings; however, additional logic will be needed to implement the isolation; this identifies an optimum region before the net benefit start decreasing.

The methodology also tries to actively prioritize isolations that introduce error compensation.

Clock overgating

Clock overgating, proposed in [Kim+16], can be considered as an extension to the concept of clock gating: when the circuit is operating at

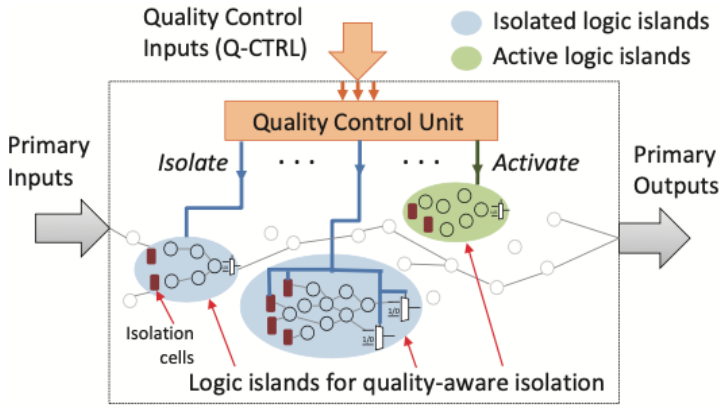


Figure 2.15. Logic isolation (from [JVR16])

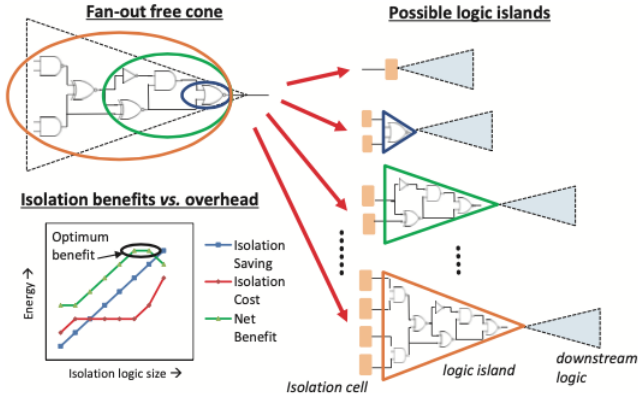


Figure 2.16. Gains and overhead of logic isolation (from [JVR16])

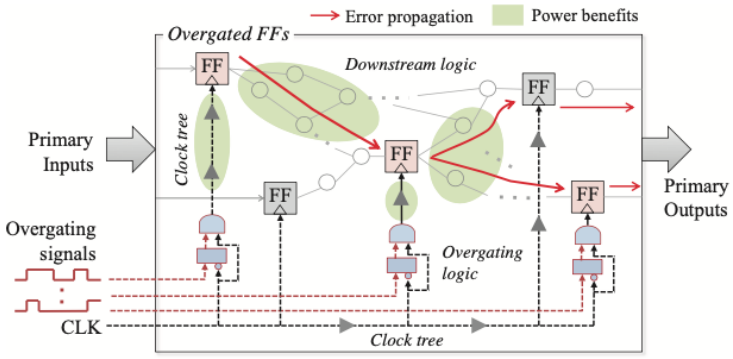


Figure 2.17. The concept of clock overgating (from [Kim+16])

reduced quality, the clock signal to selected Flip-Flops (FFs) is disabled, locking the evolution of part of the circuit and reducing the switching activity.

A number of heuristics based on RTL analysis is used to group together registers, and to select existing signals as clock overgate enables; the effect of gating the clock to selected FFs is evaluated using a gradient descent heuristic.

2.2 Error metrics

Evaluating the output quality is a fundamental problem in AxC. During an ALS flow, error metrics can be applied in two ways:

1. as **thresholds**, i.e. if an approximate circuit C' violates a given error constraint, it's rejected; or

2. as **penalties**, i.e. no hard constraint is enforced, the error for C' is recorded and used to "rank" the alternatives.

The way the alternatives are actually ranked constitutes an optimization problem, and theoretical details are given in Appendix B.

Consider a circuit C with m inputs and n outputs that computes a function

$$\mathbf{f}_C : \{0,1\}^m \to \{0,1\}^n$$

and assume we apply to it an ALS technique, producing a new circuit C' that computes a different function

$$\mathbf{f}_{C'} : \{0,1\}^m \to \{0,1\}^n \quad .$$

In order to constrain and/or characterize the error introduced by C', different metrics can be used, based on the error magnitude, the error frequency, or both.

A summary table of the introduced metrics is provided in Table 2.1.

2.2.1 Metrics Defined on Error Magnitude

In order to define the metrics based on error magnitude, arithmetical weights (usually powers of two) have to be assigned to the outputs of the circuits.

Given a Boolean function $\mathbf{f} : \{0,1\}^m \to \{0,1\}^n$, we refer to the component relative to each output as f_k, for $k \in \{0,\ldots,n-1\}$. We denote with *inputs* the set of all possible inputs to the circuit, and define $O_C(\mathbf{i})$ (and $O_{C'}(\mathbf{i})$) as the numerical interpretation of the outputs, obtained as the arithmetical sum of the weighted outputs of the circuits, with $\mathbf{i} \in$ *inputs*; i.e. assigned a set of weights

$$W_C = \{w_{C,0},\ldots,w_{C,n-1}\}$$

we define

$$O_C(\mathbf{i}) = \sum_{k=0}^{n-1} w_{C,k}\, f_{C,k}(\mathbf{i})$$

and

$$O_{C'}(\mathbf{i}) = \sum_{k=0}^{n-1} w_{C,k}\, f_{C',k}(\mathbf{i}) \quad .$$

Each of the introduced metrics can be also defined as unidirectional, i.e. restricting the errors to be only positive or negative.

Maximum error magnitude

Constraining the maximum error magnitude means ensuring that, for each input, the difference between the outputs of the exact and the approximate circuit is less than a given threshold.

It is a very general metric, that finds application every time it's possible to define a threshold to the error magnitude above which the output becomes unusable.

$$\epsilon_{\max} = \max_{\mathbf{i} \in inputs} |O_C(\mathbf{i}) - O_{C'}(\mathbf{i})| \quad .$$

Relative error magnitude

Constraining the relative error magnitude serves to ensure that, for each input, the difference between 1 and the ratio of the approximate output to the exact output is less than a certain margin in absolute value.

It is useful for the cases where the error threshold that makes an output unusable is not constant, but it is inversely proportional to the output magnitude.

$$\epsilon_{rel} = \max_{\mathbf{i} \in inputs} \left| 1 - \frac{O_{C'}(\mathbf{i})}{O_C(\mathbf{i})} \right| \quad .$$

Average error magnitude

The average error magnitude is used to constrain the sum of all the absolute differences between exact and approximate output, averaged over all the inputs; here we use the notation $\#inputs$ to indicate the cardinality of the set $inputs$.

It is useful because some circuits may exhibit sporadic errors but behave correctly "almost always"; in these cases, averaging over the set of the inputs may give a more realistic characterization of the error.

$$\epsilon_{avg} = \frac{\sum_{\mathbf{i} \in inputs} |O_C(\mathbf{i}) - O_{C'}(\mathbf{i})|}{\#inputs} \quad .$$

Mean squared error magnitude

The mean squared error magnitude is similar to the average error magnitude; in place of the absolute difference between the exact and approximate output, the squared difference is considered.

While similar to the average error magnitude, it may be perferred in some applications because it emphasizes the weight of outliers and, when used as part of a composite metric, is easier to manipulate algebrically than metrics that use the absolute value.

$$\epsilon_{MS} = \frac{\sum_{\mathbf{i} \in inputs} (O_C(\mathbf{i}) - O_{C'}(\mathbf{i}))^2}{\#inputs} \quad .$$

Error significance

The error significance is the weight of the most significant function component that is high when the maximum error magnitude occurs.

While it gives less information that the other proposed metrics, it has the great benefit of being simple to evaluate. The matter is further elaborated in Section 2.2.4.

$$\epsilon_{sig} = \max\{w_k \mid k \in \{0, \dots, n-1\} \wedge (\exists\, \mathbf{i} \in inputs \colon f_{C,k}(\mathbf{i}) \neq f_{C',k}(\mathbf{i}))\} \quad .$$

2.2.2 Metrics defined on error frequency

The metrics bounding the error frequency are more general, because they can be applied to circuits whose outputs can't be interpreted numerically.

For a multi-output Boolean function, we say that it differs from another Boolean function if one or more outputs differ, that is

$$\mathbf{f}(\mathbf{i}) \neq \mathbf{f}'(\mathbf{i}) \iff \exists\, k \in \{0, \dots, n-1\} : f_k(\mathbf{i}) \neq f'_k(\mathbf{i}) \quad .$$

Error probability

The error probability is simply the ratio of the number of inputs for which the outputs differ (i.e. at least one of the output bits differ) to the number of all inputs.

$$P_\epsilon = \frac{\#\{\mathbf{i} \in inputs \mid \mathbf{f}_C(\mathbf{i}) \neq \mathbf{f}_{C'}(\mathbf{i})\}}{\#inputs} \quad .$$

This metric formally represents a probability only under input equiprobability assumption; a more general definition that may better characterize the error in real use scenarios requires that the input probability $P_\mathbf{i}$ is defined for each input, and that the metric is evaluated as the probability of an erroneous input occurring.

$$P_\epsilon = \frac{\sum_{\{\mathbf{i} \in inputs \mid \mathbf{f}_C(\mathbf{i}) \neq \mathbf{f}_{C'}(\mathbf{i})\}} P_\mathbf{i}}{\sum_{inputs} P_\mathbf{i}} \quad .$$

This definition is unwieldly, because the task of characterizing each element of the input set rapidly becomes impractical. Section 2.2.4 details a possible approach to the estimation of the error probability.

Bit error probability

In some cases, it can be useful to constrain separately the error probability for the single output bits; in these cases, one or more bit error probabilities bounds can be defined.

$$P_{\epsilon,k} = \frac{\#\{\mathbf{i} \in inputs \mid f_{C,k}(\mathbf{i}) \neq f_{C',k}(\mathbf{i})\}}{\#inputs} \quad .$$

Similar considerations as for the previous case are relevant for the bit error probability.

2.2.3 Other metrics

The introduced error metrics are common in literature and vastly applicable. In some cases, different metrics are needed for specific applications and requirements.

Composite metrics are a combination of error magnitude and error frequency metrics. For example, the *rate significance* used in [SG11], is defined as

$$RS = \epsilon_{sig} \cdot P_\epsilon \quad .$$

Defined on	Error metric	Symbol	Formula
Error magnitude	Maximum error magnitude	$\epsilon_{\max}$	$\max_{\mathbf{i}\in inputs}\lvert O_C(\mathbf{i}) - O_{C'}(\mathbf{i})\rvert$
	Relative error magnitude	ϵ_{rel}	$\max_{\mathbf{i}\in inputs}\left\lvert 1 - \frac{O_{C'}(\mathbf{i})}{O_C(\mathbf{i})}\right\rvert$
	Average error magnitude	ϵ_{avg}	$\frac{\sum_{\mathbf{i}\in inputs}\lvert O_C(\mathbf{i}) - O_{C'}(\mathbf{i})\rvert}{\#inputs}$
	Mean squared error magnitude	ϵ_{MS}	$\frac{\sum_{\mathbf{i}\in inputs}(O_C(\mathbf{i}) - O_{C'}(\mathbf{i}))^2}{\#inputs}$
Error frequency	Error probability	P_ϵ	$\frac{\#\{\mathbf{i}\in inputs \mid \mathbf{f}_C(\mathbf{i}) \neq \mathbf{f}_{C'}(\mathbf{i})\}}{\#inputs}$
	Bit error probability	$P_{\epsilon,k}$	$\frac{\#\{\mathbf{i}\in inputs \mid f_{C,k}(\mathbf{i}) \neq f_{C',k}(\mathbf{i})\}}{\#inputs}$

Table 2.1. Summary table of error metrics

Some applications require *ad-hoc metrics*, that better describe the quality of the output, such as the PSNR or the Structural SIMilarity index (SSIM) for image processing [HZ10].

2.2.4 Evaluation of metrics

In general, a design flow could make use of both penalties and thresholds; unfortunately, for efficiency reasons, not every metric lends well to being used in both ways.

As an example, consider maximum error magnitude: evaluating the effective ϵ_{max} requires, in principle, that the golden and approximate circuits are simulated for each input vector. If we are only interested in using it as a threshold, however, we merely need to find a single input vector that violates it, or show that it doesn't exist; that is, for a given threshold $\epsilon_{max,th}$, we want to solve the Boolean SATisfiability (SAT) instance

$$\exists\, \mathbf{i} \in inputs\colon |O_C(\mathbf{i}) - O_{C'}(\mathbf{i})| > \epsilon_{max,th}$$

and, if such an input vector $\mathbf{i}$ exists, discard C'. This technique bears some similarity to the use of miters to prove properties of circuits [Bra93], but

requires some additional care to break long XOR chains that negatively impact SAT solver performances [HJ12].

While no hard rule exists, intuitively metrics defined as the maximum of a set imply existential quantification, and lends well to be encoded as SAT instances, to be used as thresolds; metrics that require counting of elements that satisfy a given formula are instead instances of the Sharp Satisfiability (#SAT) problem [Val79], that is possibly even harder than NP-complete ones [Erc+89].

For these reasons, we give in the following sections a number of alternatives for the evaluation of metrics in practical scenarios.

Specifying a workload The simplest method to avoid exhaustive computation of the error metrics is to simply restrict the computation to a representative workload; this may prove an effective choice in those cases where the circuit is not going to actually operate on all possible inputs.

Statistical estimation Simulation of the circuit may be used to estimate error metrics that require counting; as an example, in [HLB06b] the authors provide a method to determine an upper bound on the number of test patterns required to satisfy given accuracy requirements on the error probability.

Probabilistic modeling Probabilistic models of approximate adders [AHS17; Maz+16] and multipliers [Maz+19] have been proposed, which can be used to predict the circuit error probability without simulation. Moreover, being based on analytical models, they can provide some level of insight on approximation approaches, serving as tools to study the impact of the techniques.

Approximation miters Approximation miters are miters used to prove properties on error behavior of a circuit, and can be used to formally prove if an approximate circuits violates a given threshold.

A possible construction for a miter that checks a constraint on the maximum error magnitude is provided in [Češ+17]. It first evaluates the difference d between the outputs of the exact and approximate circuits;

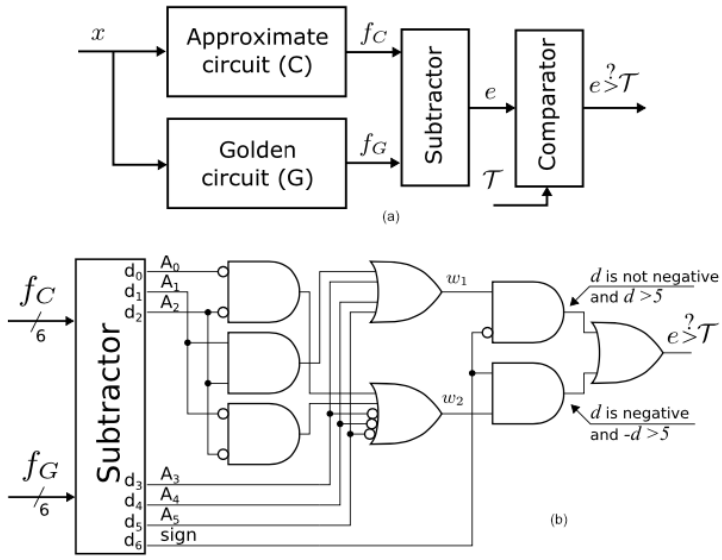


Figure 2.18. Example of approximation miter (a) and detail of the comparation (b) (from [Češ+17])

then, instead of evaluating the condition

$$|d| > \epsilon_{max,th} \quad ,$$

it evaluates the equivalent

$$((d > 0) \wedge (d > \epsilon_{max,th})) \vee ((d < 0) \wedge (\overline{d} > (\epsilon_{max,th} - 1) \quad .$$

No comparator is used in the approximation miter design, because the $\epsilon_{max,th}$ is constant for a given design and can be hard-wired.

An example of an approximation miter constructed this way is reported in Figure 2.18.

Exploiting ATPG tools Automatic Test Pattern Generation (ATPG) tools can be used to evaluate some metrics [JG05]; for example, in [SG11] a tool that generates test patterns for multiple SA faults is used to evaluate the error significance by evaluating the test vectors and checking the most

significant bit for which an error can be elicited.

Exploiting different data structures Recently exact algorithms have been proposed for the evaluation of error probability, maximum error magnitude and average error magnitude on Branch Decision Diagrams (BDDs) [Soe+16].

Unfortunately, BDDs are ill-suited to represent some circuits, for which they can be exponential in the number of nodes [Bra06]; the existance of similar algorithms for more scalable data structures, such as AIGs, remains an open question.

Chapter 3

Proposed methodology and workflow

The ALS approach proposed in this chapter is an extension of the functional approach of [Cha+16b], and was first presented in [Mor19] and extended and thoroughly evaluated in [Bar+22a].

As with any systematic approach to the synthesis of approximate variants of circuits, it's useful to conceptually separate the three main problems that are encountered: (1) how is an approximate variant obtained, (2) how is the variant evaluated w.r.t. some relevant metric and (3) how to mitigate the complexity of the design space exploration.

3.1 Generation of approximate variants

In order to tackle the problem of approximate variant generation, a number of concepts must be introduced first: (1) the data structure chosen to represent combinational circuits, (2) the concept of exact synthesis, and (3) the way it can be used for functional approximation. Once these basic parts are introduced, it can be shown how to combine them to build a catalog of possible local modifications to an original circuit.

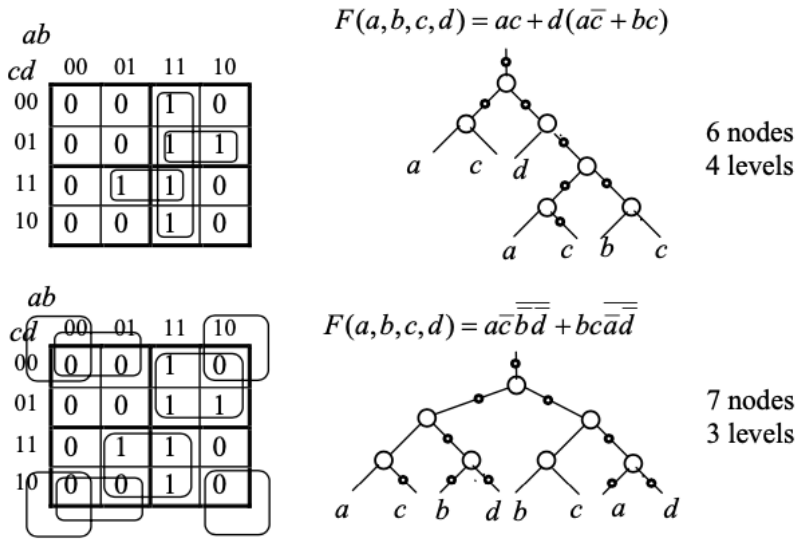


Figure 3.1. Two different AIGs for the same Boolean function (from [Mis+05])

3.1.1 And-Inverter Graphs

The data structure used to represent combinational circuits in this methodology is the AIG [Bra06]. An AIG is a Directed Acyclic Graph (DAG) with two kind of nodes: (1) Primary Inputs (PIs), with no incoming edges, and (2) AND gates, with 2 incoming edges. An edge can be marked as complemented, or not. Certain nodes are marked as POs; as it will be clear from the following sections, we can consider a single PO for an AIG without loss of generality in our tractation.

AIGs have construction time and size proportional to the size of the represented circuit; they are, however, non-canonical, i.e. the same Boolean function can be represented by more than one AIG (Figure 3.1).

Functional reduction algorithms [Mis+05] can be used to create AIGs that have no functionally-redundant nodes by construction.

3.1.2 Exact synthesis for AIGs

The exact synthesis problem is the task of finding a combinational circuit that implements a given specifications and is optimal w.r.t. some cost criteria, usually the number of nodes and/or the circuit depth. The complexity of this problem is not known; however, the Minimum Circuit Size Problem (MCSP), of which it is an instance, has been extensively studied and efficient algorithms for it are considered unlikely [MW17].

Recent advances in SAT solvers however have made it possible to solve some instances in acceptable times; exact synthesis algorithms have been implemented in the state-of-the-art *Berkeley ABC* system [BM10b; Soe+18] and synthesis flows based on it have been proposed [Haa+16].

Moreover, the expressiveness of modern SMT solvers [MDS07] make it possible to obtain a problem formulation that is simple to understand and to implement. The formulation we provide is an adaptation to AIGs of the constraint encoding used for Majority-Inverter Graphs (MIGs) in [Soe+17c].

An SMT formulation

In order to encode the exact synthesis problem for Satisfiability Modulo Theories (SMT) solving, a set of *variables* and *constraints* has to be defined. Given the Boolean function $f : \{0,1\}^m \to \{0,1\}$, we can introduce the variables $x_1, \ldots, x_m$, that represent the inputs to f, and the special variable x_0, that represents the constant 0.

Definition 3.1. An AIG of size r over variables $x_0, x_1, \ldots, x_m$ is a sequence $x_{m+1}, \ldots, x_{m+r}$ of variables that combine previous variables according to the AND functionality

$$x_i = x_{s1i}^{p1i} \wedge x_{s2i}^{p2i} \quad \text{for} \quad m < i \leq m + r \quad .$$

with $s_{1i} < s_{2i} < i$ indices, and p_{1i}, p_{2i} polarities of the input variables.

Conventionally, polarity 0 means that the variable is complemented. We say that an AIG realizes a function $f : \{0,1\}^m \to \{0,1\}$ if, chosen a polarity p, $f = x_{m+r}^p$.

Definition 3.2. *Size-optimum exact synthesis* is the optimization problem of finding the smallest r and the corresponding AIG $x_{m+1}, \ldots, x_{m+r}$ that realize a given Boolean function f.

The equivalence constraint can be encoded in *implicit* (equation) or *explicit* (truth table) form. For our application we'll work with truth tables, therefore we choose an explicit approach. In order to encode the behavior of the circuit for every possible input vector, each gate with index $i \in \{m+1, \ldots, m+r\}$ is replicated for each input assignment $t \in \{0, \ldots, 2^m - 1\}$; this replication is realized adding the three variables $a_{1i}^{(t)}, a_{2i}^{(t)}, b_i^{(t)}$ that encode the two inputs and the output of gate x_i.

The constraints for gate functionality, input connections and function semantics can be expressed as:

$$s_{1i} < s_{2i} < i \qquad i \in [n+1, n+r] \tag{3.1}$$

$$b_i^{(t)} = a_{1i}^{(t)} \wedge a_{2i}^{(t)} \tag{3.2}$$

$$s_{ci} = j \Rightarrow a_{ci}^{(t)} = b_j^{(t)} \oplus \overline{p}_{ci} \qquad c = \{1, 2\} \tag{3.3}$$

$$b_{m+r}^{(t)} = \overline{p} \oplus f(t) \tag{3.4}$$

for all $i \in \{m+1, \ldots, m+r\}$, $t \in \{0, \ldots, 2^m - 1\}$ and given r.

These constraints can be encoded in the smt-lib format [BST+10]; as an alternative, SMT solvers provide bindings for general-purpose programming language. Therefore, we can assume we have a procedure *HasAig(f, r)* that, given a function specification f and size r, returns an option [Mil18] of *none* if no AIG of size r that realizes f exists, or a pair *(x, p)*, with x structural description of the AIG, p polarity of the output; compact and efficient formats exist for representing *(x, p)*, such as the AIGER format [BHW11].

Basic algorithm for exact synthesis

The simplest algorithm *ExactAig(f)* for exact synthesis of function f makes use of the *HasAig(f, r)* procedure we described in Section 3.1.2: we can just start with $r := 0$ and iteratively check if an AIG exists for given r.

An example implementation, using the Z3 solver [DB08] Python bindings and an integer encoding for *HasAig(f, r)*, is provided in Appendix C; for efficiency reasons, a different version based on the Boolector 3.0 solver [NPB15] C bindings with a bitvector encoding is actually used in the tool imple-

Algorithm 1: ES algorithm from [Soe+17a]

```
Function ExactAIG(f):
    Input: function f
    Output: AIG x, polarity p
    r ← 0;
    while true do
        aig ← HasAig(f, r);
        if aig ≠ nil then
            return aig;
        else
            r ← r + 1;
```

mentation.

Exact synthesis of approximate variants

The same algorithm can be used to search for approximate variants of the original AIG, i.e. to search for AIGs that realize a different specification that is acceptable according to some metric. For our approach, we are interested in AIGs that realize a specification with a given Hamming distance from the exact one.

In this case, we need a procedure *HasAIG(f, r, d)* that, given function specification f, size r and maximum Hamming distance d from the truth values of the exact function, returns an option of *none* if no AIG of size at most r exists that realizes a function with Hamming distance at most d from f, otherwise a pair *(x, p)* with x structural description of the AIG, p polarity of the output.

This procedure can be implemented just like *HasAIG(f, r)*, but using different function semantics; instead of the exact semantics used in Equation 3.4 we encode the Hamming distance semantics on the function specification as

$$\exists_{\leq d}\, t \in \{0, \ldots, 2^m - 1\} \colon b_{m+r}^{(t)} \neq \overline{p} \oplus f(t) \tag{3.5}$$

where $\exists_{\leq d}\, x \in X \mid p(x)$ means that at most d distinct elements of set

X satisfy property p.

Algorithm 2: ES algorithm for approximate AIG variants

```
Function ExactAIG(f):
    Input: function f, Hamming distance d
    Output: AIG x, polarity p
    r ← 0;
    while true do
        aig ← HasAig(f, r, d);
        if aig ≠ nil then
            return aig;
        else
            r ← r + 1;
```

Refer to Section C.2.2 for an example implementation of the SMT exact synthesis algorithm for approximate variants.

3.1.3 AIG variants generation

Ideally, an algorithm for exact synthesis of approximate variants, like the one proposed in Section 3.1.2, would be a perfect fit for approximate computing, because it would enable the user to obtain a circuit that is both size-optimum and differs from the golden specification in a very controlled way.

Unfortunately, exact synthesis is a computationally intensive task, and hardly applicable to real circuits as-is. Moreover, circuits are usually specified in HDL and not explicitly in truth-table form; readily available open source tools [Wolb; BM10b] can however be used to construct an AIG from an HDL description. Therefore, we applied the proposed algorithm to *k-feasible cuts* of AIGs.

Definition 3.3. A *cut* c of node x, called *root*, is a set of nodes of the circuit, called *leaves*, such that each path from a PI to x passes at least one leaf.

Definition 3.4. A cut is *k-feasible* if the number of nodes in it does not exceed k.

Required gates	Functions	Avg. time (s)
0	10	0.00
1	80	0.01
2	640	0.03
3	3300	0.06
4	10352	0.13
5	40064	0.01
6	11058	3.09
7	32	18.38
Total	65536	

Table 3.1. Average time for exact synthesis of 4-input Boolean functions (from [Soe+17c])

Algorithms to enumerate a subset of cuts of an AIG are used for FPGA technology mapping to K-Input Look-Up Tables (K-LUTs); a number of efficient, i.e. linear [Mis+07b] or polynomial [CD94] in the size of the circuit, algorithms have been proposed and implemented in open-source synthesis flows.

More specifically, we chose $k = 4$ because

- this choice lends neatly to FPGA synthesis, where 4-LUT technologies are still currently in great use [Lew+05; Chu11], and
- there is extensive study of performance of the algorithm on 4-input functions [Soe+17c].

Moreover, the class of 4-input Boolean functions, i.e. of the functions that can be implemented by a 4-cut, is small enough that all of them can be pre-calculated and a database can be used in our synthesis flow; there are in fact $2^4 = 16$ possible input vectors, and 2^{2^4} possible ways to map output values to these vectors, that is 65536 functions.

Given a circuit C, the first step in generating variants is to compute a (functionally equivalent) 4-LUT mapping C_{lut} using one of the proposed algorithms; most synthesis tools already provide commands to do so, therefore we can assume we have such a procedure *MapToLut(C, k)*.

The next-step is building a *catalog* of AIGs for each (unique) LUT specification in C_{lut}, starting with the exact synthesis of the function,

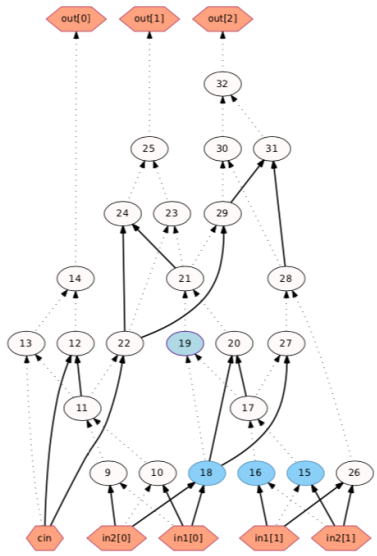


Figure 3.2. AIG for 2-bit Full Adder (from [Cha+16b])

then considering the exact synthesis of functions with increasing Hamming distance from the original specification, up to the point where the specification can be realized with 0 gates (i.e., up to the constants 0 and 1 or the input variables, complemented or not).

Let's consider, for example, the AIG in Figure 3.2; the cut $c_{19} = \{n_{15}, n_{16}, n_{18}\}$, with root in node x_{19}, determines the Boolean expression

$$x_{19} = \overline{x_{18}} \wedge \overline{\overline{x_{15}} \wedge \overline{x_{16}}} = \overline{x_{18}} \wedge (x_{15} \vee x_{16}) \quad ,$$

that has the truth table shown in Table 3.2.

For shortness, we identify such a function specification as the number in binary representation 00001110, that is the rightmost (i.e. the output) column of the truth table where the most significant digit is the entry of the last row, up to the first row, i.e. we write f to indicate its number

$$N(f) = \sum_{(x_1,\ldots,x_m)\in\{0,1\}^m} f(x_1,\ldots,x_m) \cdot 2^{\sum_{k=1}^{m} x_k \cdot 2^{k-1}} \quad .$$

x_{18}	x_{15}	x_{16}	x_{19}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Table 3.2. Truth table for the function 00001110

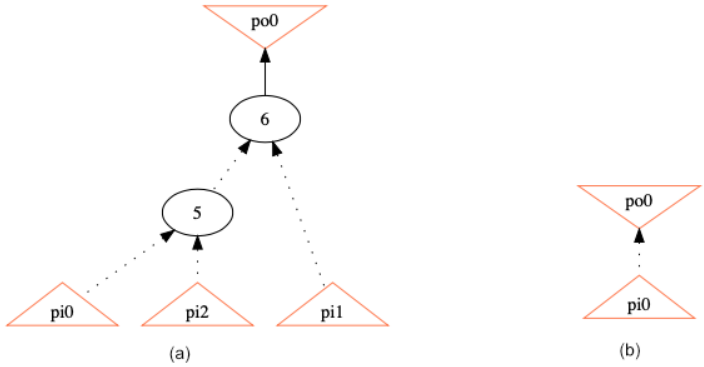


Figure 3.3. Size-optimum AIGs for (a) 00001110 and (b) 00001111

Applying *ExactAIG(f)* with *f := 00001110* yields a size-optimum AIG with 2 logic nodes; however, *ExactAIG(f, d)* with *f := 00001110* and *d := 1* yields an AIG with 0 gates, namely one whose truth table is 00001111 and whose Boolean expression is simply $\overline{x_{18}}$, as shown in Figure 3.3.

The algorithm used to create this catalog can be described in pseudocode as follows:

Where *f_ of(node)* returns the function specification of the node.

A variant C' of C can be generated simply by substituting each LUT node x in C_{lut} with an element of *catalog[f_ of(x)]*, obtaining C'_{lut}, and then rewriting each LUT node with the corresponding AIG from the catalog.

We expect C' to have less nodes than the original circuit C', because it

Algorithm 3: Catalog generation algorithm

```
Function CreateCatalog(C_lut):
    Input: C_lut: LUT-mapped circuit
    Output: catalog: LUT catalog
    catalog ← ∅;
    foreach node ∈ C_lut do
        if f_of(node) ∉ catalog then
            catalog ← catalog ∪ f_of(node);
            d ← 0;
            do
                catalog[f_of(node)][d] ← ExactAIG(f_of(node), d);
                d ← d + 1;
            while gates(catalog[f_of(node)][d]) > 0;
    return catalog;
```

has been constructed using smaller sub-AIGs. Unfortunately, the substitution also introduces an error, because the modified LUTs now implement a different function. The task of effectively trading off the correctness for a benefit in circuit size is an hard problem, as discussed in the next section.

Scalability issues

Since the catalog-generation procedure does not depend on the specific circuit being approximated, it is possible to enumerate all the k-inputs Boolean functions, i.e., all the possible configurations of k-feasible cuts, and pre-compute it. This would improve the scalability of the approach, since it would allow reusing the catalog several times, avoiding having to repeatedly solve the same ES problem while performing approximation of different Boolean functions.

However, such a generic catalog-generation procedure may be enormously time-consuming, depending on the chosen k, since the catalog size, i.e., the number of k-inputs-1-output Boolean functions, grows rapidly with k. Indeed, for each of the 2^{2^k} possible catalog entries, the set of all functions at $d \in [1, 2^k]$ Hamming distance must be recorded. Thus, the full size of the catalog is $2^{2^k} \times \sum_{i=0}^{2^k} \binom{2^k}{i}$. Anyway, a pragmatic approach to outflank such an issue would consist of incrementally building the catalog, while considering different circuits.

Concerning the SMT problem formulation, properties of the ES problem can be exploited to shorten the computational time. Indeed, the *HasAig(f, r)* ES problem has a monotonic behavior, i.e., if *HasAig(f, r)* can be satisfied for a given r, then *HasAig(f, r+1)* can also be satisfied [Soe+17a]. Thus, linear search can ideally be superseded using binary search, since the upper bound for the number of gates required by the synthesis of an arbitrary Boolean function always exists [Lup70]. Though, proving that the *HasAig(f, r)* cannot be satisfied is much more time-consuming than proving *HasAig(f, r+1)* can be satisfied [Soe+17a], hence binary search does not necessarily perform better than linear one. In addition, this observation paves the way for a heuristic approach to circumvent time-consuming *unsatisfiability* trials: if *HasAIG(f, r)* cannot be solved within a given time budget, the heuristic behaves as if such a problem is *unsatisfiable*.

3.2 Design space exploration

Let's consider once more our LUT-mapped circuit C_{lut}; we'd like to find substitutions for some of its LUT nodes that are acceptable according to our error constraints and, when rewritten as AIGs, induce some beneficial property, e.g.

- reducing the number of AND nodes,
- reducing the length of the longest path in the AIG,
- reducing the switching activity in the circuit under some workload, etc.

Unfortunately, exhaustive analysis of each alternative is unfeasible, because their number grows exponentially with the size of C_{lut}: with a conservative estimate of 5 entries in the catalog for each LUT, a circuit with as little as 30 LUTs has more than 10^{20} variants; a realistic circuit mapped on 500 4-LUTs will have more than 10^{349}.

For this reason, it's useful to state the problem in a way that lends well to be solved heuristically.

3.2.1 ALS as an optimization problem

The number of approximate variants and resulting approximate configurations grows quickly with the size of the Boolean functions which are subject to approximation. Suppose n different k-cuts have been pinpointed during partial cut-enumeration while targeting a given assertion-function, and suppose that, for each of these k-cuts, it is possible to synthesize non-trivial functions at Hamming distance up-to $m \leq 2^k$ from the original k-cut specification, i.e, each k-cut allows m different degrees of approximation. The number of approximate variants which can be defined by simultaneously approximating j k-cuts within the considered AIG is $\binom{n}{j}$, each originating m^j different approximate configurations. Hence, the total number of approximate configurations is $\sum_{i=1}^{n} m^i \times \binom{n}{i}$. At this point, the main challenge is to find k-cut replacements leading to the Pareto-optimal trade-offs between accuracy-losses and hardware overhead reduction. The latter are utterly conflicting design goals, thus recent AxC approaches from the scientific literature cope with them through Multi-objective Optimization Problem (MOP)-based Design Space Exploration (DSE) [BBM21; Bar+22b].

In this thesis, we chose to formulate the ALS problem as a MOCO problem; the required theoretical background on optimizations problems can be found in Appendix B.

The proposed formulation is the following:

- the decision variables of the problem are the LUT nodes of the circuit; we indicate them as $L = \{lut_1, \ldots, lut_n\}$, where the indices are assigned according to a topological ordering for the circuit underlying DAG;
- the variable domains are the entries of the catalog for the given specification of each LUT, that is $D_i = catalog[f\_of(lut_i)]$;
- no contraints are enforced between variables;
- the objective function $\mathbf{f}$, to be minimized, is defined according to some relevant error and penalty metrics.

The choice of the error metrics and evaluation techniques is virtually independent from the techniques used to generate alternative circuits; how-

ever, as discussed in Section 2.2.4, it may be influenced from the particular data structure used to represent the circuit.

A tool used to guide the design space exploration for the proposed methodology should ideally give a choice of different metrics in order to guide the design space exploration, as some metrics are only meaningful and suitable for specific kinds of circuits; in section 4.1, a number of different application-specific metrics are proposed. Solving MOPs is not as straightforward as solving single-objective problems. Anyway, one of the commonly adopted approaches is precisely converting MOPs to single-objective optimization problems, i.e, scalarizing the problem. This approach allows achieving good solutions when the Pareto-front is convex; otherwise, the resulting solutions may be centered around a few dominant design alternatives, not covering the whole Pareto-front [DD97]. Hence, since solving MOPs through exact algorithms, such as linear optimization or branch&bound, turns out to be very computation-intensive, we resort to the Archived Multi-Objective Simulated Annealing (AMOSA) heuristic [Ban+08a] to obtain an estimate of the Pareto front; details of the algorithm are presented in Section B.2.1. In order to converge to a suitable solution, the algorithm first initializes its *archive*, i.e. its estimate of the Pareto front, with solutions that can be randomly generated or seeded in specific points in the solution space. Each of the initial candidate solutions is firstly refined using hill-climbing, and clustering can be used to reduce the size of the archive while preserving diversity. Subsequently, the algorithm performs several refinement iterations on randomly chosen candidate solutions taken from the archive. Such candidate solutions are altered in their distinctive parameters, resulting in new candidate solutions. Pareto-dominance between the latter solutions and the archived ones is evaluated, but whether a solution is accepted as part of the archive depends not only on the dominance relationship yet on the current temperature of the "matter". Indeed, perturbations that worsen fitness values are still accepted during the first iterations, to increase diversity. Nevertheless, the probability of accepting such perturbations decreases with temperature; hence all perturbations are bound to improve fitness values of candidate solutions.

3.3 Proposed workflow

Figure 3.4 details the proposed workflow. Since circuits are typically encoded in HDL rather than AIG, a parsing step using an HDL frontend is needed to convert them to a representation that is suitable for the subsequent steps. Then, k-LUT mapping is performed to enumerate k-feasible cuts, and a catalog of variants is generated as discussed in Section 3.1.3. To do so, constraints (3.1-3.3) are encoded to solve the approximate ES problem (3.5) while increasing Hamming distance d until, due to the approximation itself, the synthesis becomes trivial. As discussed in Section 3.1.3, catalog entries are organized and stored in a database so that they can be subsequently reused. Moreover, properties of the ES problem are exploited to improve the overall scalability.

The AMOSA heuristic orchestrates the MOP-based DSE: k-cuts constitute the set of decision variables of the MOP, their indexes are assigned according to the topological ordering defined by the underlying graph, and their domain is given by catalog entries. Starting from a randomly chosen archived solution, the AMOSA selects a random cut, and replaces it using a suitable implementation taken from the catalog. Then, fitness-functions are evaluated to state the Pareto-dominance relationship between new candidate solutions and archived ones, and non-dominated candidate solutions are archived for further consideration. At the end of the DSE phase, HDL implementations of the final archived solutions are provided, allowing for the final hardware implementation.

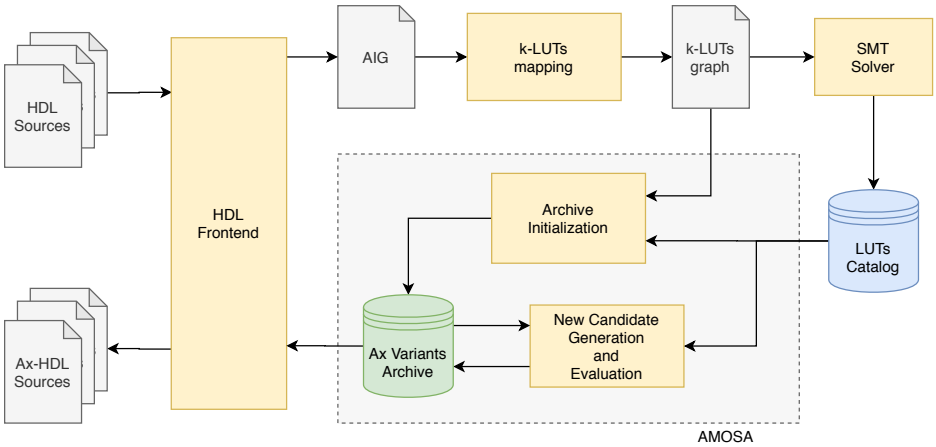


Figure 3.4. Details of the proposed workflow.

Chapter 4

Application to the ASIC technology

The approach presented in Section 3.3 has been implemented as a plug-in of the Yosys synthesis tool [WG13], using the Boolector SMT solver [NPB14] to perform the ES problem during the catalog-generation procedure. Source code and documentation are freely available at [Mor21]. The experimental campaign and case study have been originally published in [Bar+22a].

4.1 Experimental evaluation

4.1.1 Generic logic and arithmetic benchmarks

In order to evaluate the proposed methodology, it is first applied to a subset of the LGSynt91 generic logic benchmark [Yan91a], and a selection of arithmetic circuits, including various adders and multipliers. In both cases, the ES is performed targeting either 4-feasible or 6-feasible cuts, since increasing the cardinality of AIG-cuts may result in prohibitively time-consuming ES [Soe+17a]; the fitness-functions driving the optimization are the error, according to a relevant metric, and the number of AIG nodes, that is directly proportional to the circuit area; both fitness-functions are to be minimized.

For generic logic from the LGSynt91, we resort to the EP metric (4.1).

$$e_{prob}(f,\hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} [\![f(x) \neq \hat{f}(x)]\!] \tag{4.1}$$

$$[\![P]\!] = \begin{cases} 1 \text{ if P is True} \\ 0 \text{ otherwise} \end{cases} \tag{4.2}$$

Anyway, since assessing the EP using exhaustive test-patterns is prohibitively time-consuming, the latter is performed only when the amount of involved test vectors is less than 10000, e.g., when PIs is less than or equal to 13. Conversely, for circuits having more PIs, we resort to an error-sampling method and to Equation (4.3), which provides a 99.7% confidence-level for the actual EP, given the number of test patterns N_s and the measured error R_S [HLB06a]. During DSE, we set $N_s = 10000$. Furthermore, no error threshold has been set during this first experimental campaign, even though our approach allows us to impose constraints on both maximum error and maximum gate-count.

$$R_E = R_S + \tfrac{4.5}{N_S} \times \left(1 \pm \sqrt{1 + \tfrac{4}{9} \times N_S \times R_S \times (1 - R_S)}\right) \tag{4.3}$$

Concerning arithmetic circuits, we resort to the AWCE (4.4) as metric for error assessment, assigning a weight equal to the significance of the bit – i.e., the least significant bit has been assigned a weight of $2^0 = 1$, the next one a weight of $2^1 = 2$, and so forth – to each of the POs.

$$e_{awce}(f,\hat{f}) = max_{\forall x \in \mathbb{B}^n} |int(f(x)) - int(\hat{f}(x))| \tag{4.4}$$

For what pertains to hardware resources requirements, ASIC silicon die area, or the number of FPGA LUTs, power consumption and maximum clock speed should be measured for an accurate assessment. Unfortunately, this would require the synthesis and simulation of hardware designs, leading to a very time-consuming estimation process. Therefore, for both the generic logic LGSynt91 and arithmetic circuits, we resort to a model-based estimation to drive the DSE. In particular, we estimate the silicon area requirements from the number of AIG nodes, since the relationship between the latter and hardware requirements – in terms of critical path and LUTs

or standard cells – has been empirically proven in [Mis+07a]. We evaluate both the number of nodes and the depth for a given approximate configuration on Functionally Reduced And-Inverter Graphs (FRAIGs) [Mis+15].

Regarding the DSE, parameters governing the behavior of the AMOSA optimization heuristic need to be accurately set, since them may severely affect results. Furthermore, according to the "No free-lunch theorem" [WM97], it is impossible to find a group of parameters which are always effective across various optimization problems. In other words, parameters governing the AMOSA optimization must be selected on a per-circuit basis, and since multiple parameters are involved, the complex interrelationships and interactions among them mandates an iterative-refinement approach. Therefore, for each of the considered circuits, we conducted several runs while varying configuration parameters of the heuristic, observing that:

(a) to increase diversity of solutions, suboptimal candidate solutions should be allowed to be accepted during the initial stages of the annealing process; thus, the initial temperature parameter should be high enough, and 500 degrees is typically a good starting point;

(b) to allow significant refinements to be performed close to the "hardening point" of the matter – i.e., the final temperature –, the cooling factor should be high enough, while the final temperature should be very low; values in the [0.8, 0.95] and 10^{-7} degrees typically serve the purpose;

(c) in order to bound the number of comparisons to determine Pareto-dominance relationship among archived and new candidate solutions, the hard archive-size limit should be as small as possible; nevertheless, this may cause clustering to be performed too frequently, hence the soft archive-size limit should be sufficiently large; we observed that hard archive-size limit set to 50 elements, and soft archive-size limit set to 150 elements allow a good trade-off between diversity of final non-dominated solutions and computational time;

(d) iterations performed during the initial hill-climbing, and those performed during the annealing phase of the AMOSA both severely affects the quality of results provided by the optimization process; we

Circuit	adderfds	alu2	alu4	apex6	c6288	cc	cm163a	cm42	count	decod	des	frg2	i6	rot	term1	unreg	x2	x3
Comp.Time	10s	16m	7m	6m	9m	9s	5m	3s	4m	12s	7.5h	23m	11s	8m	52m	2m	7s	7m

Table 4.1. Computational-time for circuits belonging to the LGSynth91 benchmark.

observed that these parameters should be iteratively adjusted among runs, with 1500 and 1000 being good starting points, respectively.

Experiments have been performed using a personal computer equipped with a 4.9GHz 16-cores/32-threads AMD Ryzen 9550 and 128 GB of RAM (of which only a bit are actually required by our framework) running GNU/Linux. Depending on the circuit and the chosen configuration for the AMOSA heuristic, experiments took from a few minutes to several hours to complete, as reported in Table 4.1.

At the end of the DSE, we synthesized resulting HDL implementations targeting the 45 nm standard-cell library, in order to measure actual hardware requirements. Due to the heuristic nature of technology mapping, solutions belonging to the Pareto-front defined by EP and gate count minimization may not belong to the one defined by EP and circuit area minimization. Therefore, a further non-dominance selection step is performed, in order to select non-dominated solutions w.r.t. EP and circuit area.

Figure 4.1 plots silicon-area against EP for circuits from the LGSynth91 benchmark: the red star denotes the exact circuits, while blue dots denote approximate configurations resulting from our workflow. As the reader can observe, a general decreasing trend for silicon area can be observed while the error increase, and, depending on the specific error-resiliency of the circuit being considered, our technique allows achieving significant savings. Furthermore, although we do not report plots for brevity's sake, a similar trend can be observed for power consumption.

Figure 4.2, instead, plots the silicon-area against AWCE for various 8-bits and 16-bits arithmetic circuits, including array-tree multipler (ATM), Dadda-tree multipler (DTM), Wallace-tree multipler (WTM), carry-skip adder (CSkA), ripple-carry adder (RCA) Han-Carlson adder (HCA) and carry-lookahead adder (CLA). The x-axis for Figure 4.2 is in semilogarithmic scale. Once again, the red star is the reference – i.e., the exact circuits – while the blue dots denote approximate configurations resulting from our

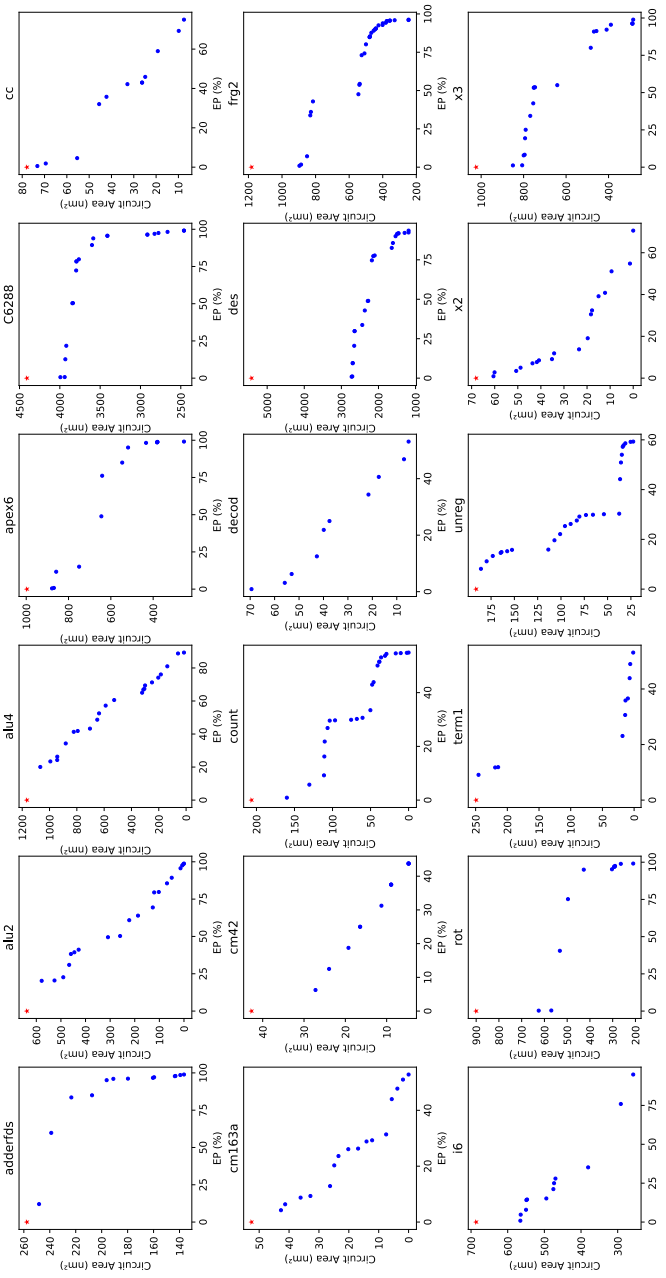


Figure 4.1. Experimental results for LGSynt91 benchmark circuits.

Circuit	atm8	dtm8	wtm8	cska16	rca16	hca16	hca8	rca8	rcl8
Time	22h	21h	18h	21m	19m	23m	5m	7m	6m

Table 4.2. Computational-time for circuits belonging to the LGSynth91 benchmark.

workflow. A general decreasing trend for silicon area can be observed in these results too, denoting the overall approach allows effectively introducing approximation also within arithmetic circuits, resulting in significant savings as the allowed error increases. Concerning the run-time of experiments involving arithmetic circuits, they are reported in Table 4.2.

4.1.2 Comparison with previous works

Although it trivially replaces k-feasible cuts using stuck-at faults with constant-zero, the AIG-rewriting based approximation technique proposed in [Cha+16a] bears similarities to the one we discussed in Section 3.1.3; it's therefore natural to ask how the catalog-based AIG-rewriting approximation techniques we introduced behaves w.r.t. trivial cut replacement, e.g., stuck-at fault injection. Concerning the evaluation of their technique, also authors of [Cha+16a] considered a subset of the LGSynt91 benchmark and arithmetic circuits, reporting synthesis results as provided by the Berkeley-ABC tool [BM10a] while targeting the *mcnc.genlib* library. Therefore, to perform a fair comparison, we resort to results discussed above and to the Berkeley-ABC tool [BM10a] while targeting the mentioned cell library. As shown in Figure 4.3, the presented approach allows better trade-offs between error and circuit area requirements w.r.t. the approach from [Cha+16a], and the same also applies to results depicted in Figure 4.4, that compares our method and the one from [Cha+16a] while targeting arithmetic circuits – specifically, 8-bits ATM, DTM and WTM. The x-axis of Figure 4.4 is in logarithmic scale.

The approach proposed in [Cas+21] is also based on circuit transformation, which are performed at the netlist level, i.e., after standard-cell mapping. Set an error threshold, the netlist of a given circuit is transformed either by gate pruning or by replacing gates with wire connections, until error constraints are violated due to the introduced approximation. Then, hardware requirements, e.g., silicon area, are measured through ac-

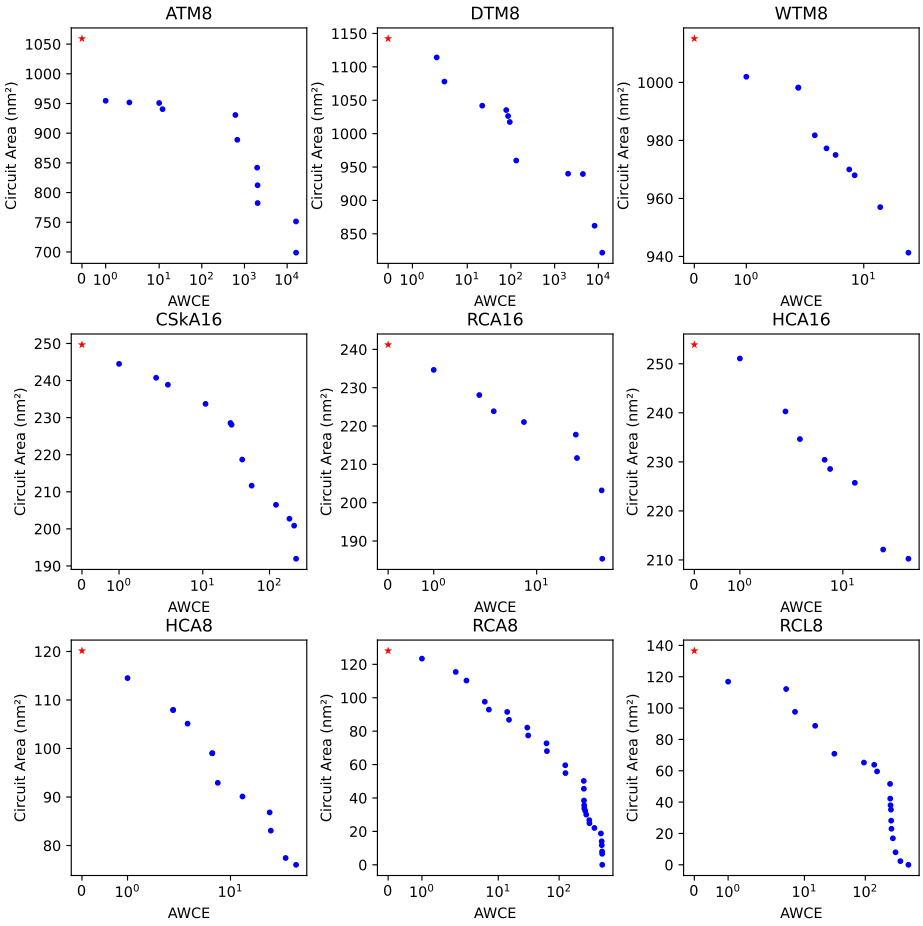


Figure 4.2. Experimental results for arithmetic circuits. The x-axis is in semilogarithmic scale.

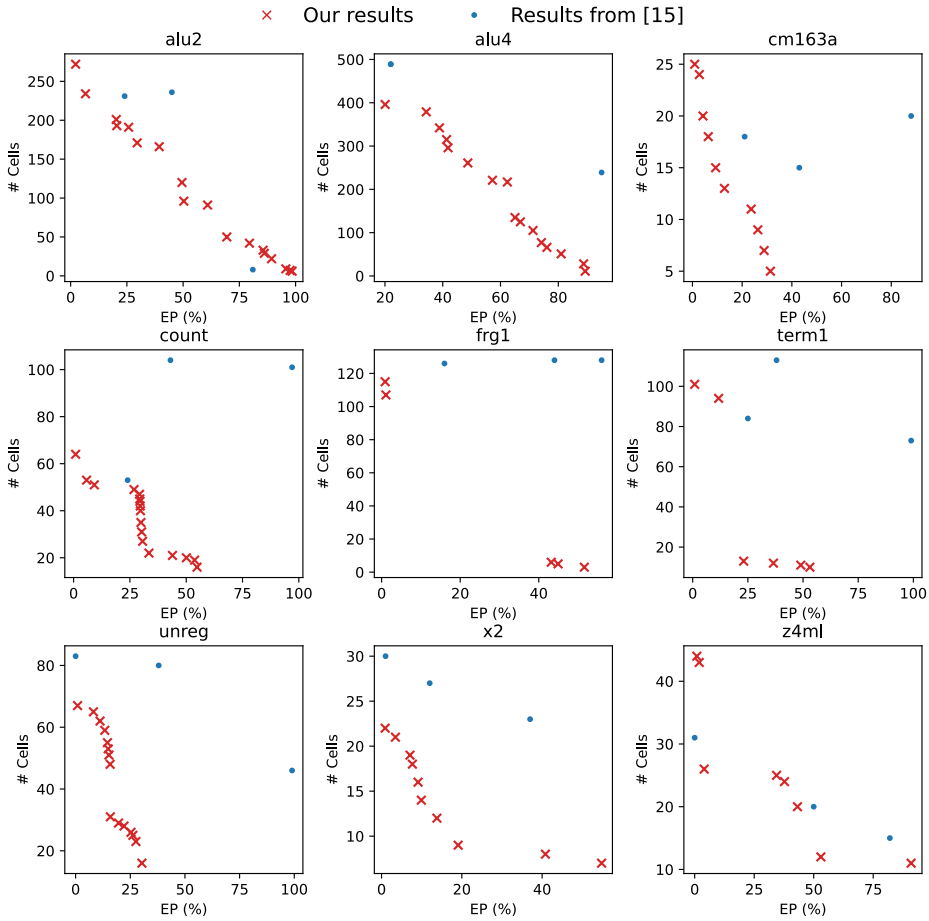


Figure 4.3. Comparison with results from [Cha+16a] while using the EP on LGSynt91 benchmark circuits.

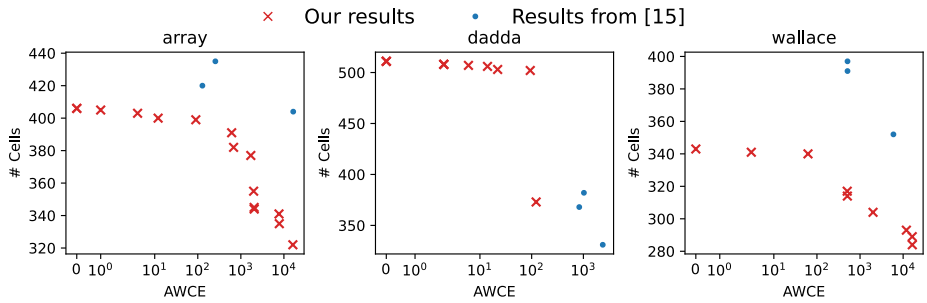


Figure 4.4. Comparison with results from [Cha+16a] while using the AWCE on arithmetic circuits. The x-axis is in semilogarithmic scale.

tual synthesis. Approximate circuits are generated using two alternative search approaches, i.e., the probabilistic pruning (PP) from [Lin+11], and a novel approach, namely the *in-out*. The latter

(i) explores all nodes which depend on constant PIs, and prunes each dependent node while checking for accuracy, and recovers pruned nodes which cause the error constraint to be violated; and

(ii) explores dependencies of POs, pruning nodes while checking for accuracy, and recovers pruned nodes which cause the error constraint to be violated; if a PO is no more in functional dependency with any of the PIs, due to the introduced approximation, it is assigned with a constant value.

The method is evaluated on several arithmetic circuits, including 16-bits Kogge-Stone adder (KSA), CLA and RCA, and 8-bits WTM, targeting the 15 nm NanGate standard-cell library. For what pertains to error, two different metrics are considered, i.e., the AWCE (4.4), and the MED (4.5), where $O_x = |int(f(x)) - int(\hat{f}(x))|$ and $P(O_x)$ is the probability of occurrence of O_x.

$$e_{med}(f, \hat{f}) = \sum_{x \in \mathbb{B}^n} O_x P(O_x) \tag{4.5}$$

For comparison purpose, we repeat experiments from [Cas+21], and perform our workflow on the mentioned arithmetic circuits, targeting the same technology library for synthesis purpose. Figure 4.5 and Figure 4.6

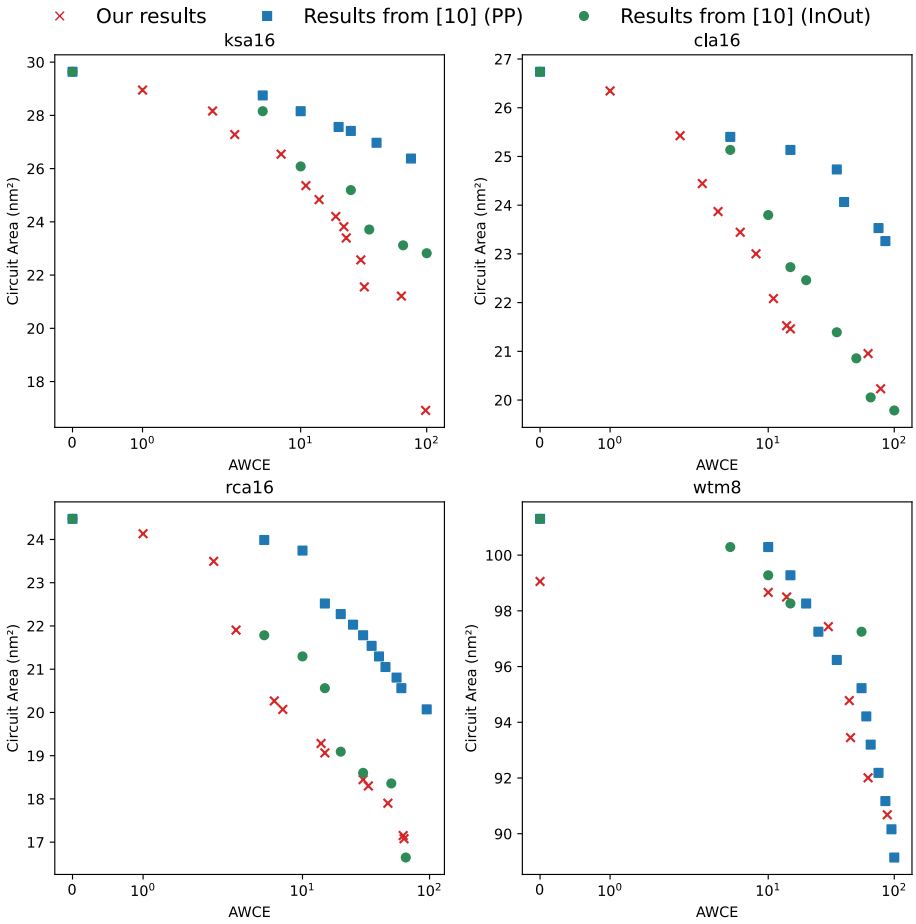


Figure 4.5. Comparison with results from [Cas+21] while using the AWCE.

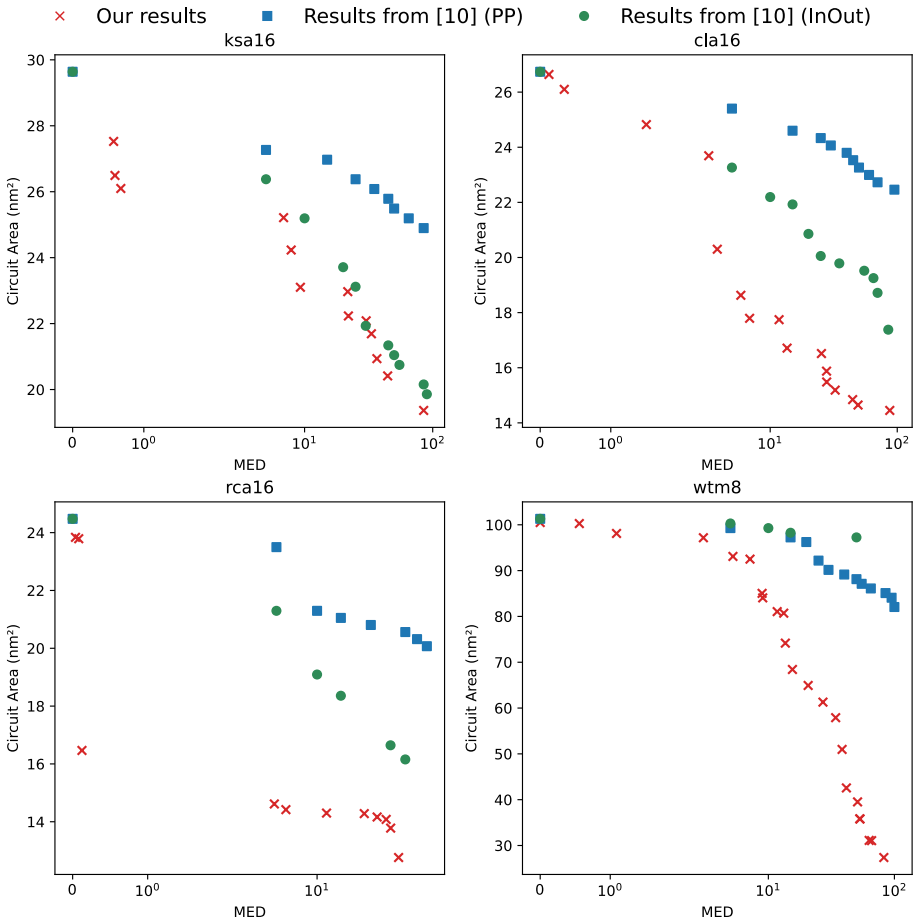


Figure 4.6. Comparison with results from [Cas+21] while using the MED.

report the comparison while considering the AWCE and MED as error metric, respectively.

In conclusion, while, on one hand, our approach require addressing the ES problem to compute non-trivial cut replacements, on the other hand it allows for a gradual introduction of approximation, hence resulting in qualitatively better trade-offs between error and hardware requirements. Conversely, naive cut replacement, such as stuck-at constant, allows a quicker approximation, though it leads to worse results.

4.2 Case study: the JPEG compression

As aforementioned, AxC is suitable for being exploited in the field of image processing, since imperceptible reduction of image quality can possibly lead to important computational resources savings. Therefore, we provide an actual case-study concerning the design of hardware accelerators for JPEG image compression exploiting approximate components resulting from our approach. In order to provide the reader with the needed technical background, we first report the state-of-the-art concerning JPEG compression, in particular we briefly discuss the DCT computation using approximate algorithms. Indeed, the latter is the most demanding step of the JPEG algorithm, and most of the research mainly focused on it [AKL18], that is known to have $O(N^2)$ complexity, and requires resource-intensive functional units, such as floating-point arithmetic modules.

Approximate DCT

Cosine coefficients required by the DCT can be scaled and rounded such that floating-point operations can be superseded by integer ones: the resulting algorithms are significantly faster, and they find extensive use in practical applications. However, integer multiplication is still complex and resource intensive; thus, many low-complexity multiplier-less algorithms have been proposed [BAS08; BC12; Pot+14]. They all avoid computing DCT transformed coefficients separately or iteratively, rather they extensively resort to matrix algebra and its properties.

The DCT-transform of an input image tile X, which is a 8×8 matrix, is given by $F = C \cdot X \cdot C'$, where C contains the cosine function values at

the needed frequencies, and it is referred to as *DCT matrix*. Multiplier-less algorithms first split the C matrix into two sub-matrices, namely T and D, as reported in Equation (4.6). T contains only the values $\{0, \pm\frac{1}{2}, \pm 1, \pm 2\}$ and it is orthogonal – i.e. $T' = T^{-1} \Rightarrow TT' = T'T = I$, where I is the identity matrix – while D is a diagonal matrix consisting of values in the $[-1, 1]$ range, with $\{\frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{8}}\}$ being typical values.

$$F = C \cdot X \cdot C' = D \cdot (T \cdot X \cdot T') \cdot D \tag{4.6}$$

The multiplication of D in (4.6) still requires floating-point operations; nevertheless, resorting to properties of diagonal matrices, the integer null-multiplicative part $T \cdot X \cdot T'$ can be isolated from floating-point operations required by D, as reported in Equation (4.7), where $\circ$ is the Hadamard product, i.e., an element-wise multiplication.

$$F = T \cdot X \cdot T' \circ (diag(D) \cdot diag(D)'), \tag{4.7}$$

Afterwards, floating-point operations can be performed outside the DCT, and embedded into the JPEG quantization step, as shown in Equation (4.8), where $\hat{Q}$ is the *complete quantization matrix* and the $\oslash$ operator is the Hadamard division, i.e., an element-wise division.

$$\begin{aligned} F_Q &= \lceil F \oslash Q \rfloor = \lceil T \cdot X \cdot T' \circ (diag(D) \cdot diag(D)') \oslash Q \rfloor \\ &= \lceil T \cdot X \cdot T' \circ \hat{Q} \rfloor = \lceil (T \cdot (T \cdot X')') \circ \hat{Q} \rfloor \\ \hat{Q} &= (diag(D) \cdot diag(D)') \oslash Q, \end{aligned} \tag{4.8}$$

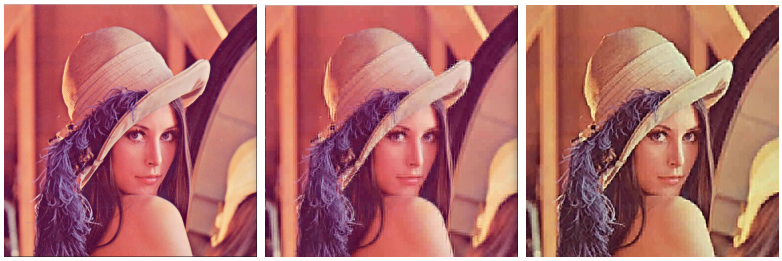
Besides allowing the use of integer arithmetic for the calculation of coefficients, Equation (4.8) also allows computing the two-dimensional DCT using the one-dimensional DCT transform twice, reducing the computational complexity from quadratic to linear.

Implementing approximate hardware-accelerators

By exploiting Equation (4.8) to compute the DCT, and resorting to results discussed in Section 4.1.1, we selected an approximate configuration for the 16 bits Han-Carlson adder exhibiting AWCE of 31, and requiring 212 nm^2 against 253 nm^2 required by the exact adder, in order to imple-

ment a DCT hardware accelerator using the BAS08 [BAS08] algorithm. We synthesized both the non-approximate BAS08 hardware accelerator – i.e., the one using exact adders – and the approximate accelerator – i.e., the one using approximate adder – while targeting the 45 nm standard-cell library. While the former requires approximately 2400 nm^2 to be implemented, the latter only requires approximately 2040 nm^2, providing approximately 16% savings.

Figure 4.7 shows the standard JPEG-compressed image of Lena and Baboon on the left, i.e., compressed using full DCT employing floating-point representation, the same images compressed using multiplier-less DCT with exact adder, and the same images compressed employing multiplier-less DCT based on the minimum-error approximate configuration of the selected adder on the right. The latter exhibit 23.34 and 20.61 PSNR, respectively, when compared against standard JPEG-compressed images (the higher the PSNR, the higher the image quality).



(a) Visual test with Lena

(b) Visual test with Baboon

Figure 4.7. Visual test. JPEG compressed images in which DCT is computed using floating-point arithmetic are on the left; while, in the center, compression is performed using the BAS08 algorithm to calculate DCT (without further approximation). Finally, on the right, the BAS08 algorithm is further approximate using approximate adders.

Chapter 5

Application to the FPGA technology

Given their increasing popularity, both for prototyping and for low-volume production of digital electronic circuits, an interesting question that sparks from the methodology presented in Chapter 3 is if it is applicable *as-is* to FPGA technologies.

Intuitively, by simply selecting approximate variants of LUTs in the original circuit, we would obtain a different circuit that (1) does not produce the exact output for some of the inputs and (2) has the same number of LUTs, unless for some of them the catalogue entry that is chosen is a constant 1 or a constant 0. However, as the entire circuit is rewritten in a new AIG form, and the k-cut selection is repeated on the approximate variant, it can provide advantages in area occupation. Moreover, the proposed methodology can provide significant power savings in the FPGA technology.

In order to motivate this result, a simple power model for the LUT components of the FPGA fabric is introduced; experimental findings are then reported. The results presented in this chapter have been originally published in [Bar+24].

5.1 The power dissipation model of LUTs

In this section, we aim to analyze the correlation existing between AIG node count and power consumption of approximate circuits. To this aim, two different strategies are possible. On one hand, we could rely on simulation-based tools that considering an FPGA configuration (namely bitstream), a specific FPGA device and a user-define workload (namely, a test-bench) can estimate power consumption by stimulating the input signals and executing a low-level device simulation. On the other hand, as we will discuss later, we could resort to a LUT model, that is generic and independent of a specific FPGA fabric, from which we could estimate the switching activity and get the power consumption avoiding costly simulation. Furthermore, instead of providing to each circuit a proper workload, we can consider each time a worst-case scenario.

5.1.1 Switching activity estimation

The power consumption for FPGA includes two terms, i.e., the static and the dynamic power. The former is the power from transistor leakage on all connected voltage rails and the circuits required for the FPGA to operate normally. It is highly operating-temperature dependent, yet a function of the manufacturing process and supply voltage, but it is not correlated to the design configured onto the device fabric. Conversely, the dynamic power is the power of the user design, due to the input data pattern and the design internal activity. This power is instantaneous and varies at each clock cycle, it depends on voltage levels, logic, and routing resources used (including I/O terminations, clock managers, and other circuits that need power when used).

Anyway, it can be roughly estimated as

$$P = V_{dd}^2 \cdot f_{clk} \cdot \sum_{n_i} C_i \cdot E_{sw}(n_i) \tag{5.1}$$

where C_i is the capacitance of the node n_i, V_{dd} is the supply voltage, f_{clk} is the clock frequency, and $E_{sw}(n_i)$ is the average number of output transitions per time-unit $T = \frac{1}{f_{clf}}$ at node n_i, i.e., the SwA.

As it is easy to foresee, reducing the resource overhead required by a

given circuit – i.e., the number of LUTs a circuit requires, as provided by our approach – implies a reduction in the dynamic power consumption. But experimental results, shown in the previous section, evidences that there is a further contribution to power savings, that clearly comes from SwA of involved LUTs reduction, being operating conditions – i.e., V_{dd}, f_{clk}, – out of the scope of our approach and left unchanged w.r.t. original circuit specification.

Estimating SwA, i.e., $E_{sw}(n_i)$, requires determining $p_0(n_i)$ and $p_1(n_i)$, i.e., the probability of the output signal at each node n_i being either logic-0 or logic-1, respectively. Under the zero-delay model, the $E_{sw}(n_i)$ can be estimated using Equation (5.2) from [Ped96]. Equation (5.2) also supposes that the n_i node can assume only the two mentioned complementary logic values; hence, $p_0(n_i) = 1 - p_1(n_i)$.

$$E_{sw}(n_i) = \frac{p_0(n_i) \cdot p_1(n_i)}{p_0(n_i) + p_1(n_i)} = p_0(n_i) - p_0(n_i)^2 \tag{5.2}$$

In the case of FPGA, determining $p_0(n_i)$ requires characterizing the activity of inputs and output signals, resorting, for instance, to a model such as the one from [Ale97].

Essentially, a K-LUT takes K input signals $S = \{s_0, \cdots, s_{K-1}\}$ that allow selecting, as output, one between 2^K configuration bits from $X = \{x_0, \cdots, x_{2^K-1}\}$, as defined by the technology mapping $T_m(S, X)$.

Although the actual implementation may be quite different, the model from [Ale97] states K-LUTs can be modeled as fully balanced binary trees of height $K-1$, 2^K-1 internal nodes corresponding to 2-to-1 multiplexers, and 2^K leaves representing configuration bits, as depicted in Figure 5.1.

For an accurate estimation of the E_{sw}, all the configuration bits, not just the one being selected as output, have to be considered, since even those that do not actually reach it partially propagate towards the output anyway contribute to the SwA, affecting the frequencies with which each configuration bit passes through multiplexers. Thus, the model computes the actual frequency with which the configuration bits traverse each node of the tree based on frequencies f_i with which the $i-th$ configuration bit

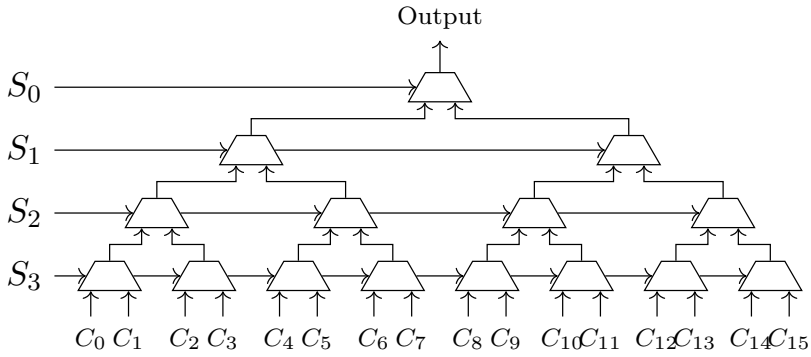


Figure 5.1. Model of a 4-LUT as a fully balanced binary-tree of 2-to-1 multiplexers.

is selected as output, as in (5.3).

$$F = \{f_i \in \mathbb{R} : f_i \in [0,1] \wedge \sum_{i=0}^{2^K-1} f_i = 1\} \tag{5.3}$$

The frequency whereby the configuration bits traverse each node of the tree can be computed using Equation (5.4). The $f_{\ell,L}$ is the sum of the frequencies of the leaves on the left-hand side of the 2^ℓ trees rooted at level ℓ, while $f_{\ell,R}$ is the sum of the frequencies of the leaves on the right-hand side of the 2^ℓ trees rooted at level ℓ. Clearly, $f_{\ell,L} + f_{\ell,R} = 1$.

$$f_{\ell,L} = \sum_{j=0}^{2^\ell-1} \sum_{i=0}^{2^{K-\ell-1}-1} f_{j\cdot 2^{K-\ell}+i} \tag{5.4}$$

To compute $p_0(n_i)$, we resort to Equation (5.5) from [Ale97], that allows recursively computing $p_0(n_i)$ for each internal node of the tree based on the same but computed at child nodes, plus $f_{\ell,L}$ and $f_{\ell,R}$. In such an Equation, $L(n_i)$ and $R(n_i)$ respectively denote the left-hand-side and right-hand-side child of n_i, while $f_{\ell,L}$ and $f_{\ell,R}$ denote the left-hand side and right-hand side frequencies of the tree at level ℓ, as given by (5.4).

The root-node is at level 0.

$$p_0(n_i) = \begin{cases} p_0\left(L(n_i)\right) f_{\ell_i,L} + p_0\left(R(n_i)\right) f_{\ell_i,R}, \\ \qquad\qquad\qquad\qquad i \in \left[0, 2^K - 2\right] \\ \left(x_j = 1 \rightarrow 0\right) \wedge \left(x_j = 0 \rightarrow 1\right), \\ \qquad i \in \left[2^K - 1, 2^{K+1} - 1\right], j \in [0, 2^K - 1] \end{cases} \tag{5.5}$$

The first (top) case applies to all nodes which index ranges in $\left[0, 2^K - 2\right]$, i.e., all nodes of the tree but leaves. For such nodes, $p_0(n_i)$ – i.e., the probability of the output signal at node n_i being logic-0 – is recursively computed as the sum of the probability of the output at left and right children nodes being logic-0 – viz. $p_0(L(n_i))$ and $p_0(R(n_i))$, respectively – each multiplied to the frequency whereby the configuration bits traverse each node of the tree until the considered child of n_i – i.e., $f_{\ell,L}$ and $f_{\ell,R}$, respectively, as computed by Equation (5.4). Conversely, the second (bottom) case applies to nodes with index in $\left[2^K - 1, 2^{K+1} - 1\right]$, that are leaves of the tree, that matches configuration bits of the LUT. In this case, the probability of the output signal at node n_i simply matches configuration bits; hence, it coherently collapses to $p_0(n_i) = 0$ if the corresponding configuration bit is 1, else it equals to 1. Hence, $p_0(n_i)$ directly depends on the configuration bits and left-hand side and right-hand side frequencies values as computed by Equation (5.4). The SwA for a whole K-LUT, given its configuration bits, F, can be computed using (5.6), that only contemplates internal nodes of the LUT.

$$E_{sw} = \sum_{i=0}^{2^K-2} p_0(n_i) - p_0(n_i)^2 \tag{5.6}$$

It is worth noticing that, there are more than one $T_m(S, X)$ that satisfies the proper mapping for a given boolean function and, depending on which one is selected during the synthesis, it exhibits different SwA. The selection of mapping solution with less SwA is due to the synthesis tool, namely Xilinx Vivado in our case, involving a signal reordering of the K-LUT. It goes that SwA optimization through signal reordering is out of our scope and not considered by the proposed approach.

5.2 Experimental evaluation

For evaluation purposes, a re-implementation of the original tool [Mor21] was used; the tool is released under GNU GPL3 open-source license.

The tool takes the HDL implementation of the concerned circuit, and generates the corresponding AIG representation. Then, k-cuts enumeration is performed by leveraging fabric-independent k-LUT mapping, i.e, we do not consider any vendor-specific FPGA fabric at this stage. The catalog generation takes place as discussed in Section 3.1.3, and, as in Chapter 3, catalog entries are organized and stored in a database so that they can be subsequently reused.

The AMOSA heuristic [Ban+08b] orchestrates the DSE: decision variables of the problem are k-LUT within the mapped circuit, and their domain is given by catalog entries. A LUT within the mapped circuit is randomly selected, and replaced using a suitable entry taken from the catalog; then, fitness-functions are evaluated to state the Pareto-dominance relationship between new candidate solutions and archived ones. While dominated candidate solution may be discarded by the heuristic, non-dominated ones are archived for further consideration. At the end of the DSE phase, the tool provides the HDL implementation for each non-dominated archived solutions, allowing for the final hardware implementations and measurement of the actual FPGA resource-requirement.

5.2.1 Generic logic and arithmetic benchmarks

For evaluation purpose, we considered a subset of the LGSynt91 generic-logic benchmark [Yan91b]. Furthermore, since they are building blocks for larger applications, such as Artificial Neural Networks (ANNs) [Mra+19; Ans+20; Tas+20; Ahm+23; Kim+22], we also consider several arithmetic circuits, including various adders and multipliers. Specifically, we considered arithmetic circuits from the ArithsGen [KM22] and from the Arithmetic Module Generator [HA22; UNT17] frameworks, as well as circuits from the EPFL Combinational Benchmark Suite [AGM15].

We performed the ES of 4-feasible or 6-feasible cuts within the mentioned circuits, since a higher cardinality of AIG-cuts prohibitively increases the time needed to accomplish the ES [Soe+17b]. The fitness-functions driving the optimization are error and hardware requirements,

both to be minimized.

Pertaining to error metrics, as in [Bar+22a], we resort to the EP metric (4.1) when dealing with the LGSynth91, where the $[\![\cdot]\!]$ notation denotes the Iverson bracket (4.2). For what pertains to arithmetic circuits, we resort to the AWCE (4.4) as a metric for error assessment, assigning weights to POs according to their significance, i.e., the least significant bit is assigned a weight of $2^0 = 1$, the next one a weight of $2^1 = 2$, and so forth.

Regarding hardware-resource requirements, the number of FPGA LUTs, the power consumption and the maximum clock speed should be accurately measured for an accurate assessment. Anyway, this would require FPGA synthesis to be performed, leading the computational-time of DSE to be unsustainable. Therefore, as in [Bar+22a], we resort to a model-based estimation to drive the DSE. We estimate both the number of FPGA LUTs and latency from the AIG representation of circuits, since the correlation between hardware requirements and the number of nodes and depth of the AIGs has been empirically proven in [Mis+07c].

Speaking of the AMOSA configuration, we address it on a per-circuit basis, resorting to advice from [Bar+22a]. As a summary of a typical configuration, when generic-logic circuits from LGSynth91 are concerned, we let the cooling factor to vary in the [0.8, 0.9] range, the final temperature of the matter varying in the $[10^{-7}, 1]$ range, the hard and soft archive-size limit vary in the [50, 15] and [100, 300] ranges, respectively, the initial hill-climbing and annealing iterations in the [100, 350] and [250, 750] ranges, respectively. Having said that, depending on the circuit and the chosen configuration for the AMOSA heuristic, experiments involving such circuits take from a few minutes to several hours to complete.

For what pertains to arithmetic circuits, we observed that, as far as adders are concerned, the same configuration allows achieving good trade-offs between diversity of final non-dominated solutions and computational time. Such a configuration consists of a cooling factor varying in the [0.8, 0.95] range, the final temperature of the matter varying in the $[10^{-3}, 10^{-1}]$ range, hard and soft archive-size limits set to 100 and 250 elements, respectively, while the initial hill-climbing and annealing iterations are set to 200 and 350 respectively. On the other hand, when multipliers are concerned, the increased complexity of the problem – in terms of number of decision variables – requires more effort to be spent during DSE. We came up

Circuit	alu2	alu4	apex6	C6288	count	frg2	i5	i6	rot	term1	unreg	x3
Comp.Time	16m	7m	6m	9m	4m	23m	13s	11s	8m	52m	2m	7m

Table 5.1. Computational-time for circuits belonging to the LGSynth91 benchmark.

Circuit	CLA16	CSelA16	CSkA16	HCA16	KSA16	RCA16	HCA8	RCA8	RCL8	ATM8	DTM8	WTM8	sine
Comp.Time	≈20h	≈20h	≈26h	≈25h	≈20h	≈21h	≈13h	≈12h	≈13h	≈66h	≈60h	≈62h	≈210h

Table 5.2. Computational-time for circuits arithmetic circuits.

with the following configuration: the cooling factor is set to 0.95, the final temperature of the matter set to 10^{-7} degrees, the hard and archive-size limits set to 150 and 300 elements, respectively, initial hill-climbing and annealing iterations set to 550 and 750, respectively. As it is easy to foresee, the more effort is spent during DSE, the more it lasts; consequently, while experimenting on adders typically requires a few hours, experiments involving multipliers may require a few days to complete.

Results, as well as computational time, are discussed in the following.

Table 5.1 and Table 5.2 report the computational time that experiments tool to complete, respectively, for generic logic and arithmetic circuits.

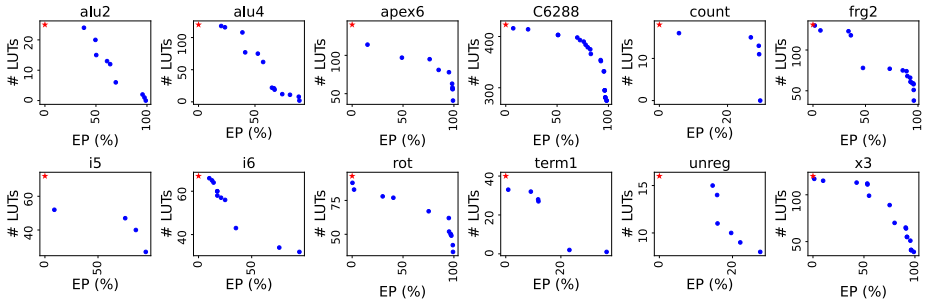
At the end of the DSE, we targeted the Xilinx xc7a35ticsg324-1L Artix-7 FPGA to measure actual hardware requirements of resulting circuits. Synthesis was performed by Xilinx Vivado 2021.01 with default settings and disabling DSPs.

Figure 5.2a, Figure 5.2b plot the number of required FPGA LUTs and power consumption against EP for circuits from the LGSynth91 benchmark.

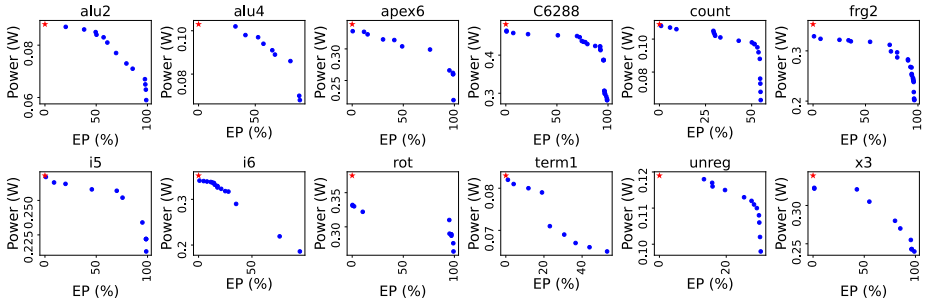
Figure 5.3a and Figure 5.3b, instead, plots resource requirements, in terms of the number of required FPGA LUTs and power consumption against AWCE for various 8-bits arithmetic circuits, including ATM, DTM, WTM, CSkA, RCA HCA and CLA, which have been generated while resorting to the Arithmetic Circuit Generator for Hardware Accelerators (ArithsGen) [KM22].

Last, Figure 5.4a and Figure 5.4b plots resource requirements for the circuits from the EPFL Combinational Benchmark Suite [AGM15].

The red star denotes the exact circuit, while the blue dots • denote approximate configurations resulting from DSE. There may be solutions

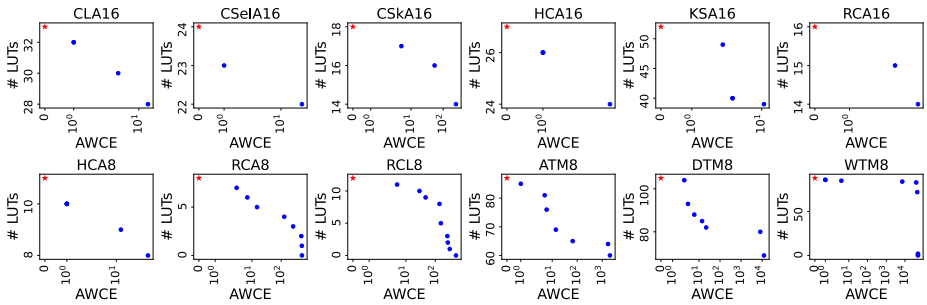


(a) Amount of LUTs

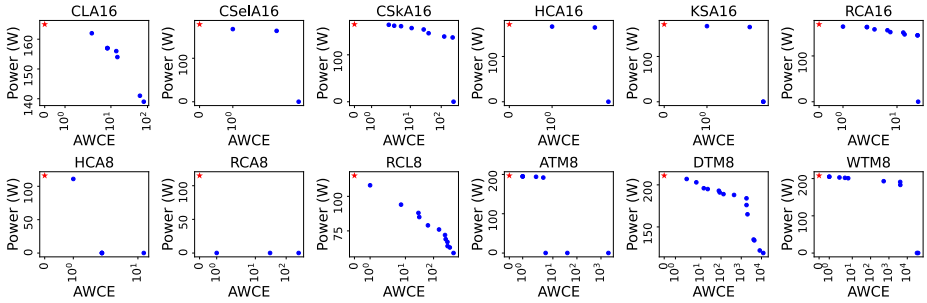


(b) Power consumption

Figure 5.2. FPGA resource requirements for generic-logic circuits from the LGSynth91 benchmark



(a) Amount of LUTs



(b) Power consumption

Figure 5.3. FPGA resource requirements for arithmetic circuits. The x-axis is in semilogarithmic scale.

(a) Amount of LUTs (b) Power consumption

Figure 5.4. FPGA resource requirements for the for the *sin* circuit, from the EPFL Combinational Benchmark Suite [AGM15]. The plots are in semilogarithmic scale.

that have the same error and gate-count (they are in a non-dominance relationship with each other) coming from the DSE that, when synthesized to FPGA still have the same error, of course, but they require different amount of LUTs to be implemented, or they consume different power. Hence, they are no more in non-dominance relationship; therefore, the dominated ones are actually discarded and not plotted. Hence, any point in Figure 5.2a exhibiting the same error as in Figure 5.2b may not refer to the same approximate configuration. This, of course, also applies to Figure 5.3a and Figure 5.3b.

As for the area overhead, it is clear that approximate circuits require less FPGA LUTs on the error increasing since, as we select approximate solutions minimizing AIG node-count. In the same way, power consumption takes advantage of the approach as it decreases on the introduced error. But, as one can notice, Figure 5.2b reports more non-dominated solutions than Figure 5.2a as well as Figure 5.3b and Figure 5.3a. This implies that given two different approximate configurations exhibiting the same error and requirements in terms of FPGA LUTs, one of these has lower power consumption. This phenomenon can certainly also happen with ASIC technology, Anyway, at a first glance, it appears to be occurring very frequently, contrary to the experimental result reported in Section 4.1.1 based onto ASIC implementations.

This experimental observation led us to investigate how two different approximate implementations could exhibit two power consumption values against the same amount of FPGA LUTs, i.e., how approximate variants of a given circuit exhibit different values of dynamic power consumption despite requiring the same amount of LUTs. We provide an in-depth analysis of FPGA dynamic power consumption based on LUTs model in Section 5.1.

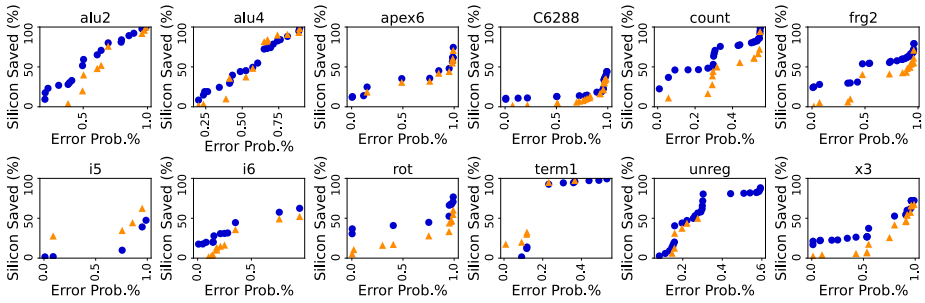
5.2.2 Comparison between ASIC and FPGA technologies

As discussed before, almost all the AxC approaches operate at the transistor or gate-netlist level, and allow for large silicon area and power savings while targeting ASIC. Nonetheless, due to the architectural differences between target technologies, savings achieved are far modest while targeting FPGA [Pra+20; Ull+22b; Ull+22a]. Though, since the previous Section empirically prove the AxC technique from [Bar+22a] can be effectively adopted for both FPGA-based systems, besides ASIC ones, it is interesting to compare results the technique provides, varying the target implementation, in terms of silicon and power savings. Figure 5.5 and Figure 5.6 report such a comparison for generic logic circuits and arithmetic ones. The blue bullets ● denotes savings induced in ASIC implementations, while the orange triangles ▲ denote savings for FPGA technology. Except in sporadic cases, the results shown in the figures mentioned above confirm what is known in the literature: applying the same approximation technique does not produce the same gain when considering different implementations. However, the gap is not much, especially for arithmetic circuits. And for some circuits the gain on FPGA is greater, opening the way for future investigations about which error metrics or properties a circuit must possess in order for it to be approximated with the same profit on both ASIC and FPGA.

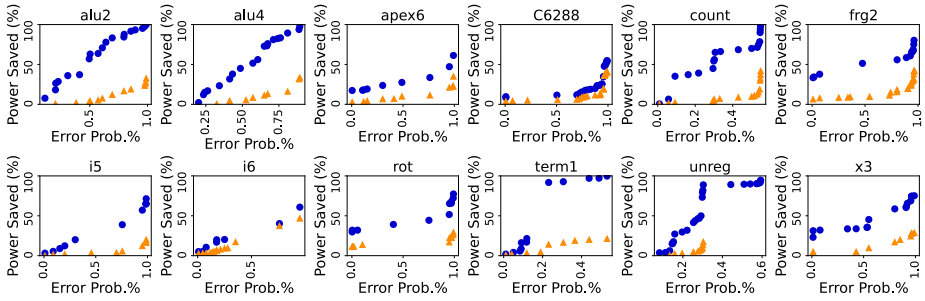
Despite expectations arising from experimental result, no direct proportionality relationship between the number of nodes and SwA is evident from the model discussed before. Furthermore, counterintuitive examples can be found.

Consider, for instance, the LUT configuration

$$X = \{0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1\}$$

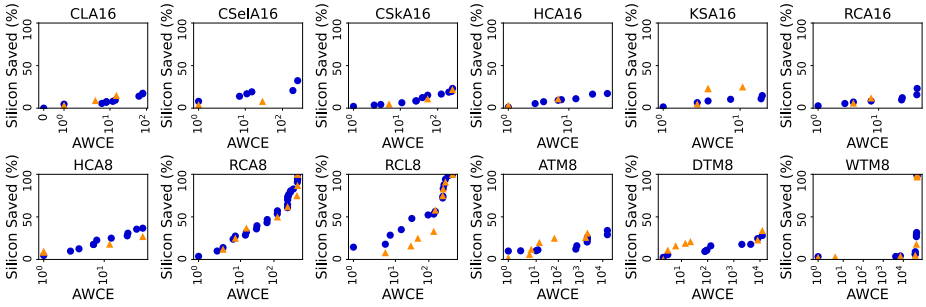


(a) Amount of saved silicon

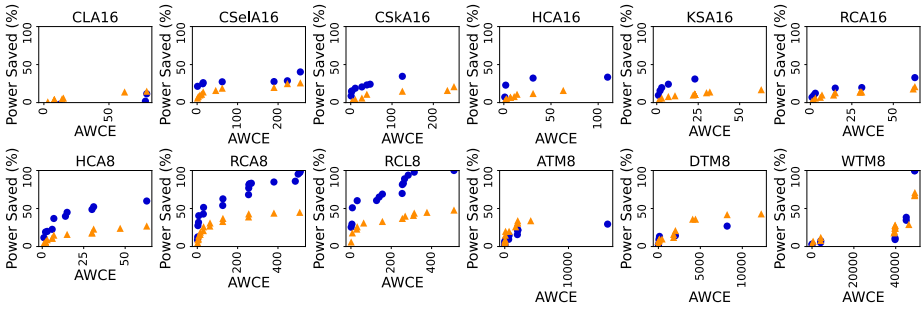


(b) Amount of saved power

Figure 5.5. Comparison of savings provided with generic circuits while targeting either the ASIC or the FPGA technology. The blue bullets ● denotes savings induced in ASIC implementations, while the orange triangles ▲ denote savings for FPGA technology.



(a) Amount of saved silicon



(b) Amount of saved power

Figure 5.6. Comparison of savings provided with arithmetic circuits while targeting either the ASIC or the FPGA technology. The blue bullets ● denotes savings induced in ASIC implementations, while the orange triangles ▲ denote savings for FPGA technology.

that requires 4 AIG nodes and, according to equations (5.2 - 5.6), has 1.625 SwA: its approximation at Hamming distance 1 resulting from ES, i.e.

$$X = \{1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1\}$$

exhibits 1.98 SwA despite requiring 3 AIG nodes. Besides, to the best of our knowledge, no correlation between the AIG node-count and SwA for LUTs has ever been identified in the scientific literature either.

Thus, to shed light on the source of the observed phenomenon, we exploited equations (5.4-5.6) to characterize every catalog entry – i.e., every LUT specification – we collected during our experimental campaign both in terms of AIG node-count and SwA. To this end, we assume inputs are equally probable – i.e, $f_i = \frac{1}{2^K},\ i = 1 \cdots 2^K - 1$ in (5.3) – and we take the signal reordering of inputs into account, by considering, for each of the catalog entries, the actual reordering that minimizes the SwA. The box-and-whisker plot in Figure 5.7 reports this preliminary characterization. At first glance, a general decreasing trend in terms of SwA, as far as the AIG node-count decreases, can be observed. Consequently, during DSE, while the AMOSA is attempting to minimize the AIG node-count, it is very likely that it (accidentally) pursues also SwA minimization; hence, resulting approximate circuits exhibit a lower power dissipation.

Anyway, the SwA variation ranges overlap for adjacent AIG node values, and, furthermore, we can also observe that, although values of the SwA tend to be gathered around the median value, there are several outliers, even for those LUTs requiring a moderate number of AIG nodes, e.g., three or four nodes. Therefore, even though the SwA of a LUT is close to the median value, during the catalog-generation procedure it may produce configurations exhibiting higher SwA, despite requiring less nodes. This gives reason to the existence of counterintuitive examples, although they do not have significant impact on the overall approximate circuit.

Indeed, still concerning counterintuitive examples, the LUTs we collected during the experimental campaign underwent a further characterization, to highlight their common features, if any. Such a characterization drew attention to the AIG topology and truth-density – i.e., is the number of truth assignments over the input set. In particular, approximate LUTs that constitute counterintuitive examples exhibit the following common features: (a) they exhibit complete functional dependency from all

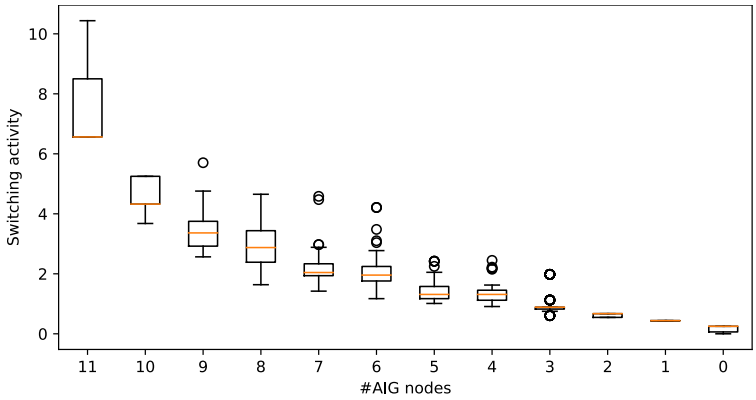


Figure 5.7. SwA of catalog entries, varying the AIG node-count.

inputs; (b) they do not implement elementary Boolean functions, such as logic-AND or logic OR; (c) the exact specification from which they originate has truth-density is almost 50%; (d) their truth-density is almost 50%; (e) the number of nodes of the corresponding AIG is very close to the theoretical minimum required by (a).

Additionally, we observed that in case (c) is true, to reach the termination criterion, the catalog-generation procedure is very likely to reorganize the truth table until resulting in propagating an input signal (meant that the approximate specification has 50% of truth density). Hence, during intermediate stages of the mentioned procedure, the truth-density fluctuates around to 50%, without ever being equal to it. Under these circumstances, the signal reordering of inputs is ineffective in reducing the SwA, since either the truth assignment is insusceptible to reordering or any reordering equals the original specification.

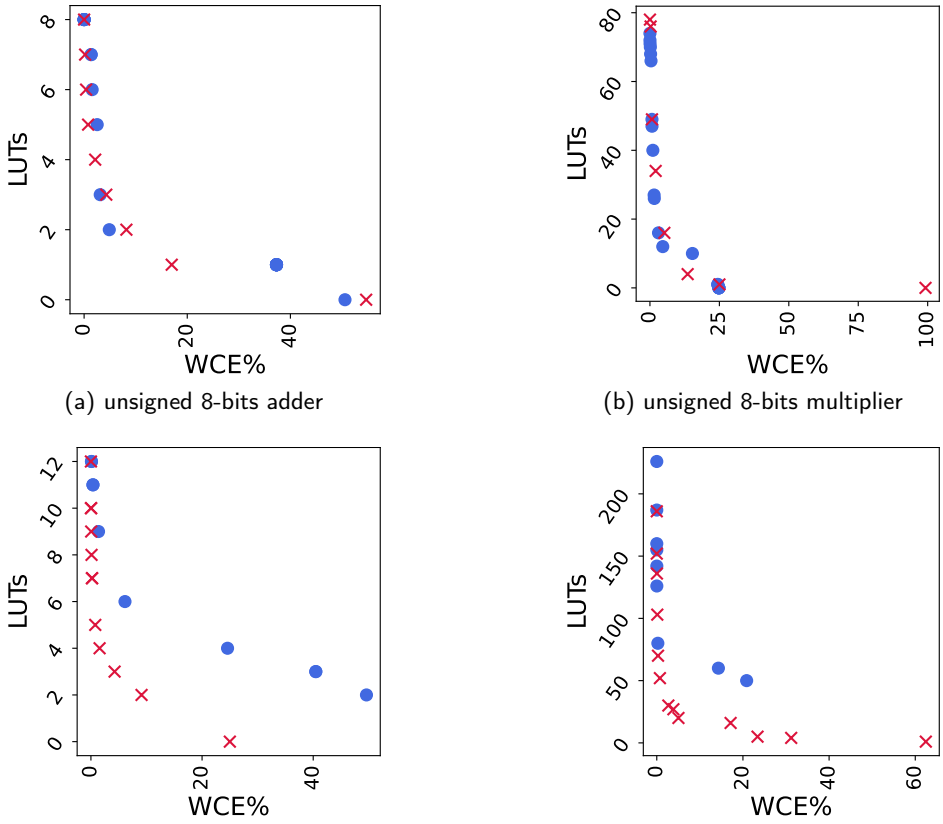
5.2.3 Comparison with previous works

In Section 2 we discussed several contributions from the scientific literature pertaining to the design of FPGA-based approximate computing systems, emphasizing the most recent contributions. In the following, we compare our approach with the state of the art, both qualitatively and quantitatively.

One of the most interesting approach is that from [Pra+20], that,

roughly, sifts through a library of approximate components (specifically, the EvoapproxLib [Mra+18], that has been designed while targeting ASIC) looking for those circuits providing optimal error/resources trade-offs for FPGA. Machine-learning models are exploited to predict FPGA requirements, avoiding FPGA synthesis. From a methodological perspective, when compared to the mentioned approach, our methodology allows designing FPGA-based approximate components from scratch. This means it does not require a library of ASIC-based approximate components, from which those providing optimal trade-offs between error and FPGA resource requirements have to be sifted. Furthermore, our method does not require high-fidelity predictors, since it exploits the node-count and depth of AIGs to estimate resource requirements during the DSE. In order to get a fair comparison of circuits resulting from our method against those from [Pra+20], we considered the exact (non-approximate) multiplier and adder from the ApproxFPGAs library, which are freely available at `https://github.com/ehw-fit/approx-fpgas` as starting point for our approach. In particular, we considered 8-bits and 12-bits operators, and once our workflow was completed, we synthesized resulting circuits while targeting a Xilinx xc7a35ticsg324-1L Artix-7 FPGA. For an unbiased comparison, we also resynthesized the circuits from the ApproxFPGAs library on the same device. Figure 5.8 and Figure 5.9 report results both in terms of FPGA LUT and power consumption, respectively. Red crosses × denote results from [Pra+20], while blue dots • denote results from our method. As the reader can observe, results from our method are competitive with the state-of-the-art, since the Pareto-fronts in Figure 5.8 and Figure 5.9 are barely distinguishable.

It is interesting to note, as far as power is concerned, that the approximate circuits resulting from our approach are equally good compared with those obtained from other state-of-the-art methods even though they may be more onerous in terms of LUTs, as is evident in Figure 5.8c and Figure 5.8d, which report LUTs for 8-bits adder and multipliers, respectively, and Figure 5.9c and Figure 5.9d, which, instead, report power consumption. While power-driven optimization will be the focus of Chapter 6, this can be explained intuitively by noticing in Figure 5.7 that, when optimizing for the number of AIG nodes and then mapping to LUTs with a reduced number of nodes, switching activity also follows a descending trend, with



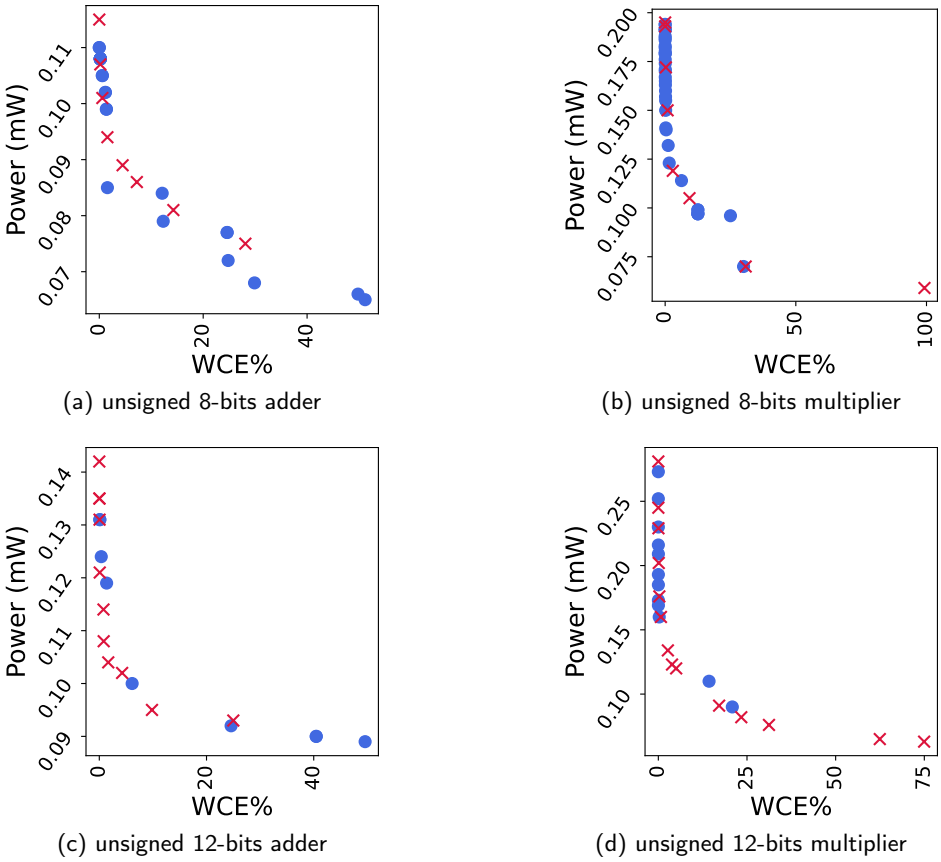
(a) unsigned 8-bits adder

(b) unsigned 8-bits multiplier

(c) unsigned 12-bits adder

(d) unsigned 12-bits multiplier

Figure 5.8. Comparison with [Pra+20] in terms of FPGA LUT. Red crosses × denote results from [Pra+20], while blue dots • denote results from our method.



(a) unsigned 8-bits adder

(b) unsigned 8-bits multiplier

(c) unsigned 12-bits adder

(d) unsigned 12-bits multiplier

Figure 5.9. Comparison with [Pra+20] in terms of power consumption. Red crosses × denote results from [Pra+20], while blue dots • denote results from our method.

Circuit	mul8u	add8u	mul12u	add12u
Our method	≈67h	≈13h	≈72h	≈15h
ApproxFPGAs [Pra+20]	≈192h	≈24h	≈34h	≈22h

Table 5.3. Computational-time required to design 8-bits unsigned-integer arithmetic circuits.

the exception of few outliers.

Pertaining to computational time, Table 5.3 compares our methodologies against [Pra+20]. The first row of that Table reports the computational time needed to perform our workflow as a whole – including the ES, the DSE and the final rewriting and hardware synthesis – while resorting to the algorithm configuration already discussed, performed on a 16-cores/32-threads AMD Ryzen 9550. The second row of the table reports the computational time required by the ApproxFPGAs as reported in [Pra+20], while running on a 16-cores/32-threads Intel Xeon E5 CPU. These times include the time required for synthesizing the data-set, training and evaluating the models, and re-synthesizing the Pareto optimal circuits. For the method in [Pra+20], the larger the circuit, the smaller the search space, due to the reduced number of candidate circuits in the library; hence, circuit size and computational time do not correlate. Since we do not need any model to be trained to estimate hardware requirements of candidate approximate circuits, and since we can perform faithful estimations from AIG nodes, when compared with [Pra+20], our method requires significantly less computational time to provide approximate circuits.

The approach from [Ull+22a] is one of the latest contributions to the scientific literature. Basically, it encodes circuits as binary strings. Given a circuit requiring N LUTs for partial product generation, the method from [Ull+22a] encodes each of the variant through a binary string of length N, and allows generating 2^N different approximate variants. A "0" at any location in the N-bits string disables corresponding LUT, which means that LUT will not contribute to producing the intermediate results. Concerning fitness functions driving the DSE, the authors of [Ull+22a] resort to the product between the power-delay-product and the number of LUTs as a measure for hardware resource requirements, while the AWCE (4.4) or the

Mean Absolute Error (MAE) (5.7) have been adopted to measure the error.

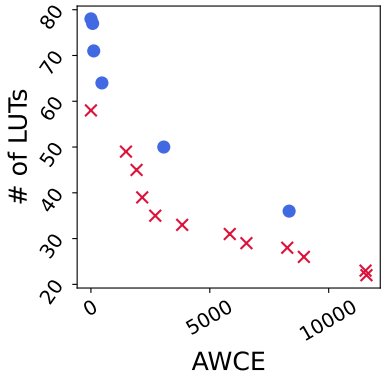
$$e_{\text{mae}}(f, \hat{f}) = \frac{1}{2^n} \sum_{x \in \mathbb{B}^n} |f(x) - \hat{f}(x)| \tag{5.7}$$

From the qualitative perspective, the grain with which approximation can be introduced in circuits while exploiting our approach is finer when compared to that in [Ull+22a]. In fact, as we observed in the previous section, results indicate that some replacements produce a reduction in power consumption, even though they did not result in any decrease in the number of LUTs. As a result, a more gradual and controlled degradation of the quality of results can be produced while lowering the hardware overhead. Indeed, the only case our approach leads to LUT suppression happens when the replacement being selected from the catalog either propagates one of the input signals, or even a constant signal. The only downside of having such a fine degree with which it is possible to act on the circuits is, as is easy to imagine, a generalized increase in the size of the search space.

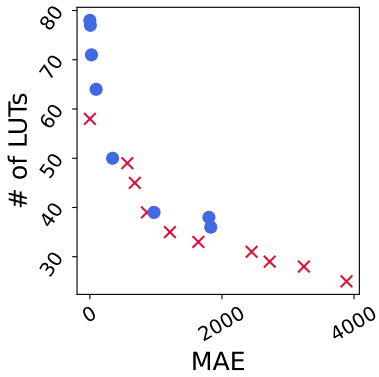
In order to quantitatively compare the methods, we performed our method while performing the DSE to minimize the AIG node count as well as the error entailed by approximation. The latter is measured either as AWCE or MAE. For a fair comparison, we measured the actual FPGA requirements by synthesizing resulting circuits while targeting the xc7vx330t device of the Virtex-7 family, as done in [Ull+22a].

In Figure 5.10 and Figure 5.11 we report a comparison of results from the mentioned methods in terms of FPGA resources (measured either as number of LUT or power consumption) and error (which is measured in terms of AWCE or MAE). In the mentioned figures, red crosses × denote results from [Ull+22a], while blue dots • denote results from our method. As the reader can observe, such results confirm our previous statement: our method allows more gradual and controlled degradation of the quality of results while lowering the hardware overhead. Indeed, results from our method exhibit lower power consumption w.r.t. those from [Ull+22a], albeit the latter produces circuits requiring less LUTs.

As for the computational time, once again we can safely claim our approach allows faster design, as it does not involve any model to be trained to estimate hardware requirements of candidate approximate circuits. Indeed, since we can perform faithful estimations from AIG nodes, when

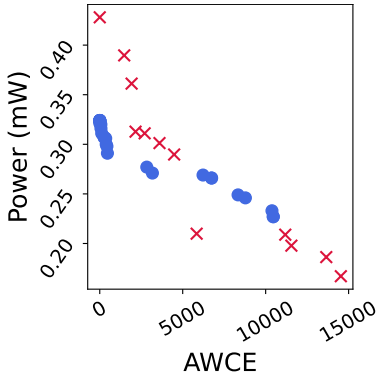


(a) AWCE (4.4) vs. number of LUTs

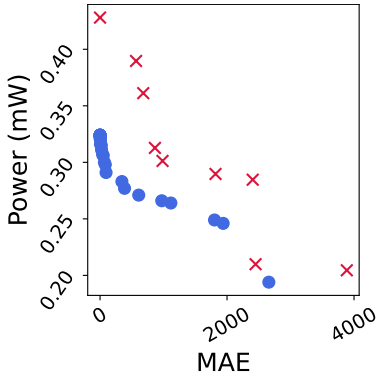


(b) MAE (5.7) vs. number of LUTs

Figure 5.10. Comparison with [Ull+22a] in terms of LUTs. Red crosses × denote results from [Ull+22a], while blue dots • denote results from our method.



(a) AWCE (4.4) vs. power consumption



(b) MAE (5.7) vs. power consumption

Figure 5.11. Comparison with [Ull+22a] in terms of power consumption. Red crosses × denote results from [Ull+22a], while blue dots • denote results from our method.

compared with [Ull+22a], our method requires significantly less computational time to provide approximate circuits: while the accuracy and PPA estimation for a single approximate 8-bits multiplier consumes nearly 3.55 minutes of processing time with the method in [Ull+22a], our method allows evaluating hundreds of variants in the same amount of time, since assessing the error on the whole 2^{16} possible inputs for an 8-bit multiplier only a few seconds, and the time required to count the number of AIG nodes is paltry.

5.3 Case studies

In this Section, we discuss several case-studies in which approximate circuits resulting from our method are exploited to reduce hardware requirements of larger and complex applications. In particular, we discuss the implementation of a Sobel edge detector in Section 5.3.1, a FIR filter in Section 5.3.2, and, finally, we discuss convolutional neural networks in Section 5.3.3.

5.3.1 The Sobel edge-detector

One of the major fields of application for AxC is image-processing, since, due to perceptual limitations of human eyes, imperceptible reduction of image quality can lead to important savings. One quite common image-processing application is the Sobel edge-detector; therefore, we provide a case-study concerning the design of a hardware accelerator for the aforementioned application, exploiting approximate components resulting from our approach. We only approximate adders, since, at the hardware level, multiplications by two can be implemented as a left shift, which is just wiring requiring no resources. Specifically, we adopted an approximate implementation of a 16-bits Kogge-Stone adder that exhibits AWCE of 32, costs 46 LUTs when synthesized on a Xilinx xc7a35ticsg324-1L Artix-7 FPGA (6 LUTs, or 11% less than its exact counterpart), and consumes 155 mW (26 mW, or 16% less than its exact counterpart) on the same device. We target the above-mentioned FPGA while performing synthesis of both the exact and the approximate implementations of the Sobel edge-detector, to measure their actual hardware requirements. The exact

(a) Visual test with Pepper

(b) Visual test with House

Figure 5.12. Visual test. Original images are on the left; while the output of the exact implementation of the edge-detector is reported in the center. Images resulting from the approximate edge-detector are on the right. From top to bottom, the right-most images exhibit PSNR of 28.9 and 28.5.

implementation of the mentioned edge-detector requires 435 LUTs and consumes 743 mW, while its approximate implementation requires 349 LUTs and 620 mW, providing savings up to 19% and 16% for LUTs and power consumption, respectively.

Furthermore, we also report a comparison between images computed through the use of the exact and the approximate implementations in Figure 5.12: images computed using the approximate implementation are reported on the right of Figure 5.12, and exhibit a PSNR in the $[28.5, 28.9]$ range w.r.t. reference images, i.e., those reported at the center of Figure 5.12. The higher the PSNR the higher the quality of the images.

5.3.2 The FIR filter

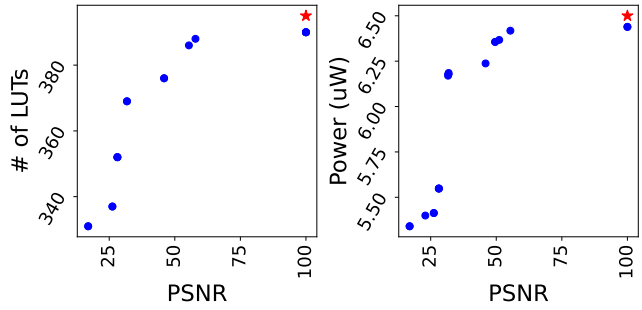
In this case study, we target one of the most common signal processing applications, which is the FIR filter, whose general definition is reported in (5.8). There, x_j is the j^{th} sample of the input signal, b_i is the i^{th} coefficient of the signal, and y_n is the n^{th} sample of the output signal. Here we will replace the multiplication in (5.8) using an approximate multiplier to save FPGA LUTs and power.

$$y_n = \sum_{i=0}^{N} b_i \cdot x_{n-i} \tag{5.8}$$

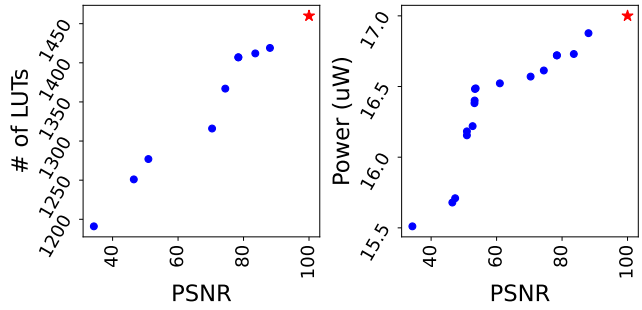
We consider two different FIR filters: they both have a 500 Hz cut-off frequency, 60 dB attenuation in the stop band, and use fixed-point arithmetic. The first is a low-pass filter that adopts the Q1.7 fixed-point arithmetic, while the second is a high-pass filter using the Q1.15 representation. In order to evaluate our approach, we assess the impact of approximation by measuring the PSNR between the output signals produced by the non-approximate and approximate FIRs.

Specifically, we adopted the 8-bits and 16-bits Wallace tree multiplier as starting point, and, after the DSE, we synthesized the design tartegint a Xilinx xc7a35ticsg324-1L Artix-7 FPGA, collecting hardware requirements. We, then, performed simulation in order to compute the PSNR while processing a saw-tooth signal. Figure 5.13a and Figure 5.13b report results for the 8-bits and 16-bits FIRs, respectively. The red star ⋆ denotes the reference (non-approximate) implementation, while the blue bullets • denotes filters being implemented using approximate multipliers. As the reader can observe, approximate multipliers resulting from our method allow achieving significant savings while introducing only a restrained amount of error in the final application.

It's interesting to note, however, that the quality of the output degrades rapidly with this kind of circuits. While this phenomenon has not been analyzided in depth, it is probably due to the interdependencies between intermediate results. The results suggest that this kind of application could benefit from a more custom-tailored approach, such as specifying different approximation constraints for different parts of the circuit, or from a faster metrics evaluation, that could allow to explore larger parts of the design



(a) 8-bits FIR



(b) 16-bits FIR

Figure 5.13. PSNR and hardware resources for 8-bits and 16-bits FIRs while using approximate multipliers. The red star ⋆ denotes the reference (non-approximate) implementation, while the blue bullets • denotes filters being implemented using approximate multipliers.

space more efficiently.

5.3.3 Convolutional neural networks

In this case study, we target CNNs, a class of ANNs most commonly applied to analyzing visual imagery [Ben09; Sch15]. The computational model of artificial neurons is inspired by its biological counterpart, but ANNs are organized in distinct *layers*, which define the network's depth, rather than being modeled as an amorphous blob of connected neurons. Neurons belonging to adjacent layers can be either fully or partially connected, while there is no connection between neurons within the same layer. Various types of layers with various topologies have been defined over the years. For instance, in Fully-Connected Layers (FCLs), neurons of the layer are connected to all the neurons of the preceding layer, while in Convolutional Layers (CLs), neurons in a layer are connected only to a small region of the previous layer, and they perform computations that are not just a function of the inputs. Indeed, as reported in Equation (5.9), the output of each of the neurons is a non-linear function f of the weighted-sum involving learned weights w_i and inputs x_i, plus a bias b.

$$y = f\left(\sum_{i}^{n} w_i \cdot x_i + b\right) \tag{5.9}$$

Concerning approximation, it is typically introduced by replacing exact multiplications within neurons with approximate ones [Ans+20]. Hence, we aim at designing a multiplier to perform the $w_i \cdot x_i$ part of Equation (5.9), targeting convolutional layers of the CNN with the goal of simultaneously reducing the hardware requirements and the impact of approximation on the classification accuracy loss during the inference phase. We resort to the approach from [Mra+19], which replaces accurate convolution layers within ANNs using approximate ones. Specifically, for each convolution layer, a suitable approximate multiplier is selected, while the overall classification error – i.e., accuracy loss – and energy consumption are simultaneously minimized through a MOP. During the DSE, we estimate the power-consumed by the approximate CNN as the weighted sum of power-consumption of each multiplier of the net, respectively, as done in [Mra+19].

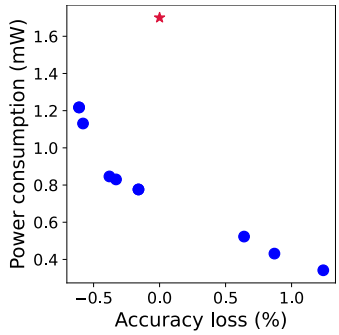


Figure 5.14. Accuracy loss and hardware resources for LeNet-5 while using approximate multipliers. The red star ⋆ denotes the reference (non-approximate) implementation, while the blue bullets ● denotes CNN being implemented using approximate multipliers.

The network architecture we consider in our case study is LeNet5 [LeC+89], trained to classify images from the Modified National Institute of Standards and Technology (MNIST) benchmark [LCB98], which consists of 70000 28×28 grayscale images of handwritten digits, of which 60000 are for training purpose and 10000 testing. The network has been quantized using 8-bit signed integer arithmetic, and it exhibits 99.07% accuracy.

Results are reported in Figure 5.14. Once again, we denote the reference (non-approximate) implementation of the CNN using the red star ⋆, while the blue bullets ● denotes CNN being implemented using approximate multipliers. While area requirements strictly depends on the architecture of the tensor processing unit, power largely depends on the number of multiplications being performed. Since the negligible accuracy loss and the savings we can observe in Figure 5.14, we can reiterate that our method is able to provide efficient implementations for approximate multipliers suitable for FPGA synthesis.

An interesting phenomenon in this case study is that there are negative accuracy loss points. While this is currently under study, it may suggest that approximation could be used to mitigate overfitting. However, more thorough study with a larger benchmark should be conducted in order to ascertain this impact.

Chapter 6

Driving optimization with the LUT power model

In Chapter 5 we showed that the methodology introduced in Chapter 3 can be applied as-is to circuits targeting the FPGA technology, obtaining significant savings both in area and power.

The obtained power savings however can be considered a *"byproduct"* of an optimization process that, as specified by the methodology and as implemented by the tools used, aims to reduce the number of AIG nodes. Moreover, selecting the best candidates in terms of power requires a cumbersome synthesis and power analysis performed by using external flows, such as the ones provided by the FPGA vendor.

It is natural, therefore, to consider the integration of the power model proposed in 5.1.1 directly in the optimization step of the workflow. This allows to directly drive the optimization algorithm using the effective target metric, avoiding unneeded synthesis steps. This modification makes the methodology more sound for application to FPGA devices, and saves the computation time associated with the synthesis and power analysis.

6.1 Experimental evaluation

In this Section, we provide the reader with a brief discussion concerning the experimental setup for evaluating the use of the power model described in Section 5.1 in designing approximate circuits.

We first discuss a preliminary analysis we conducted to validate the model, to verify that it provides a faithful estimate of the SwA, hence that it is suitable to lead the DSE. Then, we fully evaluate our method through an extensive experimental campaign, involving several benchmark circuits, and two different FPGA architectures. We provide comparison with state-of-the-art in Section 6.1.2 and, finally, we provide a case-study concerning designing approximate neural networks in Section 6.2.

6.1.1 Generic logic and arithmetic benchmarks

This preliminary analysis involved more than 2500 different approximate configurations, which have been generated through the technique discussed in this chapter starting from 25 different circuits. Specifically, we considered arithmetic circuits from the ArithsGen [KM22] and from the Arithmetic Module Generator [UNT17] frameworks, as well as generic logic circuits from the LGSynth91 logic synthesis benchmark suite. For each of the starting points, 100 approximate variants have been designed while simultaneously minimizing error and resource requirements in terms of AIG-nodes. The error metric chosen for circuits drawn from the LGSynt91 benchmark is the EP (4.1), while the AWCE (4.4) has been adopted as the error metric for arithmetic circuits.

At the end of the DSE, we estimated and measured the SwA while targeting two different FPGA fabrics, namely the AMD/Xilinx Spartan3E and the Artix-7. Measured SwA is provided by the AMD/Xilinx PlanAhead framework[1] while targeting the xc3s500efg320-5 Spartan3E FPGA, and by the AMD/Xilinx Vivado framework[2] while targeting the xc7a35ticsg324-1L Artix-7 FPGAs. This is to empirically prove that the power model in Section 5.1 provides plausible estimation for the SwA regardless the FPGA fabric. In addition, data concerning the SwA has been collected through post-implementation timing simulation involving random input assignments.

Then, we computed the correlation index and the *fidelity index* of the model w.r.t. the non-dominance relationship. The latter index, given a pair of Pareto-optimal solutions – i.e., a pair of solutions in non-dominance

[1] https://www.xilinx.com/products/design-tools/planahead.html
[2] https://www.xilinx.com/products/design-tools/vivado.html

Table 6.1. Correlation and fidelity between the estimation from the LUT power dissipation model and the one from the FPGA synthesis tool.

Circuit	Artix-7		Spartan 3E	
	$\mathbf{corr}_{XY}$	**Fidelity (%)**	$\mathbf{corr}_{XY}$	**Fidelity (%)**
alu2	0.848	81.02	0.829	85.412
alu4	0.846	90.039	0.88	88.078
count	0.912	86.437	0.82	87.701
i6	0.994	95.48	0.96	92.79
term1	0.95	81.311	0.891	83.513
unreg	0.968	85.118	0.903	83.122
x3	0.912	93.188	0.929	89.647
z4ml	0.959	90.353	0.963	95.484
8-bits Array Mul.	0.984	92.863	0.929	83.373
8-bits Dadda Mul.	0.97	91.156	0.98	84.966
8-bits Wallace Mul.	0.953	91.892	0.841	80.882
16-bits Carry-lookahead Add.	0.995	95.922	0.919	89.647
16-bits Carry-select Add.	0.96	92.179	0.85	81.795
8-bits Carry-lookahead Add.	0.949	86.295	0.926	80.256
16-bits Han-Carlson Add.	0.947	89.25	0.95	88.088
8-bits Han-Carlson Add.	0.963	81.075	0.871	95.484
16-bits Ripple-Carry Add.	0.987	95.47	0.946	93.403
8-bits Ripple-Carry Add.	0.989	91.859	0.858	81.961
16-bits Carry-skip Add.	0.942	87.293	0.84	84.747
16-bits Kogge-Stone Add.	0.965	98.901	0.78	89.341
8-bits Kogge-Stone Add.	0.961	95.732	0.953	87.683

relationship among them w.r.t. the error and the estimate given by the model –, corresponds to the number of times that the non-dominance relationship still holds true when the estimate from our model is replaced by the measure of power dissipation obtained from the FPGA synthesis tool.

For each of the circuit family being considered for validation, both the correlation and the fidelity are reported in Table 6.1. Since $\mathrm{corr}_{XY} \approx 1$ for almost all the circuit families, we can conclude a strong correlation exists between the proposed model and that provided by the synthesis tool. Furthermore, since the fidelity of the model is pretty high too, we can conclude that it allows for a faithful estimation of the SwA of circuits without the need for an actual FPGA synthesis.

As the high correlation and fidelity indexes prove, the described LUT power dissipation model is well suited to drive the DSE while pursuing error and power consumption minimization. Thus, we took the same benchmark circuits mentioned before, and besides minimizing the SwA according to

the model, we minimized the EP (4.1) and the AWCE (4.4) for generic logic and arithmetic circuits, respectively. After the DSE, we synthesized the resulting Pareto-optimal configuration leveraging the AMD/Xilinx PlanAhead/Vivado framework while targeting the Spartan-3E and the Artix-7 FPGAs. We report both the error and the actual power consumption – as provided by the mentioned frameworks – in Figure 6.2 and Figure 6.1, respectively. Power savings provided by optimizing approximate variants while considering the SwA estimation provided by the model are undeniable, since decreasing trends in the power consumption as the error increases are clear. Furthermore, we can safely claim the model is well suited to drive the DSE while pursuing error and power consumption minimization, regardless of the underlying FPGA fabric.

6.1.2 Comparison with previous works

In this Section, we compare results from our methodology against the state-of-the-art. Specifically, we report a comparison with [Pra+20], [Ull+22a], and [Bar+24].

As discussed in Section 2, the approach from [Pra+20] is independent of the specific FPGA. In fact, authors do not assume knowledge of the particular target FPGA, nor exploit any peculiarities of a specific fabric to introduce approximation, since the proposed approach sifts candidate circuits from already available libraries of approximate components. Nevertheless, it needs high-fidelity predictors to estimate resource requirements. Conversely, our methodology allows designing FPGA-based approximate components from scratch.

Starting from the non-approximate circuits from the ApproxFPGAs library [Pra+20], we designed an approximate multiplier and an approximate adder for comparison purposes. We address the design while minimizing both the error and SwA. Once our workflow was completed, in order to measure their actual hardware requirements, we synthesized resulting circuits targeting the same FPGA as in [Pra+20], that is, a Xilinx xc7a35ticsg324-1L Artix-7 FPGA. Results are reported in Figure 6.3 and Figure 6.4. The ♦ denote circuits resulting from [Pra+20], while • denotes those resulting from our method. As the reader can observe, results from our method are competitive with the state-of-the-art, since the Pareto-fronts are barely distinguishable. Despite this, our method requires

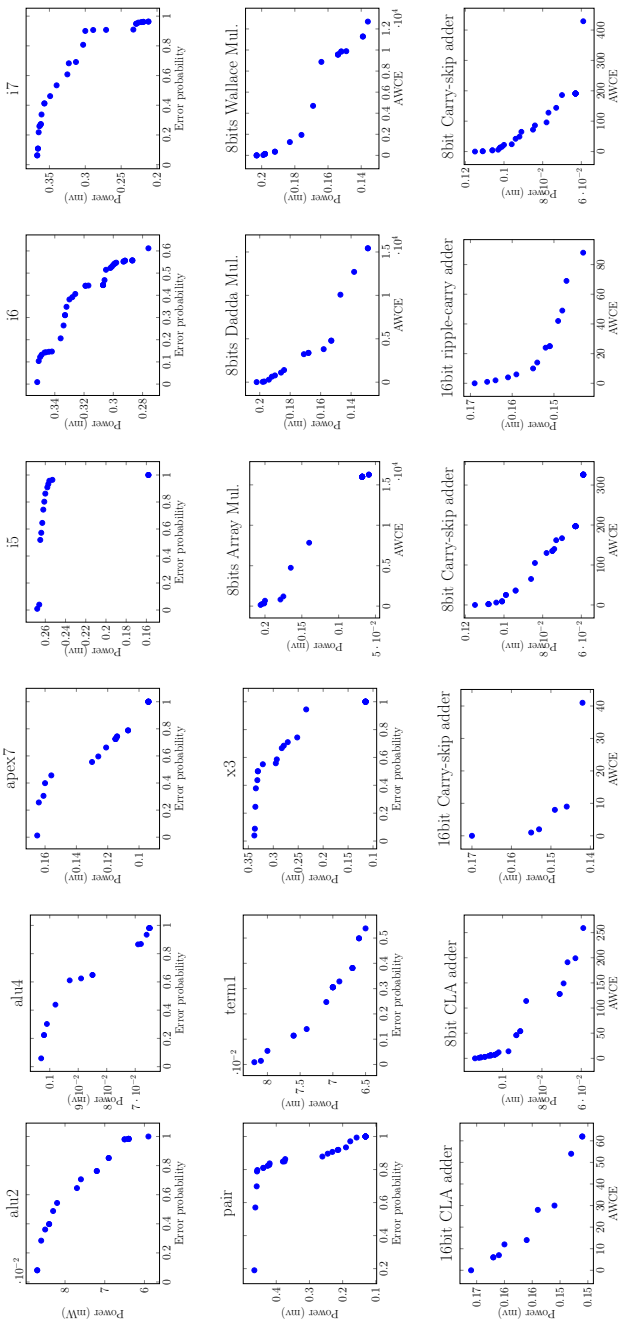


Figure 6.1. Error and power dissipation of approximate circuits while targeting the Artix 7 FPGA.

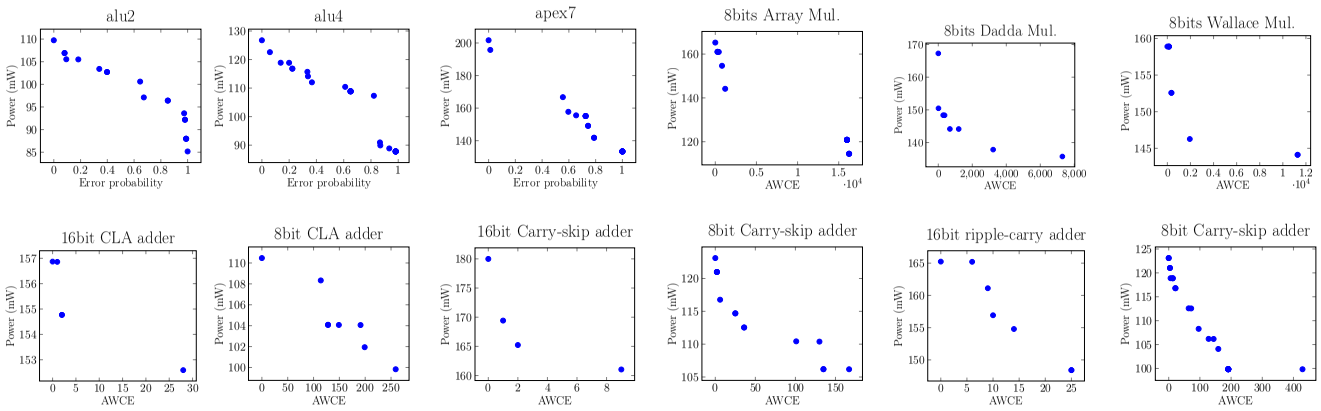


Figure 6.2. Error and power dissipation of approximate circuits while targeting the Spartan-3E FPGA.

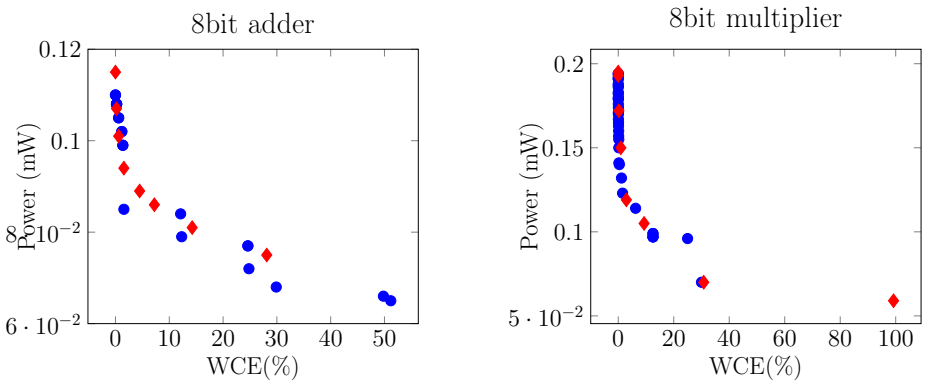


Figure 6.3. Comparison with [Pra+20] in terms of power consumption. ♦ denotes circuits from the ApproxFPGAs library [Pra+20] while • denotes those resulting from our method.

Table 6.2. Computational-time required to design 8-bits unsigned-integer arithmetic circuits.

Circuit	Multiplier	Adder
Our method	≈58h	≈9h
ApproxFPGAs [Pra+20]	≈129h	≈23h

significantly less computational time to provide approximate circuits, as reported in Table 6.2. This is because no machine-learning model is needed to estimate hardware requirements.

As for the method in [Ull+22a], for a fair comparison, we measured the actual FPGA requirements by synthesizing the resulting circuits while targeting the same FPGA as done by the authors of [Ull+22a], viz. a xc7vx330t Virtex-7. Results are reported in Figure 6.5 and in Figure 6.6, in terms of error (which is measured in terms of AWCE or MAE) power consumption and LUTs. The ♦ denotes circuits from the AppAxO framework [Ull+22a] while the • denotes those resulting from our method, that undeniably exhibit lower power consumption w.r.t. those from [Ull+22a]. Despite optimizing for SwA, our approach allows for an optimization which is comparable to the one proposed in [Ull+22a], which, instead, performs an optimization in terms of power-delay product. Indeed, appreciable differences onto the parento-front is about few spare units.

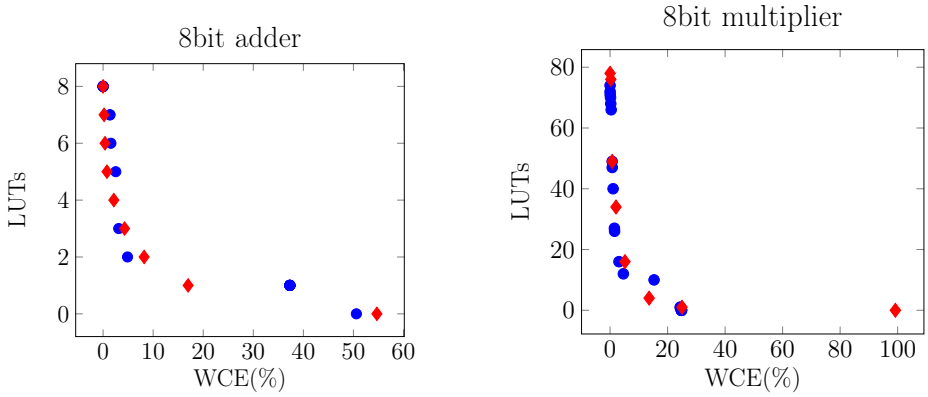


Figure 6.4. Comparison with [Pra+20] in terms of number of LUTs. ♦ denotes circuits from the ApproxFPGAs library [Pra+20] while ● denotes those resulting from our method.

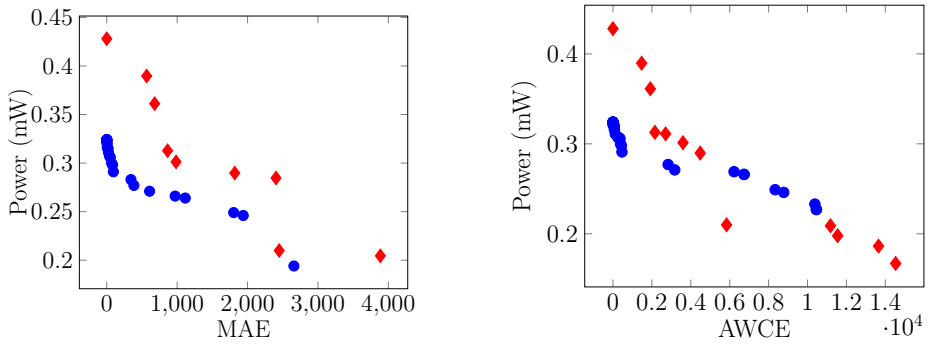


Figure 6.5. Comparison with [Ull+22a] in terms of power consumption. ♦ denotes circuits from the AppAxO framework [Ull+22a] while ● denotes those resulting from our method.

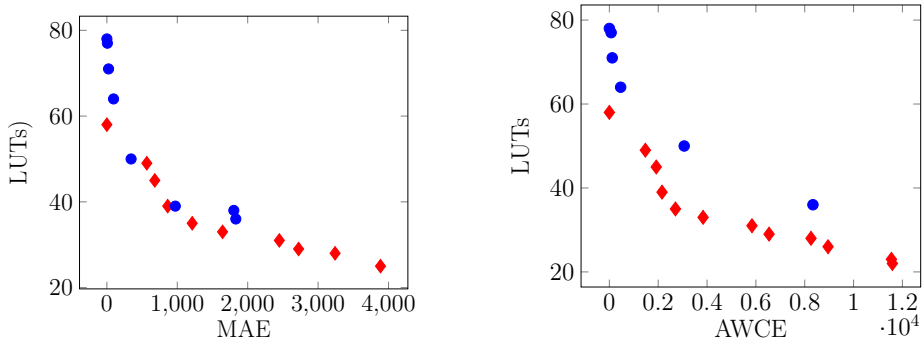


Figure 6.6. Comparison with [Ull+22a] in terms of number of LUTs. ♦ denotes circuits from the AppAxO framework [Ull+22a] while ● denotes those resulting from our method.

As for the approach published in [Bar+24], we substantially executed DSE on the same benchmark circuit set, while targeting different objective functions. Indeed, the approach from [Bar+24] leverages the AIG node-count to indirectly optimize power consumption; conversely, we directly optimize for power consumption by driving the DSE by means of SwA estimation. Intuitively, our approach does not require any other post-processing measurement to retrieve power savings w.r.t. exact syntheses. Hence, in order to provide a fair comparison, we plot in Figure 6.7 solutions taken from our approach and one proposed in [Bar+24] in terms of error and power consumption calculated from the synthesis tool. As one can see, from a quantitative point of view, results are comparable.

6.2 Case study: Convolutional neural networks

In this case study, we exploit approximate circuits resulting from our method to design approximate CNNs. These are the ideal field of application for approximate computing due to their inner error resiliency and to the tremendous amount of power consumed during computation, especially when considering hardware acceleration [Mit20].

Since multipliers are the most demanding component in terms of hardware resources, approximate CNNs are commonly implemented resorting

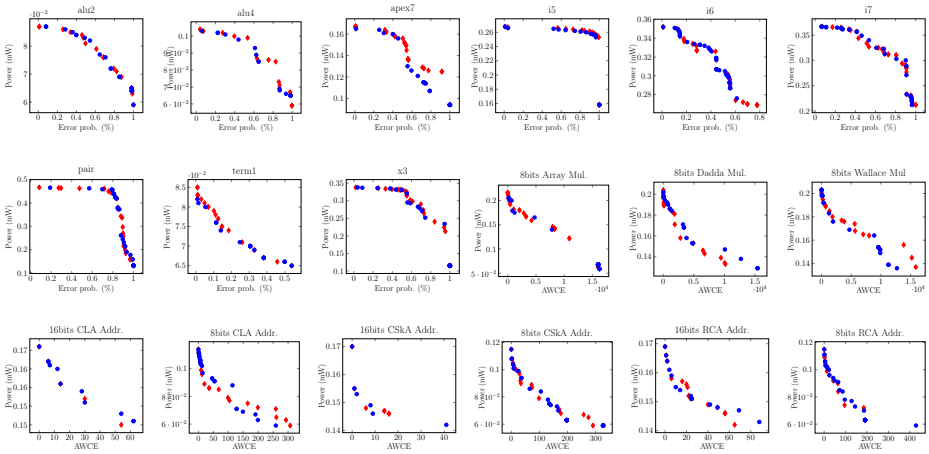


Figure 6.7. Comparison with [Bar+24] in terms of power consumption. The ♦ marker denotes circuits from [Bar+24] while the ● marker denotes those resulting from our method.

to approximate multipliers, with the ultimate goal of improving the energy efficiency of the inference phase [Arm+22]. We resort to the approach from [Mra+19], which replaces accurate convolution layers within CNNs with approximate ones. Specifically, for each convolution layer, a suitable approximate multiplier is selected, while the overall classification error – i.e., accuracy loss – and energy consumption – estimated as the weighted sum of power-consumption of each multiplier of the net – are simultaneously minimized, using NSGA-II [Deb+02b] to select the best trade-offs. The network architectures we consider in our case study are LeNet5 [LeC+89], trained to classify images from the MNIST benchmark [LCB98], four CNNs from the ResNet family [He+16] – namely, ResNet-8, ResNet-14, ResNet-24 and ResNet-50 – and AlexNet [KSH12]. The ResNet-8, ResNet-14 and ResNet-24 have been trained to cope with the CIFAR-10 dataset [KNH10], while the ResNet-50 and AlexNet have been trained to address the CIFAR-100 dataset [KNH10]. Results are reported in Figure 6.8, where ✳ symbol refers to the non-approximate CNN, ● markers denote approximate CNNs. As it is easy to foresee, using approximate multipliers leads to significant enhancements in efficiency,

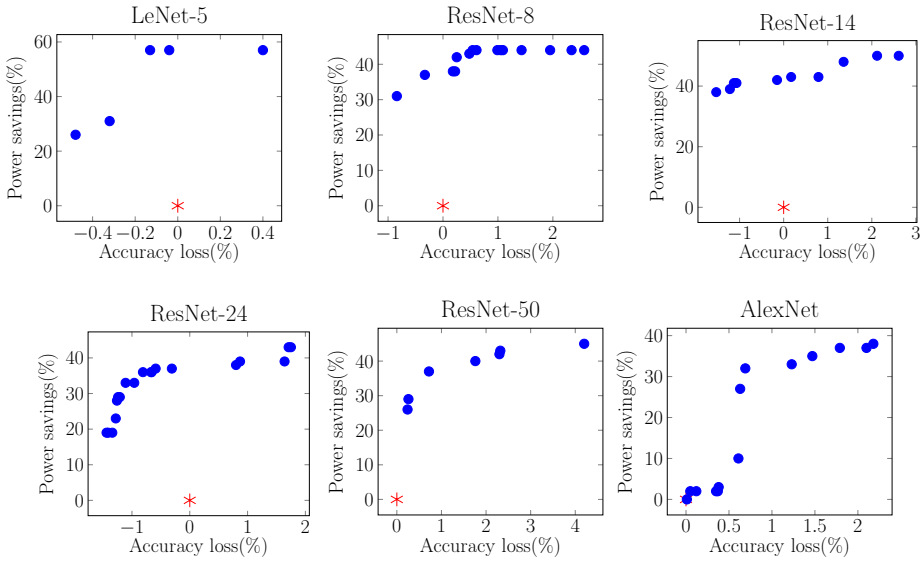


Figure 6.8. Accuracy loss (%) and power savings for CNNs while using approximate multipliers resulting from our method. The ✳ symbol refers to the non-approximate CNN, ● markers denote approximate CNNs.

allowing for up to 45% power savings, maintaining an acceptable level of accuracy loss. Moreover, for some configurations, the accuracy loss is negative, indicating an accuracy gain. As discussed in [Ans+20], approximate multipliers introduce small inaccuracies to synaptic weights, which can help mitigate overfitting issues that characterize trained CNNs.

Chapter 7

Conclusions

In this thesis, we proposed a generic and automatic methodology for the synthesis of approximate circuits based on the local rewriting of AIG. Given a HDL description of a combinational circuit, SMT based exact synthesis is used to build a catalogue of candidate substitutes for selected cuts of the circuit, that are formally proven to be size-optimal w.r.t. the number of AND nodes.

A heuristic based on a multi-objective simulated annealing algorithm has been used to guide the choice of the cuts to substitute, in order to obtain a set of dominant configurations w.r.t. the error rate and the number of nodes in the circuit.

We evaluated our approach using different benchmarks, and, in order to measure actual gains, we performed actual synthesis of Pareto-optimal approximate configurations, both targeting the ASIC and FPGA technologies. We also proposed an alternative approach that uses a simple LUT power model to drive the optimization when targeting the FPGA technology.

Experimental results showed that the proposed approach allows achieving significant savings, since resulting approximate circuits exhibited lower requirements and restrained error w.r.t. their exact couterparts.

Appendix A

Boolean circuits

In this appendix we provide precise definitions for the concepts of Boolean functions and circuits that we used intuitively in the manuscript. The definitions we use are taken from [Vol13].

Definition A.1. A *Boolean function* is a function $f\colon \{0,1\}^m \to \{0,1\}$ for some $m \in \mathbb{N}$.

Definition A.2. A *family of Boolean functions* is a sequence $f = (f^n)_{n\in\mathbb{N}}$ where f^n is an n-ary Boolean function.

Definition A.3. A *basis* is a finite set consisting of Boolean functions and families of Boolean functions.

Intuitively, a basis is the "logic family" we use to build our circuit.

Definition A.4. Let B be a basis. A *Boolean circuit* over B with n inputs and m outputs is a tuple

$$C = (V, E, \alpha, \beta, \omega)$$

where (V, E) is a finite directed acyclic graph, $\alpha\colon E \to \mathbb{N}$ is an injective function, $\beta\colon V \to B \bigcup \{x_1, \ldots, x_n\}$, and $\omega\colon V \to \{y_1, \ldots, y_m\} \bigcup \{*\}$, such that the following conditions hold:

1. If $v \in V$ has in-degree 0, then $\beta(v) \in \{x_1, \ldots, x_n\}$ or $\beta(v)$ is a 0-ary Boolean function (i.e. a Boolean constant) from B.

2. If $v \in V$ has in-degree $k > 0$, then $\beta(v)$ is a k-ary Boolean function from B or a family of Boolean functions from B.

3. For every i, $1 \leq i \leq n$, there is at most one node $v \in V$ such that $\beta(v) = x_i$.

4. For every i, $1 \leq i \leq m$, there is exactly one node $v \in V$ such that $\omega(v) = y_i$.

If $v \in V$ has in-degree k_0 and out-degree k_1, then we say: v is a *gate* in C with *fan-in* k_0 and *fan-out* k_1. If v is a gate in C then we also write $v \in C$ instead of $v \in V$. If $e = (u, v) \in E$ then we say: e is a *wire* in C; more specifically we say that e is an input wire of v and an output wire of u; and that u is a *predecessor gate* of v. If $\beta(v) = x_i$ for some i then v is an *input gate* or *input node*. If $\omega(v) \neq *$ then v is an *output gate* or *output node*.

In order to be able to compute a function f_C with a circuit C, we need another function that ties a Boolean value to the nodes of the circuit.

Definition A.5. Let $C = (V, E, \alpha, \beta, \omega)$ be a circuit over B with n inputs and m outputs; let $\mathbf{A} = \{a_1, \ldots, a_n\}$ be arbitrary binary values. We define for every $v \in V$ a function $val_v \colon \{0,1\}^n \to \{0,1\}$.

If v has fan-in 0 and $\beta(v) = x_i$ for some $i \in 1, \ldots, n$, then $val_v(\mathbf{A}) =_{def} a_i$. If v has fan-in 0 and $\beta(v) = b$ is a 0-ary function from B, then $val_v(\mathbf{A}) = b$.

If v has fan-in $k > 0$ and $v_1, \ldots, v_k$ are the gates that precede it ordered in such a way that $\alpha(v_1) < \cdots < \alpha(v_m)$, we consider $b(v) = f \in B$. If f is a $k - ary$ function, then

$$val_v(\mathbf{A}) =_{def} f(val_{v_1}(\mathbf{A}), \ldots, val_{v_k}(\mathbf{A})) \quad .$$

Otherwise, f is a family of Boolean functions, and we define

$$val_v(\mathbf{A}) =_{def} f^k(val_{v_1}(\mathbf{A}), \ldots, val_{v_k}(\mathbf{A})) \quad .$$

Appendix B

Optimization problems

Many problems in EDA, particularly logic design problems, are discrete in nature and can be stated as combinatorial optimization problems [Mic94]. For this reason, in this appendix we briefly introduce the concepts in optimization theory we need for our approach to ALS and a metaheuristic for multi-objective problems.

B.1 Combinatorial optimization

Informally, every time we have to choose a certain number of "parts", each from one set, in a way that maximizes a given criterion, we are solving a combinatorial optimization problem. The definition we use is from [BR03].

Definition B.1. A *combinatorial optimization problem* $P = (S, f)$ can be defined by:

- a set of *decision variables* $X = \{x_1, \ldots, x_n\}$;
- variable domains $D_1, \ldots, D_n$;
- constraints among variables;
- an objective function f to be maximized,

$$f : D_1 \times \cdots \times D_n \to \mathbb{R} \quad .$$

The set

$$S = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \mid v_i \in D_i \wedge s \text{ satisfies all constraints}\}$$

is the set of all feasible solutions, usually called *search space*.

From this definition, it's clear that our objective is to find the "best" element $s^* \in S$; such an element is called a *globally optimum solution*, and more than one may actually exist.

Definition B.2. A *globally optimal solution* to a combinatorial optimization problem $P = (S, f)$ is an element $s^* \in S$ such that

$$f(s^*) \leq f(s) \; \forall s \in S$$

The set of all globally optimal solutions is denoted with S^*.

The same definitions, with minor modifications, apply to minimization problems.

B.2 Multi-objective optimization

In many cases (such as the one of our interest) we're not interested in optimizing a single objective function, but two or more. For example, we may want to realize a circuit that is both cheap and energy efficient.

That is, the objective in this case is to maximize a function

$$\mathbf{f}\colon D_1 \times \cdots \times D_n \to \mathbb{R}^k \quad .$$

These kind of problems are called MOCO problems, and defining the concept of a solution for them isn't as straightforward because the different objectives, can be constrasting, and the choice of a solution usually happens to be a trade-off. Intuitively, we'd like to concentrate on the "best options" for solutions. A possible way to characterize the set of our best options is to introduce the concept of *Pareto front* [Luk13] of our search space.

Definition B.3. For a given Multi-Objective Combinatory Optimization (MOCO) problem $P = (S, \mathbf{f})$ and two feasible solutions $s_1, s_2 \in S$ we say

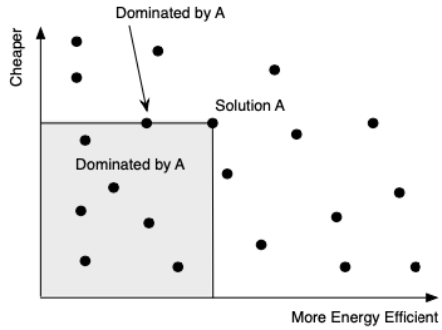


Figure B.1. Region of solutions Pareto dominated by solution A (from [Luk13])

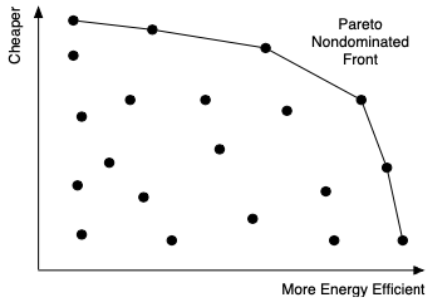


Figure B.2. The Pareto front (from [Luk13])

that s_1 *Pareto dominates* s_2, and we write $s_1 \prec s_2$, iff

$$(f_i(s_1) \geq f_i(s_i)\ \forall\ i \in \{0,\ldots,k{-}1\}) \wedge (\exists\ h \in \{0,\ldots,k{-}1\}\colon f_h(s_1) > f_h(s_2))$$

The definition means that s_1 is "at-least as good as" s_2 w.r.t. all considered criteria, and better w.r.t. at least one. If we can't find another feasible solution that dominates the one we're looking at, then it belongs to the Pareto front.

Definition B.4. The *Pareto non-dominated front* (or simply the *Pareto front*) for a given problem $P = (S, \mathbf{f})$ is the subset S^* of elements of S that

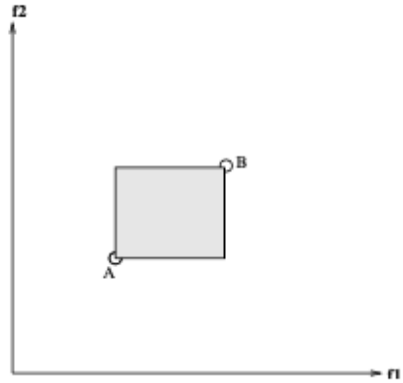


Figure B.3. Amount of domination (from [Ban+08c])

are non-dominated, that is

$$S^* = \{s \in S \mid \nexists\, \overline{s} \neq s \in S : \overline{s} \prec s\}$$

B.2.1 Multi-objective simulated annealing

The *simulated annealing* algorithm [KGV83], sometimes called *Metropolis algorithm*, is a metaheuristic that can be adapted to solve a number of optimization problem.

Intuitively, it's based on the physical metaphor of the annealing process of solids: in order to obtain a matter configuration without defects, it has to be first heated to its melting point, then the temperature is slowly lowered and a long time is spent at temperatures near the freezing point.

In a similar fashion, we minimize by simulated annealing by considering the objective function as a sort of "energy" function of a system, and perturbating iteratively our current solution. Initially, a temperature parameter of the algorithm is high and even solutions that increments the energy of the solution are accepted: this enables the algorithm to escape local minima; then, as the temperature lowers, the probability to accept a solution that increments energy decrease, and all state transitions are bound to make the solution better.

The AMOSA algorithm [Ban+08c] is a simulated-annealing based multi-

objective optimization algorithm, that instead of keeping track and modifying a single solution, builds an estimate of the Pareto front of the feasible solution set of the problem by keeping track of an archive of the best encountered solutions.

Assume that at a given temperature T, the algorithm is in state q; a new state s is then created as a random perturbation in a neighborhood of q, and it's with probability

$$P_{qs} = \frac{1}{1 + e^{\frac{-(E(q,T)-E(s,T))}{T}}}$$

where the function $E(s,T)$ is an energy function that is evaluated as a function of the objective.

The pseudocode of the algorithm is presented in Figure B.4. The concept of *amount of domination* can be defined in many ways; the simples method, used in [Ban+08c], is to define it as a volume. As an example, consider the case of Figure B.3 where $\mathbf{f} = \{\mathbf{f_1}, \mathbf{f_2}\}$ and f_1 and f_2 have values in $[0,1]$; for solutions A, B, we define

$$\Delta dom_{a,b} = |f_1(a) - f_1(b)||f_1(b) - f_2(b)| \quad .$$

Algorithm AMOSA

Set *Tmax*, *Tmin*, *HL*, *SL*, *iter*, α, temp=*Tmax*.
Initialize the *Archive*.
current-pt = random(*Archive*). /* randomly chosen solution from the *Archive**/
while (temp > *Tmin*)
  for (i=0; i< iter; i++)
    new-pt=perturb(*current-pt*).
    Check the domination status of *new-pt* and *current-pt*.
    /* Code for different cases */
    if (*current-pt* dominates *new-pt*) /* Case 1*/
      $\Delta dom_{avg} = \frac{\left(\sum_{i=1}^{k} \Delta dom_{i,new-pt}\right) + \Delta dom_{current-pt,new-pt}}{(k+1)}$.
      /* *k*=total-no-of points in the *Archive* which dominate *new-pt*, $k \geq 0$. */
      prob= $\frac{1}{1+exp(\Delta dom_{avg} * temp)}$.
      Set *new-pt* as *current-pt* with probability=*prob*
    if (*current-pt* and *new-pt* are non-dominating to each other) /* Case 2*/
      Check the domination status of *new-pt* and points in the *Archive*.
      if (*new-pt* is dominated by k ($k \geq 1$) points in the *Archive*) /* Case 2(a)*/
        prob= $\frac{1}{1+exp(\Delta dom_{avg} * temp)}$.
        $\Delta dom_{avg} = \frac{\left(\sum_{i=1}^{k} \Delta dom_{i,new-pt}\right)}{k}$.
        Set *new-pt* as *current-pt* with probability=*prob*.
      if (*new-pt* is non-dominating w.r.t all the points in the *Archive*) /* Case 2(b)*/
        Set *new-pt* as *current-pt* and add *new-pt* to the *Archive*.
        if *Archive-size* > *SL*
          Cluster *Archive* to *HL* number of clusters.
      if (*new-pt* dominates k, ($k \geq 1$) points of the *Archive*) /* Case 2(c)*/
        Set *new-pt* as *current-pt* and add it to the *Archive*.
        Remove all the k dominated points from the *Archive*.
    if (*new-pt* dominates *current-pt*) /* Case 3 */
      Check the domination status of *new-pt* and points in the *Archive*.
      if (*new-pt* is dominated by k ($k \geq 1$) points in the *Archive*) /* Case 3(a)*/
        Δdom_{min} = minimum of the difference of domination amounts between the *new-pt* and the k points
        prob= $\frac{1}{1+exp(-\Delta dom_{min})}$.
        Set point of the archive which corresponds to Δdom_{min} as *current-pt* with probability=*prob*
        else set *new-pt* as *current-pt*.
      if (*new-pt* is non-dominating with respect to the points in the *Archive*) /* Case 3(b) */
        Set *new-pt* as the *current-pt* and add it to the *Archive*.
        if *current-pt* is in the *Archive*, remove it from the *Archive*.
        else if *Archive-size*> *SL*.
          Cluster *Archive* to *HL* number of clusters.
      if (*new-pt* dominates k other points in the *Archive*) /* Case 3(c)*/
        Set *new-pt* as *current-pt* and add it to the *Archive*.
        Remove all the k dominated points from the *Archive*.
  End for
temp= α*temp.
End while
if *Archive-size* > *SL*
  Cluster *Archive* to *HL* number of clusters.

Figure B.4. The AMOSA algorithm (from [Ban+08c])

Appendix C

Tools

C.1 Yosys

The Yosys Open Synthesis Suite [Wolb] is a framework for Verilog RTL synthesis.

It provides a number of front-ends, notably Verilog-2005 (with partial support for SystemVerilog), and corresponding backends; it can be used for synthesis and mapping, with support for ASIC standard cell libraries in Liberty format and some FPGA technologies, including Xilinx 7 Series.

It also provides *SAT*-based formal methods for checking properties and equivalence.

The Yosys manual [Wola] provides extensive documentation on design principles, usage and extension programming. In this section, we provide a number of simple usage cases.

C.1.1 Yosys basics

Yosys can be controlled by directly entering commands at the prompt, using synthesis script (simple text files that contain sequences of commands) or using Tcl scripts.

The sequences of commands we'll use can be adapted to be used in all of these cases; of course, Tcl scripts can be used to realize more complex synthesis scripts.

Reading an RTL file

The RTL description for a 2-bit multiplier is saved to the file `mult_2_bit.sv`.

```
// 2-bit multiplier

module mult_2_bit (
    input  logic [1:0]  a,
    output logic [3:0]  o,
    input  logic [1:0]  b
);

  logic and01;
  logic and10;
  logic and11;
  logic and0110;

  assign and01 = a[0] & b[1];
  assign and10 = a[1] & b[0];
  assign and11 = a[1] & b[1];
  assign and0110 = and01 & and10;

  assign o[0] = a[0] & b[0];
  assign o[1] = and01 ^ and10;
  assign o[2] = and11 ^ and0110;
  assign o[3] = and11 & and0110;

endmodule
```

The following script reads the file, does some basic checks on the design consistency and sets the module `mult_2_bit` as the top module:

```
  read_verilog -sv mult_2_bit.sv
  hierarchy -check -top mult_2_bit
```

Basic synthesis

The design can be mapped to the Yosys primitives using the following script:

```
  proc; opt; fsm; opt; memory; opt
  techmap; opt
```

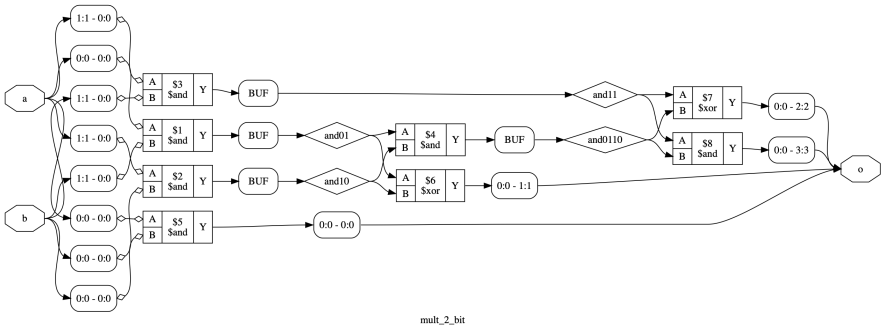


Figure C.1. 2-bit multiplier

The first line isn't actually required for a combinatorial-only design, and is used to translate processes, FSMs and memory to basic cells. The `techmap` command executes a simple technology mapping, using Verilog descriptions of cells included in Yosys. The `opt` passes execute optimization and cleanup of the design.

Generating schematics

The `show` command can be used, at any point, to graphically visualize the mapped circuit; for example, Figure C.1 shows the design after reading from file, and Figure C.2 after the synthesis script.

C.1.2 Technology mapping

Mapping to ASIC

The design can be mapped to ASIC, providing a cell library in Liberty format; for example, using the demonstration library:

```
1  library(demo) {
2  cell(BUF) {
3    area: 6;
4    pin(A) { direction: input; }
5    pin(Y) { direction: output;
6             function: "A"; }
7  }
```

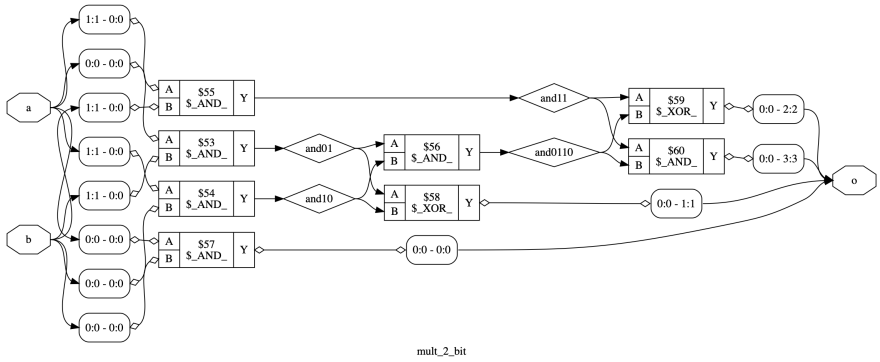


Figure C.2. 2-bit multiplier after technology mapping

```
8   cell(NOT) {
9     area: 3;
10    pin(A) { direction: input; }
11    pin(Y) { direction: output;
12             function: "A'"; }
13  }
14  cell(NAND) {
15    area: 4;
16    pin(A) { direction: input; }
17    pin(B) { direction: input; }
18    pin(Y) { direction: output;
19             function: "(A*B)'"; }
20  }
21  cell(NOR) {
22    area: 4;
23    pin(A) { direction: input; }
24    pin(B) { direction: input; }
25    pin(Y) { direction: output;
26             function: "(A+B)'"; }
27  }
28  cell(DFF) {
29    area: 18;
30    ff(IQ, IQN) { clocked_on: C;
31                  next_state: D; }
32    pin(C) { direction: input;
33             clock: true; }
34    pin(D) { direction: input; }
```

```
    pin(Q) { direction: output;
             function: "IQ"; }
  }
  cell(DFFSR) {
    area: 18;
    ff("IQ", "IQN") { clocked_on: C;
                 next_state: D;
                     preset: S;
                      clear: R; }
    pin(C) { direction: input;
                clock: true; }
    pin(D) { direction: input; }
    pin(Q) { direction: output;
             function: "IQ"; }
    pin(S) { direction: input; }
    pin(R) { direction: input; }
  }
}
```

and the following script:

```
dfflibmap -liberty cmos_cells.lib
abc -liberty cmos_cells.lib;;
write_verilog mult_2_bit_synth.v
```

Note that Yosys will require a Verilog description, such as the following, in order to be able to correctly interpret a synthesized Verilog file:

```
module BUF(A, Y);
  input A;
  output Y;
  assign Y = A;
endmodule

module NOT(A, Y);
  input A;
  output Y;
  assign Y = ~A;
endmodule

module NAND(A, B, Y);
  input A, B;
  output Y;
  assign Y = ~(A & B);
endmodule

```

```
19 module NOR(A, B, Y);
20   input A, B;
21   output Y;
22   assign Y = ~(A | B);
23 endmodule
24
25 module DFF(C, D, Q);
26   input C, D;
27   output reg Q;
28   always @(posedge C)
29     Q <= D;
30 endmodule
31
32 module DFFSR(C, D, Q, S, R);
33   input C, D, S, R;
34   output reg Q;
35   always @(posedge C, posedge S, posedge R)
36     if (S)
37       Q <= 1'b1;
38     else if (R)
39       Q <= 1'b0;
40     else
41       Q <= D;
42 endmodule
```

An output from synthesis can be read with a script like the following:

```
1   read_verilog mult_2_bit_synth.v
2   read_verilog -lib cmos_lib.v
```

Mapping to Xilinx 7 Series FPGA

The design can be mapped to Xilinx 7 Series FPGA and written to disk (for example, to be used for placement-and-routing using Xilinx tools), with the following script:

```
1   synth_xilinx
2   write_verilog mult_2_bit_xilinx.v
```

The produced output, stripped from comments from the tool, is the following:

```
1 module mult_2_bit(a, o, b);
2   input [1:0] a;
3   input [1:0] b;
```

```
4   output [3:0] o;
5
6   LUT2 #(
7     .INIT(4'h8)
8   ) _0_ (
9     .I0(a[0]),
10    .I1(b[0]),
11    .O(o[0])
12  );
13
14  LUT4 #(
15    .INIT(16'h7888)
16  ) _1_ (
17    .I0(a[0]),
18    .I1(b[1]),
19    .I2(a[1]),
20    .I3(b[0]),
21    .O(o[1])
22  );
23
24  LUT4 #(
25    .INIT(16'h7000)
26  ) _2_ (
27    .I0(b[0]),
28    .I1(a[0]),
29    .I2(a[1]),
30    .I3(b[1]),
31    .O(o[2])
32  );
33
34  LUT4 #(
35    .INIT(16'h8000)
36  ) _3_ (
37    .I0(a[0]),
38    .I1(b[1]),
39    .I2(a[1]),
40    .I3(b[0]),
41    .O(o[3])
42  );
43 endmodule
```

C.1.3 Formal Methods

Yosys provides an additional front-end, SimbiYosys, for more complex formal hardware verification flows. In this section, we provide only a simple example that makes use of the basic Yosys tool.

Checking combinatory equivalence

In order to check the combinatorial equivalence of the synthesis outputs, the following script creates an equivalence miter and checks it using a *SAT* solver:

```
read_verilog -sv mult_2_bit.sv
design -stash golden
read_verilog mult_2_bit_synth.v
design -stash gate
design -copy-from golden -as golden mult_2_bit
design -copy-from gate -as gate mult_2_bit
miter -equiv golden gate miter
hierarchy -check -top miter
flatten
sat -prove trigger 0
```

The resulting miter is shown in Figure C.3 and the tool output is the following:

```
Setting up SAT problem:
Final constraint equation: { } = { }
Imported 22 cells to SAT database.
Import proof-constraint: \trigger = 1'0
Final proof equation: \trigger = 1'0

Solving problem with 129 variables and 331 clauses..
SAT proof finished - no model found: SUCCESS!

                  /$$$$$$       /$$$$$$$$      /$$$$$$$
                 /$$__  $$     | $$_____/     | $$__  $$
                | $$  \ $$     | $$           | $$  \ $$
                | $$  | $$     | $$$$$        | $$  | $$
                | $$  | $$     | $$__/        | $$  | $$
                | $$/$$ $$     | $$           | $$  | $$
                |  $$$$$$/ /$$| $$$$$$$$ /$$| $$$$$$$//$$
                 \____ $$$|__/|________/|__/|_______/|__/
                      \__/
```

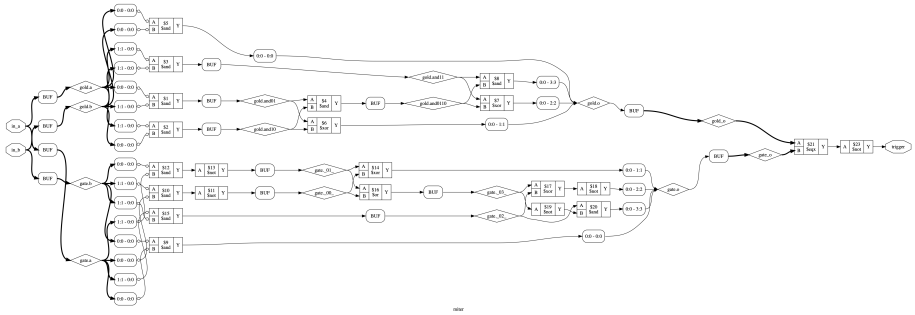


Figure C.3. 2-bit multiplier equivalence miter

C.2 Z3 SMT solver

SMT is the decision problem of the satisfiability of formulas with respect to decidable background theories [BT18].

Z3 [DB08] is a solver for SMT problems that provides high level bindings for general purpose programming languages and support for many background theories, such as integers, reals, Booleans and bitvectors.

Extensive tutorials exist for Z3, such as [Bjø+18].

C.2.1 A simple smt-lib example

Most of the commonly used SMT solvers, such as Z3 and Boolector [NPB15], can solve problems written in the smt-lib format [BST+10], a symbolic expression based syntax [McC59] for encoding assertions and instructions to the solver.

For example, the problem on the theories of Booleans and integers of finding a function $f : \mathbb{Z} \times \{0, 1\} \to \mathbb{Z}$ such that $f(x, 1) < 100$ when $x > 10$ and $f(x, 1) > 100$ when $x < 10$ can be encoded as

```
(declare-const a Int)
(declare-const b Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< b 10))
(assert (< (f a true) 100))
(assert (> (f b true) 100))
(check-sat)
```

```
(get-model)
```

and produces the following output from the solver

```
sat
(model
  (define-fun b () Int
    0)
  (define-fun a () Int
    11)
  (define-fun f ((x!0 Int) (x!1 Bool)) Int
    (ite (and (= x!0 11) (= x!1 true)) 0
    (ite (and (= x!0 0) (= x!1 true)) 101
      0)))
)
```

that reports that the problem is satisfiable and provides the model that's been found, i.e. a possible solution that satisfies the assertions.

C.2.2 Exact synthesis

The main motivating example for SMT solvers is the following Python script that, given a function specification and optionally a maximum Hamming distance, finds the size-optimum AIG that implements the function and writes it to disk in AIGER format [BHW11].

```
import math
import sys

from z3 import *

def truth_value(i, t):
    if i == 0:
        return 0
    return 1 if t % 2**i >= 2**(i-1) else 0

def function_semantics_constraints():
    ax = []
    if (ax_degree == 0):
        for t in range(len(fun_spec)):
            slv.add(b[-1][t] == Xor(Not(out_p), bool(fun_spec
                [t])))
    else:
        for t in range(len(fun_spec)):
            ax.append(Bool('ax_{}'.format(t)))
```

```
19              slv.add(ax[t] == Xor(b[-1][t], Xor(Not(out_p),
                    bool(fun_spec[t]))))
20          slv.add(AtMost(*ax, ax_degree))
21
22  if __name__ == '__main__':
23      if (len(sys.argv) != 4):
24          exit()
25
26      fun_spec = [0 if c == '0' else 1 for c in reversed(sys.
            argv[1])]
27      num_vars = math.ceil(math.log2(len(fun_spec)))
28      ax_degree = int(sys.argv[3])
29      assert 2**num_vars == len(fun_spec), "Incomplete␣
            specification"
30
31      # Entries of the truth table
32      slv = Solver()
33      b = []
34      for i in range(num_vars + 1):
35          b.append([])
36          for t in range(len(fun_spec)):
37              b[i].append(Bool('b_{}_{}'.format(i, t)))
38              slv.add(b[i][t] == bool(truth_value(i, t)))
39
40      # Input values, AIG structure and polarity
41      a = [[], []]
42      s = [[], []]
43      p = [[], []]
44
45      # Function semantics
46      out_p = Bool('p')
47      slv.push()
48      function_semantics_constraints()
49
50      while slv.check() == unsat:
51          # Update index
52          i = len(b)
53          i_gates = i - (num_vars + 1)
54
55          # Drop old function semantics constraints
56          slv.pop()
57
58          # Add lists for t entries for gate i
59          b.append([])
60          for c in range(2):
```

```
61              a[c].append([])
62
63          # Structure (no cycles, order, polarity)
64          for c in range(2):
65              s[c].append(Int('s_{}_{}'.format(c, i)))
66              slv.add(s[c][i_gates] < i, s[c][i_gates] >= 0)
67              p[c].append(Bool('p_{}_{}'.format(c, i)))
68
69          slv.add(s[0][i_gates] < s[1][i_gates])
70
71          for t in range(len(fun_spec)):
72              # AND functionality
73              b[i].append(Bool('b_{}_{}'.format(i, t)))
74              for c in range(2):
75                  a[c][i_gates].append(Bool('a_{}_{}_{}'.format
                        (c, i, t)))
76              slv.add(b[i][t] == And(a[0][i_gates][t], a[1][
                    i_gates][t]))
77
78              # Input connections
79              for j in range(i):
80                  for c in range(2):
81                      slv.add(Implies(s[c][i_gates] == j, a[c][
                            i_gates][t] == Xor(b[j][t], Not(p[c][
                            i_gates]))))
82
83          # Update function semantics
84          slv.push()
85          function_semantics_constraints()
86
87      num_gates = len(s[0])
88      print('Satisfied␣with␣{}␣gates'.format(num_gates))
89
90      # Write to AIGER text format
91      m = slv.model()
92      # File output removed for brevity
```

Bibliography

[AAH21] Waqar Ahmad, Berke Ayrancioglu, and Ilker Hamzaoglu. "Low Error Efficient Approximate Adders for FPGAs". In: *IEEE Access* 9 (2021). DOI: 10.1109/ACCESS.2021.3107370.

[AGM15] Luca Amaru, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. "The EPFL Combinational Benchmark Suite". en. In: *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*. Mountain View, California, USA, 2015. URL: http://infoscience.epfl.ch/record/207551.

[Ahm+23] Mohammad Hasan Ahmadilivani et al. "Special Session: Approximation and Fault Resiliency of DNN Accelerators". In: *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–10.

[AHS17] Muhammad Kamran Ayub, Osman Hasan, and Muhammad Shafique. "Statistical error analysis for low power approximate adders". In: *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM. 2017, p. 75.

[AKL16] Haider AF Almurib, T Nandha Kumar, and Fabrizio Lombardi. "Inexact designs for approximate low power addition by cell replacement". In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, pp. 660–665.

[AKL18] Haider A.F. Almurib, Thulasiraman Nandha Kumar, and Fabrizio Lombardi. "Approximate DCT Image Compression Us-

ing Inexact Computing". In: *IEEE Transactions on Computers* 67.2 (Feb. 2018), pp. 149–159. ISSN: 1557-9956. DOI: 10.1109/TC.2017.2731770.

[Ale97] Michael J. Alexander. "Power optimization for FPGA look-up tables". en. In: *Proceedings of the 1997 international symposium on Physical design - ISPD '97*. Napa Valley, California, United States: ACM Press, 1997, pp. 156–162. ISBN: 978-0-89791-927-2. DOI: 10.1145/267665.267707. URL: http://portal.acm.org/citation.cfm?doid=267665.267707 (visited on 03/07/2022).

[Ans+20] Mohammad Saeed Ansari et al. "Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.2 (Feb. 2020), pp. 317–328. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2019.2940943.

[Arm+22] Giorgos Armeniakos et al. "Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey". In: *ACM Comput. Surv.* 55.4 (2022). DOI: 10.1145/3527156.

[Ban+08a] Sanghamitra Bandyopadhyay et al. "A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA". In: *IEEE Transactions on Evolutionary Computation* 12.3 (June 2008), pp. 269–283. ISSN: 1941-0026. DOI: 10.1109/TEVC.2007.900837.

[Ban+08b] Sanghamitra Bandyopadhyay et al. "A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA". In: *IEEE Transactions on Evolutionary Computation* 12.3 (June 2008). Conference Name: IEEE Transactions on Evolutionary Computation, pp. 269–283. ISSN: 1941-0026. DOI: 10.1109/TEVC.2007.900837.

[Ban+08c] Sanghamitra Bandyopadhyay et al. "A simulated annealing-based multiobjective optimization algorithm: AMOSA". In: *IEEE transactions on evolutionary computation* 12.3 (2008), pp. 269–283.

[Bar+21] Salvatore Barone et al. "Multi-Objective Application-Driven Approximate Design Method". In: *IEEE Access* 9 (2021), pp. 86975–86993. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3087858.

[Bar+22a] Mario Barbareschi et al. "A Catalog-based AIG-Rewriting Approach to the Design of Approximate Components". In: *IEEE Transactions on Emerging Topics in Computing* (2022). DOI: 10.1109/TETC.2022.3170502.

[Bar+22b] Mario Barbareschi et al. "A Genetic-algorithm-based Approach to the Design of DCT Hardware Accelerators". In: *ACM Journal on Emerging Technologies in Computing Systems* 18.3 (July 2022), pp. 1–25. ISSN: 1550-4832, 1550-4840. DOI: 10.1145/3501772.

[Bar+24] Mario Barbareschi et al. "FPGA Approximate Logic Synthesis through Catalog-Based AIG-Rewriting Technique". In: *Journal of Systems Architecture* 150 (2024). DOI: 10.1016/j.sysarc.2024.103112.

[Bar+88] Karen A Bartlett et al. "Multi-level logic minimization using implicit don't cares". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 7.6 (1988), pp. 723–740.

[BAS08] Saad Bouguezel, M. Omair Ahmad, and M. N. S. Swamy. "Low-Complexity 8x8 Transform for Image Compression". In: *Electronics Letters* 44.21 (2008), pp. 1249–1250.

[BBM21] Mario Barbareschi, Salvatore Barone, and Nicola Mazzocca. "Advancing Synthesis of Decision Tree-Based Multiple Classifier Systems: An Approximate Computing Case Study". In: *Knowledge and Information Systems* (Apr. 2021), pp. 1–20. ISSN: 0219-3116. DOI: 10.1007/s10115-021-01565-5.

[BC12] F. M. Bayer and R. J. Cintra. "DCT-like Transform for Image Compression Requires 14 Additions Only". In: *Electronics Letters* 48.15 (2012), p. 919. ISSN: 00135194. DOI: 10.1049/el.2012.1148. arXiv: 1702.00817.

[Ben09] Yoshua Bengio. "Learning Deep Architectures for AI". In: *Foundations and Trends® in Machine Learning* 2.1 (Jan. 2009), pp. 1–127. ISSN: 1935-8237. DOI: 10.1561/2200000006. URL: https://doi.org/10.1561/2200000006 (visited on 08/23/2021).

[BHS90] Robert K Brayton, Gary D Hachtel, and Alberto L Sangiovanni-Vincentelli. "Multilevel logic synthesis". In: *Proceedings of the IEEE* 78.2 (1990), pp. 264–300.

[BHW11] Armin Biere, Keijo Heljanko, and Siert Wieringa. "AIGER 1.9 and beyond". In: *Available at fmv. jku. at/hwmcc11/beyond1. pdf* (2011).

[Bjø+18] Nikolaj Bjørner et al. "Programming Z3". In: *International Summer School on Engineering Trustworthy Software Systems*. Springer. 2018, pp. 148–201.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BM10a] Robert Brayton and Alan Mishchenko. "ABC: An Academic Industrial-Strength Verification Tool". In: *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.

[BM10b] Robert Brayton and Alan Mishchenko. "ABC: An academic industrial-strength verification tool". In: *International Conference on Computer Aided Verification*. Springer. 2010, pp. 24–40.

[Bow+09] Keith Bowman et al. "Circuit techniques for dynamic variation tolerance". In: *2009 46th ACM/IEEE Design Automation Conference*. IEEE. 2009, pp. 4–7.

[BR03] Christian Blum and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308.

[Bra06] Alan Mishchenko Robert Brayton. "Scalable logic synthesis using a simple circuit structure". In: *International Workshop on Logic and Synthesis*. Citeseer. 2006, pp. 15–22.

[Bra93] Daniel Brand. "Verification of large synthesized designs". In: *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. IEEE. 1993, pp. 534–537.

[BST+10] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. "The smt-lib standard: Version 2.0". In: *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*. Vol. 13. 2010, p. 14.

[BT18] Clark Barrett and Cesare Tinelli. "Satisfiability modulo theories". In: *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[Cas+21] Jorge Castro-Godínez et al. "AxLS: A Framework for Approximate Logic Synthesis Based on Netlist Transformations". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.8 (Aug. 2021), pp. 2845–2849. ISSN: 1558-3791. DOI: 10.1109/TCSII.2021.3068757.

[CD94] Jason Cong and Yuzheng Ding. "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.1 (1994), pp. 1–12.

[Češ+17] Milan Češka et al. "Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished". In: *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press. 2017, pp. 416–423.

[Cha+16a] Arun Chandrasekharan et al. "Approximation-Aware Rewriting of AIGs for Error Tolerant Applications". In: *Proceedings of the 35th International Conference on Computer-Aided Design*. Austin Texas: ACM, Nov. 2016, pp. 1–8. ISBN: 978-1-4503-4466-1. DOI: 10.1145/2966986.2967003.

[Cha+16b] Arun Chandrasekharan et al. "Approximation-aware rewriting of AIGs for error tolerant applications". In: *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM. 2016, p. 83.

[Chi+13] Vinay K Chippa et al. "Analysis and characterization of inherent application resilience for approximate computing". In: *Proceedings of the 50th Annual Design Automation Conference*. ACM. 2013, p. 113.

[Chu11] Pong P Chu. *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. John Wiley & Sons, 2011.

[DB08] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.

[DD97] I. Das and J. E. Dennis. "A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems". In: *Structural Optimization* 14.1 (Aug. 1997), pp. 63–69. ISSN: 0934-4373, 1615-1488. DOI: 10.1007/BF01197559.

[Deb+02a] K. Deb et al. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197. ISSN: 1941-0026. DOI: 10.1109/4235.996017.

[Deb+02b] K. Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002). Conference Name: IEEE Transactions on Evolutionary Computation, pp. 182–197. ISSN: 1941-0026. DOI: 10.1109/4235.996017.

[Deb+02c] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[Erc+89] S Ercolani et al. "Estimate of signal probability in combinational logic networks". In: *[1989] Proceedings of the 1st European Test Conference*. IEEE. 1989, pp. 132–138.

[EWT18] Jorge Echavarria, Stefan Wildermann, and Jürgen Teich. "Design space exploration of multi-output logic function approximations". In: *Proceedings of the International Conference on Computer-Aided Design*. ACM. 2018, p. 52.

[FMK91] Masahiro Fujita, Yusuke Matsunaga, and Taeko Kakuda. "On variable ordering of binary decision diagrams for the application of multi-level logic synthesis". In: *Proceedings of the conference on European design automation*. IEEE Computer Society Press. 1991, pp. 50–54.

[Gup+11] Vaibhav Gupta et al. "IMPACT: imprecise adders for low-power approximate computing". In: *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*. IEEE Press. 2011, pp. 409–414.

[HA22] Naofumi Homma and Takafumi Aoki. *Arithmetic Module Generator*. 2022. URL: https://www.ecsis.riec.tohoku.ac.jp/topics/amg/ (visited on 10/19/2021).

[Haa+16] Winston Jason Haaswijk et al. "LUT mapping and optimization for majority-inverter graphs". In: *Proceedings of the 25th International Workshop on Logic & Synthesis (IWLS)*. CONF. 2016.

[Haa+18] Winston Haaswijk et al. "Deep learning for logic optimization algorithms". In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2018, pp. 1–4.

[HBR15] Soheil Hashemi, R Bahar, and Sherief Reda. "DRUM: A dynamic range unbiased multiplier for approximate applications". In: *Proceedings of the IEEE/ACM international conference on computer-aided design*. IEEE Press. 2015, pp. 418–425.

[He+16] Kaiming He et al. "Deep Residual Learning for Image Recognition". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 01/05/2021).

[Hil12] Frederick S Hillier. *Introduction to operations research*. Tata McGraw-Hill Education, 2012.

[HJ12] Cheng-Shen Han and Jie-Hong Roland Jiang. "When boolean satisfiability meets gaussian elimination in a simplex way". In: *International Conference on Computer Aided Verification*. Springer. 2012, pp. 410–426.

[HLB06a] Tong-Yu Hsieh, Kuen-Jong Lee, and M.A. Breuer. "An Error-Oriented Test Methodology to Improve Yield with Error-Tolerance". In: *24th IEEE VLSI Test Symposium*. Apr. 2006, 6 pp.–135. DOI: 10.1109/VTS.2006.18.

[HLB06b] Tong-Yu Hsieh, Kuen-Jong Lee, and Melvin A Breuer. "An error-oriented test methodology to improve yield with error-tolerance". In: *24th IEEE VLSI Test Symposium*. IEEE. 2006, 6–pp.

[HO13] Jie Han and Michael Orshansky. "Approximate computing: An emerging paradigm for energy-efficient design". In: *2013 18th IEEE European Test Symposium (ETS)*. IEEE. 2013, pp. 1–6.

[HS99] Rajamohana Hegde and Naresh R Shanbhag. "Energy-efficient signal processing via algorithmic noise-tolerance". In: *Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No. 99TH8477)*. IEEE. 1999, pp. 30–35.

[HTR18] Soheil Hashemi, Hokchhay Tann, and Sherief Reda. "BLASYS: approximate logic synthesis using boolean matrix factorization". In: *Proceedings of the 55th Annual Design Automation Conference*. ACM. 2018, p. 55.

[HZ10] Alain Hore and Djemel Ziou. "Image quality metrics: PSNR vs. SSIM". In: *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, pp. 2366–2369.

[JG05] Zhigang Jiang and Sandeep K Gupta. "Threshold testing: Covering bridging and other realistic faults". In: *14th Asian Test Symposium (ATS'05)*. IEEE. 2005, pp. 390–397.

[JVR16] Shubham Jain, Swagath Venkataramani, and Anand Raghunathan. "Approximation through logic isolation for the design of quality configurable circuits". In: *2016 Design, Automation*

& Test in Europe Conference & Exhibition (DATE). IEEE. 2016, pp. 612–617.

[KB05] Randy H Katz and Gaetano Borriello. "Contemporary logic design". In: (2005).

[KGV83] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[Kim+16] Younghoon Kim et al. "Designing approximate circuits using clock overgating". In: *Proceedings of the 53rd Annual Design Automation Conference*. ACM. 2016, p. 15.

[Kim+22] Min Soo Kim et al. "The Effects of Approximate Multiplication on Convolutional Neural Networks". In: *IEEE Transactions on Emerging Topics in Computing* 10.2 (Apr. 2022), pp. 904–916. ISSN: 2168-6750. DOI: 10.1109/TETC.2021.3050989.

[KM22] Jan Klhufek and Vojtech Mrazek. "ArithsGen: Arithmetic Circuit Generator for Hardware Accelerators". In: *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. ISSN: 2473-2117. Apr. 2022, pp. 44–47. DOI: 10.1109/DDECS54261.2022.9770152.

[KNH10] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10 (canadian institute for advanced research)*. 2010. URL: https://www.cs.toronto.edu/~kriz/cifar.html (visited on 03/01/2023).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 08/27/2021).

[LCB98] Yann LeCun, Corinna Cortes, and Chris Burges. *MNIST Handwritten digit database*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (visited on 09/20/2021).

[LeC+89] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: https://doi.org/10.1162/neco.1989.1.4.541 (visited on 08/25/2021).

[Lew+05] David Lewis et al. “The Stratix II logic and routing architecture”. In: *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. ACM. 2005, pp. 14–20.

[LHL14] Cong Liu, Jie Han, and Fabrizio Lombardi. “A low-power, high-performance approximate multiplier with configurable partial error recovery”. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–4.

[Lin+11] Avinash Lingamneni et al. “Energy Parsimonious Circuit Design through Probabilistic Pruning”. In: *2011 Design, Automation Test in Europe*. Mar. 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763130.

[LL13] Chia-Hao Lin and Chao Lin. “High accuracy approximate multiplier with error correction”. In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE. 2013, pp. 33–38.

[Luk13] Sean Luke. *Essentials of Metaheuristics*. Second edition. Lulu, 2013. URL: http://cs.gmu.edu/$%5Csim$sean/book/metaheuristics/.

[Lup70] O. B. Lupanov. “On a Method of Circuit Synthesis”. In: *Journal of Symbolic Logic* 35.4 (1970), pp. 593–594. DOI: 10.2307/2271493.

[Maz+16] Sana Mazahir et al. “Probabilistic error modeling for approximate adders”. In: *IEEE Transactions on Computers* 66.3 (2016), pp. 515–530.

[Maz+19] Sana Mazahir et al. “Probabilistic Error Analysis of Approximate Adders and Multipliers”. In: *Approximate Circuits*. Springer, 2019, pp. 99–120.

[MCB06] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. "DAG-aware AIG rewriting a fresh look at combinational logic synthesis". In: *Proceedings of the 43rd annual Design Automation Conference*. ACM. 2006, pp. 532–535.

[McC59] John McCarthy. "Recursive functions of symbolic expressions and their computation by machine". In: (1959).

[MDS07] Leonardo de Moura, Bruno Dutertre, and Natarajan Shankar. "A tutorial on satisfiability modulo theories". In: *International Conference on Computer Aided Verification*. Springer. 2007, pp. 20–36.

[Mic94] Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[Mil18] B. Milewski. *Category Theory for Programmers*. Blurb, Incorporated, 2018. ISBN: 9780464825081. URL: https://books.google.it/books?id=ZaP-swEACAAJ.

[Mis+05] Alan Mishchenko et al. *FRAIGs: A unifying representation for logic synthesis and verification*. Tech. rep. ERL Technical Report, 2005.

[Mis+07a] Alan Mishchenko et al. "Combinational and Sequential Mapping with Priority Cuts". In: *2007 IEEE/ACM International Conference on Computer-Aided Design*. Nov. 2007, pp. 354–361. DOI: 10.1109/ICCAD.2007.4397290.

[Mis+07b] Alan Mishchenko et al. "Combinational and sequential mapping with priority cuts". In: *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*. IEEE Press. 2007, pp. 354–361.

[Mis+07c] Alan Mishchenko et al. "Combinational and sequential mapping with priority cuts". In: *2007 IEEE/ACM International Conference on Computer-Aided Design*. ISSN: 1558-2434. Nov. 2007, pp. 354–361. DOI: 10.1109/ICCAD.2007.4397290.

[Mis+15] Alan Mishchenko et al. "FRAIGs: A Unifying Representation for Logic Synthesis and Verification". In: *ERL Technical Report* (2015), p. 7.

[Mit16] Sparsh Mittal. "A survey of techniques for approximate computing". In: *ACM Computing Surveys (CSUR)* 48.4 (2016), p. 62.

[Mit20] Sparsh Mittal. "A survey of FPGA-based accelerators for convolutional neural networks". en. In: *Neural Computing and Applications* 32.4 (Feb. 2020), pp. 1109–1139. ISSN: 1433-3058. DOI: 10.1007/s00521-018-3761-1. URL: https://doi.org/10.1007/s00521-018-3761-1 (visited on 03/02/2021).

[Mor19] Alberto Moriconi. "FPGA Synthesis of Approximate Logic Circuits for Embedded Applications". MA thesis. University of Naples Federico II, 2019.

[Mor21] Alberto Moriconi. *Yosys-Als*. Apr. 2021. URL: https://github.com/albmoriconi/yosys-als.

[MPP11] Nasir Mohyuddin, Ehsan Pakbaznia, and Massoud Pedram. "Probabilistic error propagation in a logic circuit using the boolean difference calculus". In: *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer, 2011, pp. 359–381.

[Mra+17] V. Mrazek et al. "EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. ISSN: 1558-1101. Mar. 2017, pp. 258–261. DOI: 10.23919/DATE.2017.7926993.

[Mra+18] Vojtech Mrazek et al. "Scalable Construction of Approximate Multipliers With Formally Guaranteed Worst Case Error". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.11 (Nov. 2018), pp. 2572–2576. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2018.2856362.

[Mra+19] Vojtech Mrazek et al. "ALWANN: Automatic Layer-Wise Approximation of Deep Neural Network Accelerators without Retraining". In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (Nov. 2019). arXiv: 1907.07229, pp. 1–8. DOI: 10.1109/ICCAD45719.2019.8942068. URL: http://arxiv.org/abs/1907.07229 (visited on 12/24/2020).

[MV11] Pauli Miettinen and Jilles Vreeken. "Model order selection for boolean matrix factorization". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 51–59.

[MW17] Cody D Murray and R Ryan Williams. "On the (non) NP-hardness of computing circuit complexity". In: *Theory of Computing* 13.1 (2017), pp. 1–22.

[Nep+14] Kumud Nepal et al. "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits". In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–6.

[NPB14] Aina Niemetz, Mathias Preiner, and Armin Biere. "Boolector 2.0". In: *Journal on Satisfiability, Boolean Modeling and Computation* 9.1 (2014), pp. 53–58.

[NPB15] Aina Niemetz, Mathias Preiner, and Armin Biere. "Boolector 2.0 system description". In: *Journal on Satisfiability, Boolean Modeling and Computation* 9 (2015), pp. 53–58.

[Ped96] Massoud Pedram. "Power Minimization in IC Design: Principles and Applications". en. In: *ACM Transactions on Design Automation of Electronic Systems* 1.1 (1996), p. 54.

[PNP19] Ghasem Pasandi, Shahin Nazarian, and Massoud Pedram. "Approximate logic synthesis: A reinforcement learning-based technology mapping approach". In: *20th International Symposium on Quality Electronic Design (ISQED)*. IEEE. 2019, pp. 26–32.

[Pot+14] Uma Sadhvi Potluri et al. "Improved 8-Point Approximate DCT for Image and Video Compression Requiring Only 14 Additions". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.6 (June 2014), pp. 1727–1740. ISSN: 1558-0806. DOI: 10.1109/TCSI.2013.2295022.

[Pra+18] Bharath Srinivas Prabakaran et al. "DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems". In: *2018 Design, Automation Test in Europe*

Conference Exhibition (DATE). ISSN: 1558-1101. Mar. 2018, pp. 917–920. DOI: 10.23919/DATE.2018.8342140.

[Pra+20] Bharath Srinivas Prabakaran et al. "ApproxFPGAs: Embracing ASIC-Based Approximate Arithmetic Components for FPGA-Based Systems". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. ISSN: 0738-100X. July 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218533.

[Ram+13] Shankar Ganesh Ramasubramanian et al. "Relax-and-retime: A methodology for energy-efficient recovery based design". In: *Proceedings of the 50th Annual Design Automation Conference*. ACM. 2013, p. 111.

[Ran+14] Ashish Ranjan et al. "ASLAN: Synthesis of approximate sequential circuits". In: *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association. 2014, p. 364.

[Ran+19] Ashish Ranjan et al. "Automatic Synthesis Techniques for Approximate Circuits". In: *Approximate Circuits*. Springer, 2019, pp. 123–140.

[SAP18] Ilaria Scarabottolo, Giovanni Ansaloni, and Laura Pozzi. "Circuit Carving: A Methodology for the Design of Approximate Hardware". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2018, pp. 545–550. DOI: 10.23919/DATE.2018.8342067.

[Sch15] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". en. In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608014002135 (visited on 11/24/2020).

[SG10] Doochul Shin and Sandeep K Gupta. "Approximate logic synthesis for error tolerant applications". In: *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE. 2010, pp. 957–960.

[SG11] Doochul Shin and Sandeep K Gupta. "A new circuit simplification method for error tolerant applications". In: *2011 Design, Automation & Test in Europe*. IEEE. 2011, pp. 1–6.

[Sid+11] Stelios Sidiroglou-Douskos et al. "Managing performance vs. accuracy trade-offs with loop perforation". In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM. 2011, pp. 124–134.

[Soe+16] Mathias Soeken et al. "BDD minimization for approximate computing". In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2016, pp. 474–479.

[Soe+17a] Mathias Soeken et al. "Exact Synthesis of Majority-Inverter Graphs and Its Applications". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.11 (Nov. 2017), pp. 1842–1855. ISSN: 1937-4151. DOI: 10.1109/TCAD.2017.2664059.

[Soe+17b] Mathias Soeken et al. "Exact Synthesis of Majority-Inverter Graphs and Its Applications". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.11 (Nov. 2017), pp. 1842–1855. ISSN: 1937-4151. DOI: 10.1109/TCAD.2017.2664059.

[Soe+17c] Mathias Soeken et al. "Exact synthesis of majority-inverter graphs and its applications". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.11 (2017), pp. 1842–1855.

[Soe+18] Mathias Soeken et al. "Practical exact synthesis". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 309–314.

[SR10] O Sarbishei and Katarzyna Radecka. "Analysis of precision for scaling the intermediate variables in fixed-point arithmetic circuits". In: *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press. 2010, pp. 739–745.

[SVM19] Lukas Sekanina, Zdenek Vasicek, and Vojtech Mrazek. "Automated Search-Based Functional Approximation for Digital Circuits". In: *Approximate Circuits*. Ed. by Sherief Reda and Muhammad Shafique. Cham: Springer International Publishing, 2019, pp. 175–203. DOI: 10.1007/978-3-319-99322-5_9.

[Tas+20] Zois-Gerasimos Tasoulas et al. "Weight-Oriented Approximation for Energy-Efficient Neural Network Inference Accelerators". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.12 (Dec. 2020), pp. 4670–4683. ISSN: 1558-0806. DOI: 10.1109/TCSI.2020.3019460.

[Tia+15] Ye Tian et al. "Approxma: Approximate memory access for dynamic precision scaling". In: *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM. 2015, pp. 337–342.

[Ull+22a] Salim Ullah et al. "AppAxO: Designing Application-specific Approximate Operators for FPGA-based Embedded Systems". en. In: *ACM Transactions on Embedded Computing Systems* (Feb. 2022), p. 3513262. ISSN: 1539-9087, 1558-3465. DOI: 10.1145/3513262. URL: https://dl.acm.org/doi/10.1145/3513262 (visited on 03/02/2022).

[Ull+22b] Salim Ullah et al. "High-Performance Accurate and Approximate Multipliers for FPGA-Based Hardware Accelerators". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.2 (Feb. 2022), pp. 211–224. ISSN: 1937-4151. DOI: 10.1109/TCAD.2021.3056337.

[UMK18] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar. "*SMApproxlib*: library of FPGA-based approximate multipliers". en. In: *Proceedings of the 55th Annual Design Automation Conference*. San Francisco California: ACM, June 2018, pp. 1–6. ISBN: 978-1-4503-5700-5. DOI: 10.1145/3195970.3196115. URL: https://dl.acm.org/doi/10.1145/3195970.3196115 (visited on 01/24/2022).

[UNT17] Rei Ueno, Naofumi Homma, and Takafumi Aoki. "Automatic Generation System for Multiple-Valued Galois-Field Parallel Multipliers". en. In: *IEICE Transactions on Information and Systems* E100.D.8 (2017), pp. 1603–1610. ISSN: 0916-8532, 1745-1361. DOI: 10.1587/transinf.2016LOP0010. URL: https://www.jstage.jst.go.jp/article/transinf/E100.D/8/E100.D_2016LOP0010/_article (visited on 11/29/2021).

[Val79] Leslie G Valiant. "The complexity of computing the permanent". In: *Theoretical computer science* 8.2 (1979), pp. 189–201.

[VBI08] Ajay K Verma, Philip Brisk, and Paolo Ienne. "Variable latency speculative addition: A new paradigm for arithmetic circuit design". In: *Proceedings of the conference on Design, automation and test in Europe*. ACM. 2008, pp. 1250–1255.

[Ven+12] Swagath Venkataramani et al. "SALSA: systematic logic synthesis of approximate circuits". In: *DAC Design Automation Conference 2012*. IEEE. 2012, pp. 796–801.

[Vol13] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 2013.

[VRR13] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits". In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2013, pp. 1367–1372.

[WD92] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[WG13] Clifford Wolf and Johann Glaser. "Yosys - A Free Verilog Synthesis Suite". In: *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*. (2013), p. 6.

[WM97] D.H. Wolpert and W.G. Macready. "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1941-0026. DOI: 10.1109/4235.585893.

[Wola] Clifford Wolf. *Yosys manual*. http://www.clifford.at/yosys/files/yosys_manual.pdf.

[Wolb] Clifford Wolf. *Yosys Open SYnthesis Suite*. http://www.clifford.at/yosys/.

[WQ16] Yi Wu and Weikang Qian. "An efficient method for multi-level approximate logic synthesis under error rate constraint". In: *Proceedings of the 53rd Annual Design Automation Conference*. ACM. 2016, p. 128.

[XMK15] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. "Approximate computing: A survey". In: *IEEE Design & Test* 33.1 (2015), pp. 8–22.

[Yan+13] Zhixi Yang et al. "Approximate XOR/XNOR-based adders for inexact computing". In: *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*. IEEE. 2013, pp. 690–693.

[Yan91a] Saeyang Yang. *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0*. Citeseer, 1991.

[Yan91b] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer, 1991.

Author's publications

1. M. Barbareschi, S. Barone, N. Mazzocca, A. Moriconi, A catalog-based AIG-rewriting approach to the design of approximate components, IEEE Transactions on Emerging Topics in Computing, 11(1), 70-81, 2022, DOI: 10.1109/TETC.2022.3170502

2. A. Amendola, M. Barbareschi, S. De Simone, G. Mezzina, A. Moriconi, C. L. Saragaglia, D. Serra, D. De Venuto, A real-time vital control module to increase capabilities of railway control systems in highly automated train operations Real Time Systems, 59(4), 636-661, 2023, DOI: https://doi.org/10.1007/s11241-023-09401-5

3. M. Barbareschi, S. Barone, N. Mazzocca, A. Moriconi, FPGA approximate logic synthesis through catalog-based AIG-rewriting technique, Journal of Systems Architecture, 150, 103112, 2022, DOI: https://doi.org/10.1016/j.sysarc.2024.103112

4. M. Barbareschi, S. Barone, V. Casola, P. Montone, A. Moriconi, A Memory Protection Strategy for Resource Constrained Devices in Safety Critical Applications 2022 6th International Conference on System Reliability and Safety (ICSRS), Venice, Italy, Oct. 2022, pp. 533-538, IEEE, DOI: 10.1109/ICSRS56243.2022.10067350

5. G. Mezzina, A. Amendola, M. Barbareschi, S. De Simone, G. Mascellaro, A. Moriconi, C. L. Saragaglia, D. Serra, D. De Venuto A Step Toward Safe Unattended Train Operations: A Pioneer Vital Control Module. 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)), Antwerp, Belgium, Apr. 2023, pp. 1-4, IEEE, DOI: 10.23919/DATE56975.2023.10137186

6. G. Mezzina, C. L. Saragaglia, M. Barbareschi, D. Serra, S. De Simone, A. Moriconi, D. De Venuto Model-Based Vital Control Architecture for Highly

Automated Train Operations2022 International Conference on Applications in Electronics Pervading Industry, Environment and Society Genova, Italy, Sept. 2023, pp. 163-170, Springer Nature Switzerland, DOI: https://doi.org/10.1007/978-3- 031-30333-3_21

7. M. Barbareschi, S. Barone, N. Mazzocca, A. Moriconi, Design Space Exploration Tools In: A. Bosio, D.Ménard, O. Sentieys (eds) Approximate Computing Techniques: From Component- to Application-Level, 215-259, Springer, Cham, 2022, DOI: https://doi.org/10.1007/978-3-030-94705-7_8